Iterative Guessing Random Additive Noise Decoder for Universal Decoding of Product Codes

Arslan Riaz, *Member, IEEE*, Kevin Galligan, *Graduate Student Member, IEEE*, Alperen Yasar, *Graduate Student Member, IEEE*, Vaibhav Bansal, *Student Member, IEEE*, Ken R. Duffy, *Senior Member, IEEE*, Muriel Medard, *Fellow, IEEE*, and Rabia Tugce Yazicigil, *Senior Member, IEEE*

***Abstract*— A fully integrated hardware design of the universal maximum likelihood Guessing Random Additive Noise Decoding (GRAND) algorithm implemented in 40 nm CMOS is presented. It is shown how this integrated hard-detection decoder, which is designed to process component codes of up to 128 bits in length, can be extended to efficiently decode product codes as long as 16,384 bits using the Iterative GRAND (IGRAND) algorithm. Pipelined stages provide throughput gain and dynamic energy savings when channel noise conditions improve. The chip allows for decoding product codes with two distinct component codes due to its ability to interleave between two codebooks without any switch-over time. Measurements demonstrate the decoder's accuracy and efficiency in decoding a broad selection of product codes, including the capacity-achieving random linear product codes. The chip consumes an average energy of 30.6 pJ/b with a latency of 1.04 μs when decoding the BCH(127, 106, 7) component code at 68 MHz from 1.1 V at a bit flip probability of 10^{-5}. Using a single chip to decode a BCH(127, 106, 7)^2 product code which results in 16,129-bit code of rate 0.68, we demonstrate an average energy consumption of 61.2 pJ/b with an average latency of 265 μs for the same operating conditions.**

***Index Terms*— Hard-input decoding, GRAND, IGRAND, product-codes, universal.**

I. Introduction

ERROR correction codes (ECCs) identify and correct the corrupted bits of information conveyed over an imperfect, noisy channel for applications such as data storage and data communication. To do this, ECC algorithms add redundancy to the transmitted message, which poses an overhead to the channel bandwidth, but in return, enables the correction of errors. Since the field of channel coding was launched in 1948 [1], ECCs have typically been designed in tandem with a decoding algorithm as the problem of decoding an arbitrary ECC is algorithmically hard [2]. In exchange for being restricted to a certain class of ECC, a decoder can leverage the algebraic structure of that class to achieve algorithmically efficient decoding. For this reason, many different ECCs and corresponding decoding algorithms have emerged, such as Bose-Chaudhuri-Hocquenghem (BCH) codes and the Berlekamp-Massey decoder [3], Reed-Muller (RM) codes [4] with Majority Logic Decoding, and Reed-Solomon codes [5]. Some of these codes such as cyclic redundancy check (CRC) [6], [7] are typically only used for error detection. The hardware implementation of these decoders is tightly coupled to a specific algorithm and a specific code, which reduces flexibility in the design of coding schemes. This requires distinct pieces of hardware for different levels of redundancy leading to high resource utilization and lower energy efficiency.

Guessing Random Additive Noise Decoding (GRAND) is a recently introduced decoding algorithm with a distinctive perspective that challenges the limitations of traditional ECCs [8]. Instead of extracting the correct codeword based on a specific codebook structure, GRAND focuses on identifying the effect that the noise had on the transmitted message by guessing error patterns in order of decreasing likelihood. This noise-centric approach allows GRAND to decode any moderate redundancy code establishing itself as a universal decoder that yields a maximum-likelihood (ML) decoding [8]. This feature of GRAND enables the error correction for CRCs and the decoding of Random Linear Codes. Unlike many traditional ECC decoders, both hard-detection and soft-detection variants of GRAND have been developed [9], [10], [11], [12], [13], [14], [15].

The simplicity, parallelizability, and low complexity of the GRAND algorithm make it appealing for high-throughput and energy-efficient hardware implementations. Previously, we implemented the first integrated chip for the hard-input variant of the GRAND algorithm [16], designed to support code lengths up to 128 bits and corrects errors up to a Hamming weight of 3 bits. Effective operation in a more challenging channel, where the signal-to-noise ratio (SNR) is low, requires the use of longer codes with higher redundancy. In [17], we introduced the Iterative GRAND (IGRAND) algorithm for decoding a class of long, high-redundancy codes known as product codes [18], which are used in modern

Received 25 December 2024; revised 6 March 2025, 22 April 2025, and 14 May 2025; accepted 14 May 2025. This work was supported in part by the Battelle Grant Low-Probability-of-Detect/Intercept Communications Employing Peaky Frequency-Shift-Key Modulation under Grant PO US0011-0000743557, in part by the Science Foundation Ireland under Grant 18/CRT/6049, in part by the Defense Advanced Research Projects Agency (DARPA) under Grant HR00112120008, and in part by the National Science Foundation Division of Electrical, Communications and Cyber Systems (ECCS) under Award 2128517 and Award 2128555. This article was recommended by Associate Editor L. Gan. *(Corresponding author: Arslan Riaz.)*

Arslan Riaz, Alperen Yasar, Vaibhav Bansal, and Rabia Tugce Yazicigil are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA (e-mail: arslanr@bu.edu; rty@bu.edu).

Kevin Galligan is with the Hamilton Institute, Maynooth University, Maynooth, W23 F2H6 Ireland.

Ken R. Duffy is with the Department of Electrical and Computer Engineering and the Department of Mathematics, Northeastern University, Boston, MA 02115 USA.

Muriel Medard is with Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Digital Object Identifier 10.1109/TCSI.2025.3570774

applications such as optical transport networks [19], [20], [21] and data storage [22], [23]. Through simulations, we demonstrated the decoding performance for product codes up to a length of 961 bits. In this work, we implement the IGRAND algorithm in hardware by combining our custom-designed GRAND chip with an FPGA to evaluate the measured decoding performance for product codes up to 16,384 bits in length. We also architected and synthesized the IGRAND algorithm in 40 nm CMOS as a fully integrated product code decoder with the GRAND chip to provide an area and power estimation of the IGRAND implementation. Additionally, we describe the architectural features of the GRAND chip that support an efficient implementation of IGRAND and various design trade-offs, including the use of an SRAM-based syndrome multiplier, parallelization factor, number of pipeline stages, and FIFO sizing for error generators, to obtain optimal performance for the component code decoding. We present measured decoding performance, energy efficiency, and latency for a wide range of codebooks for both component and product codes, including CA-Polar codes, CRC codes, BCH and extended BCH codes, and random linear product codes (RLPCs), a class of codes that was proven to be capacity-achieving in [17]. Finally, we use the reconfigurability of the decoder to decode only the columns of a product code instead of both columns and rows decoding to save energy and latency at higher SNR and analyze the trade-off between decoding accuracy and efficiency.

While product codes with any component code can be decoded with GRAND following Elias's decoding algorithm [18], here we will show that enhanced decoding performance with reduced decoding complexity is possible through the use of IGRAND that leverages the inherent features of GRAND. As the component code decoder of these algorithms, the GRAND chip can decode product codes of up to $128^2 = 16384$ bits in length and with thousands of redundant bits, expanding its range of possible applications. Fig. 1 illustrates the expansion of the achievable code lengths and rates when the GRAND chip is used to decode product codes. This exemplifies the architecture's adaptability, demonstrating its application across various use cases. In addition to expanding the code length by using the GRAND chip as a component code decoder of product codes, we also demonstrate that the maximum error-correcting capability of the decoder increases when bounded distance iterative decoding is employed. The GRAND chip decodes the component code errors up to a maximum Hamming weight of 3 bits, however by employing it iteratively as a row and column decoder of a product code, component codes experiencing substantially more than 3-bit errors can also be decoded. This capability allows the GRAND chip to continue operating effectively under more challenging, high-noise channel conditions.

In section II, preliminaries are detailed on the GRAND decoding theory, product codes, and the IGRAND algorithm. In section III, we describe the architecture and design choice of the GRAND chip. In section IV, we present the performance results of the GRAND chip when used for the component codes and iterative decoding of product codes. In section V, we present concluding remarks.

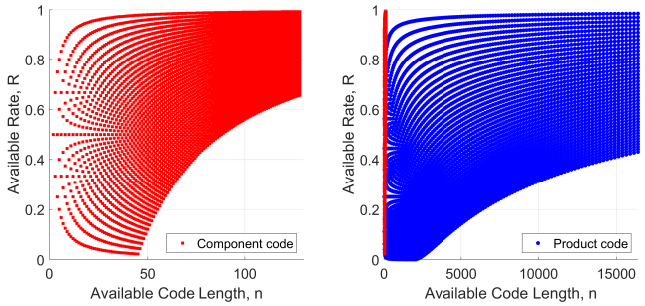


Fig. 1. The coding space of the GRAND chip with respect to code length and rate available supported by the GRAND chip, using linear codes. The area in red indicates the component code space for the GRAND chip with a code length of up to $n = 128$, while the blue area depicts the expansion of the coding space when product codes of up to $n^2 = 16384$ bits are used with linear component codes.

II. Background

A. GRAND Algorithm

Consider a binary linear codebook $\mathcal{C}$ with a code length of n and k information bits, consisting of 2^k strings in $\{0,1\}^n$. For each member of the codebook $X^n \in \mathcal{C}$, the following holds:

$$H \cdot X^n = 0, \quad \forall X^n \in \mathcal{C}, \tag{1}$$

where H is the parity check matrix of the code. During transmission, X^n is altered by an additive random binary noise sequence N^n, resulting in the received sequence

$$Y^n = X^n \oplus N^n, \tag{2}$$

where $\oplus$ represents element-wise binary addition. To guess the most likely transmitted X^n from Y^n, GRAND initially performs a membership check for $Y^n \in \mathcal{C}$. The membership check for a linear codebook can be performed by computing the product of its parity check matrix H with the received sequence:

$$H \cdot Y^n = \cancelto{0}{H \cdot X^n} \oplus H \cdot N^n = H \cdot N^n, \tag{3}$$

where the second equality follows from (1). The product $H \cdot Y^n$ is zero if and only if the sequence is a codeword belonging to that codebook. If $H \cdot Y^n$ is 0, then either N^n is zero, which means the transmission was not affected by errors, or Y^n maps to another valid codeword in $\mathcal{C}$, which can be considered as a false positive. If $H \cdot Y^n$ is non-zero, GRAND generates binary noise guesses E^n from most to least likely, according to the channel model. For each guess, GRAND calculates the product $H \cdot E^n$ and compares it to $H \cdot Y^n$. The first sequence E^n that satisfies the membership check $H \cdot Y^n = H \cdot E^n$ is the maximum-likelihood estimate of the true noise effect. $Y^n \ominus E^n$ is then the maximum-likelihood decoded output.

In a binary symmetric channel (BSC), GRAND generates noise sequences in order of increasing Hamming weight, breaking ties arbitrarily. As the Hamming weight increases, so does the decoding complexity, as there are $\binom{n}{i}$ potential error patterns for each Hamming weight i. However, in general, after 2^{n-k} noise guesses have been made, it becomes more likely that the decoder finds a sequence that satisfies the false positive membership check [8]. Therefore the search can

u_1	u_2	u_3	p_1	p_2
u_4	u_5	u_6	p_3	p_4
p_5	p_6	p_7	p_8	p_9

Fig. 2. A product code with a $[5, 3]$ row code and a $[3, 2]$ column code, where $[n, k]$ denotes a binary linear code of length n with k information bits. The information bits u_i are arranged as a 2×3 array. The rows of this array are extended by encoding them with the $[5, 3]$ code, producing parity bits p_j for $1 \leq j \leq 4$. Then the columns are encoded with the $[3, 2]$ code to produce the remaining parity bits. The resulting product code has a code rate of $2/5$, making it lower-rate and having more error-correcting capability than its rate-$3/5$ and rate-$2/3$ component codes [17].

be abandoned after 2^{n-k} guesses. In such cases, the decoder can report a decoding failure and request that the codeword be re-transmitted. This reduces complexity in terms of the number of guesses made and avoids erroneous decodings.

The GRAND decoder chip [16] presented in this work is capable of decoding codewords up to a length of 128 bits with code rates between 0.66-1. It can correct error patterns up to a Hamming weight of 3 bits, after which it abandons the search if the codeword is not found. In a BSC with a bit error rate of 10^{-3}, the probability of more than 3-bit errors affecting a 128-bit codeword is only 9.66×10^{-6}. Therefore, we choose the threshold of flipping up to only 3 bits at maximum. The GRAND decoder has previously demonstrated high decoding accuracy for various codebook families [16], including Reed-Muller (RM), BCH, CRC, Polar, CRC-Aided Polar (CA-Polar), and RLCs. It will be shown that the GRAND chip can also efficiently and accurately decode product codes as demonstrated in Section IV.

B. Product Codes and Their Construction

Product codes are encoded by arranging information bits in a 2-dimensional array. The rows of the array are extended by encoding each of them with a systematic, binary linear code. The columns are then extended by encoding each of them with another, possibly distinct, systematic code. The resulting matrix is a codeword of the product code. The codebook used for encoding rows or columns is called the component code. All rows and columns of the matrix are codewords of their respective component codes. See Fig. 2 for an example. We use $C(n, k, d_{\min})^2$ to denote a product code whose row and column codes are of type C, length n, have k information bits, and have a minimum Hamming distance of $d_{\min}$ between two valid codewords. The code rate of the component code is given as $R = k/n$. This product code has length n^2 and code rate of R^2.

C. Decoding Algorithms for Product Codes

There has been continuing interest in decoding algorithms for product codes since Elias's work [24], [25], [26], [27]. Though product codes are longer than their component codes and can correct more errors, they can still be decoded efficiently due to their structure. Elias proposed decoding each of the columns individually and then decoding each of the rows individually. Multiple iterations of this process, or *iterative decoding*, lead to better decoding accuracy than a single iteration [28].

A significant cause of error in iterative decoding is miscorrection, where new errors are introduced due to incorrect decoding of a row or column. Bounded Distance Decoding (BDD) [25], [26] avoids miscorrection by rejecting a component code decoding if the number of corrected errors is greater than some *distance bound*. Distance bound is defined as the maximum number of bits that can be flipped in the component code. Flipping a higher number of bits has a higher chance of a miscorrection [8]. BDD improves accuracy because decodings that correct fewer errors are more likely to be correct. Recently, Häger and Pfister [27] introduced anchor decoding, a complementary decoding technique to BDD that avoids miscorrection by declining to modify already-decoded components until they have been contradicted sufficiently many times. Al-Dweik and Sharif [25] proposed to apply BDD with a fixed bound of $t - 1$ in the first iteration, where t is the number of errors that the component code can correct, and then discard the bound in following iterations. This reduces the chance of miscorrections as only few bits are changed in the first iteration. Capacity-approaching soft-input decoders have been developed for product codes [24], but here we consider hard-input decoding only for applications where either the soft information is unavailable or cannot be processed due to system limitations. For instance, in data storage systems, there is no source of soft information from memory cells. Similarly, several optical communication standards also use codes designed for hard detection decoding only due to latency constraints. Consequently, hard-detection decoding is employed in these scenarios to meet application requirements.

D. IGRAND Algorithm

Iterative decoding algorithms for product codes assume the availability of a decoder for each of the component codes. GRAND, being a universal decoder, can be used as the sole decoder for any combination of component codes, reducing hardware footprint. Iterative GRAND (IGRAND) [17] is an iterative BDD algorithm that adapts GRAND for the efficient and accurate decoding of product codes. IGRAND improves decoding accuracy by enforcing BDD on GRAND, while also reducing search complexity. IGRAND distinguishes itself from other iterative BDD algorithms by not using a fixed distance bound. Instead, it starts with the lowest possible bound and increases it dynamically only when doing so is necessary to advance the decoding. For efficient decoding, IGRAND tracks the status of each row and column of the product code to avoid redundantly decoding an already-decoded component code. The status flag is initialized with a value of 0 for each row and column that needs to be decoded. The flag changes to 1 if the component code is successfully decoded and takes a value of 2 upon failure to decode for the given distance bound. If the decoding of one component changes a bit in another component, the status flag of the latter is changed to 0 as the decoding is no longer valid, and it must be decoded again. Upon increasing the distance bound, all the status flags with a value of 2 are changed to 0 since it might be possible to decode them with a higher distance bound. Fig. 3 illustrates an example of iterative BDD with a distance bound of one

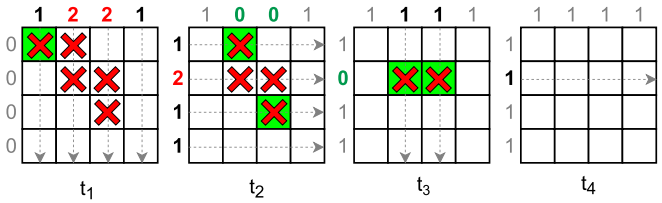


Fig. 3. This figure illustrates the iterative BDD of a product code of a 16-bit length with a distance bound of 1, indicating only a single error can be corrected at a time. The red crosses indicate an error, the dotted grey lines indicate whether columns or rows are being decoded, and a green background indicates an error is about to be corrected in the current iteration. The column and row flag values are shown on the top and left of each block. One-by-one all errors are eventually corrected by correcting one bit at a time in a row or a column [17].

bit. Reference [17] explains the IGRAND algorithm in more detail.

In addition to efficient decoding of product codes that use the same codebook for both row and column encoding, the GRAND chip also enables the decoding of product codes that may have row and column codebooks of unequal length or rate, as used in [29] and [30]. This will further increase the coding space and hence the system's flexibility for a wide range of applications.

III. GRAND Hardware Architecture

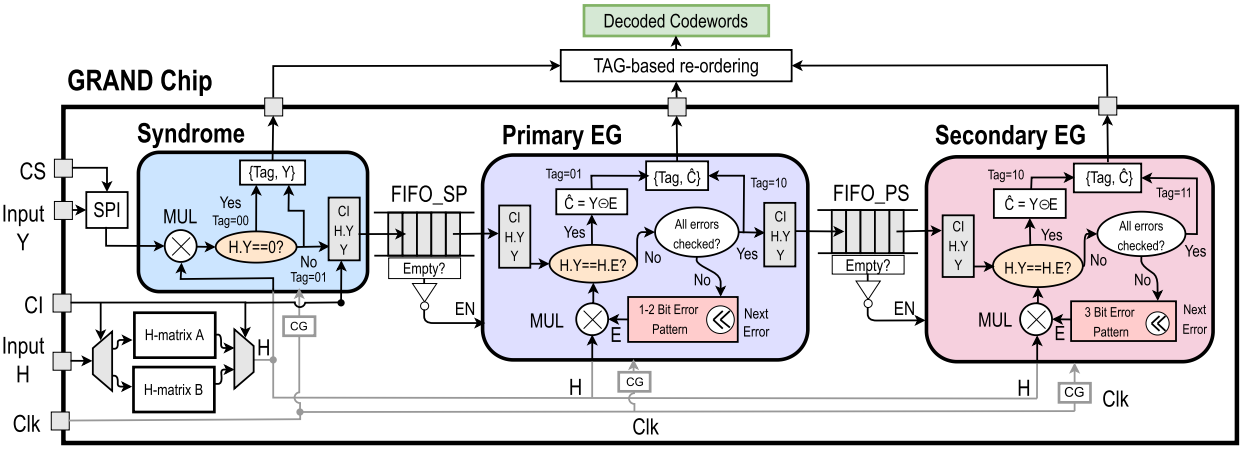
The GRAND chip architectural flow diagram is shown in Fig. 4. It receives the demodulated channel output Y^n and checks for codebook membership. This is done by computing the syndrome $H \cdot Y^n$. Thus, the first system block is the syndrome block for checking codebook membership. If the syndrome check fails, it passes the product $H \cdot Y^n$ to the error generator. The error generator produces putative noise patterns E^n in order of increasing Hamming weight and checks if the product $H \cdot E^n$ is equal to $H \cdot Y^n$. If the equality holds, then the noise pattern E^n is subtracted from Y^n to produce the maximum likelihood decoded output.

A. Syndrome

The syndrome product $H \cdot Y^n$ can be computed by parallel register file computation where $n \times (n-k)$ registers hold the H-matrix. The vector Y^n is multiplied with H using $n \times (n-k)$ AND gates, and the resulting bits for each row multiplication are XORed together to get the syndrome of length $n-k$. This operation takes a single cycle to perform, but requires a large number of gates and register files that consume high power and area. With a trade-off in latency, we can use SRAM cells to store the H matrix and perform the multiplication serially. By using $n \times (n-k)$ bits of SRAM, we take one column of $n-k$ bits and multiply it with one bit of Y^n using only $n-k$ AND gates. The output is XORed in a cyclic fashion to produce the partial product. This process is repeated until all the columns of SRAM are exhausted. In this case, the latency in cycles, will be equal to the number of columns of H matrix. The latency can be halved by using a dual-port SRAM that will provide two columns of the H matrix at a time as shown in Fig. 5. The comparison for latency, power, and energy consumption for syndrome computation using single-port SRAM, dual-port SRAM, and register file computation is shown in Fig. 5 for synthesized results at 68 MHz using 40 nm low-power CMOS technology with a nominal supply voltage of 1.1 V. The register file computation provides the lowest energy consumption but comes with a trade-off in power and area that is 10× and 47.4× higher than dual-port SRAM implementation, respectively. Although it incurs a latency trade-off, the dual-port SRAM implementation significantly reduces area and power consumption compared to register file computation, making it ideal for low-power IoT devices. It has a similar power consumption but lower latency and higher energy efficiency than the single-port SRAM implementation. Thus, we use the dual port SRAM for the computation. Before the decoding process, the parity check matrix H corresponding to a particular codebook is loaded onto the dual-port SRAM. For a maximum code length n of 128 bits and a minimum code rate R of 0.66 supported by the chip, the dimension of H is 44×128. Upon receiving the codeword which is a 128×1 vector Y, the syndrome computes the product $H \cdot Y$ by matrix multiplication which takes 64 cycles using the H matrix from the dual port SRAM.

B. Primary and Secondary Error Generator

If the membership check in the syndrome block fails, the chip will generate rank-ordered binary noise patterns from 1- to 3-bit flips. Each error sequence is multiplied by the parity check matrix H to calculate the product $H \cdot E^n$. We separate the syndrome block from the error generator (EG), forming a two-staged pipelined architecture with a FIFO in between. To increase the throughput, codewords that fail the membership check are pushed into the FIFO to be decoded by the EG independently, and the syndrome block moves on to the next codeword for membership checking. If the FIFO is empty during the decoding process when the channel noise is low, the EG remains in sleep mode to save power. The number of queries required to check the error patterns for a given Hamming weight i is given by $\binom{n}{i}$. For a Hamming weight of 3, testing all the 3-bit error patterns requires generating 341,376 noise patterns which is 41 times higher than both 1- and 2-bit errors combined. A worst-case greater than 3-bits error is not very likely but it will stall the EG until it is processed or abandoned. Therefore, the EG is further split into two stages: primary EG for testing 1- and 2-bit error patterns and secondary EG for testing 3-bit error patterns independently. The generation of a single error pattern per cycle will increase the system's latency. Therefore error patterns are generated in parallel. The primary EG checks 2 errors, and the secondary EG checks 16 error patterns in a single cycle to reduce the latency by a factor of 2 and 16, respectively. Thus, the worst-case latency of 1-, 2-, and 3-bit error pattern generation is given by $\binom{128}{1}/2$, $\binom{128}{2}/2$, and $\binom{128}{3}/16$, respectively. Fig. 6 shows that at a bit flip probability (p) of 10^{-3}, the average latency of the system reduces by 31% when the secondary EG is parallelized to generate 16 errors. The average latency reduces by an additional 17% when the 16x parallelization of the secondary EG is combined with



Universal Noise-centric Decoder Signals					
Y	Demodulated channel output	CS	Chip select signal	Tag	Two bit tag for re-ordering
H	Parity-check matrix for syndrome calculation	CG	Clock Gating	00	Tag showing no error in the codeword
E	Candidate error vector (Guessed noise sequence)	EN	Enable signal to the Error Generator	01	Tag showing 1 or 2 bit error
CI	Tag for code-interleaved architecture	H.E	Product of H-matrix and Error vector E	10	Tag showing 3 bit error
H.Y	Product of H-matrix and Y	EG	Error Generator	11	Tag showing 4 or more bit error

Fig. 4. GRAND decoder hardware system architecture and flow diagram [16].

Performance	Single-port SRAM	Dual-port SRAM	Register-File
Total Power (μW)	51.6	86.8	884.4
Leakage Power (μW)	1.75	2.25	3.45
Latency (cyc)	128	64	1
Energy (pJ/b)	0.77	0.65	0.10
Area (μm^2)	487	777	36,805

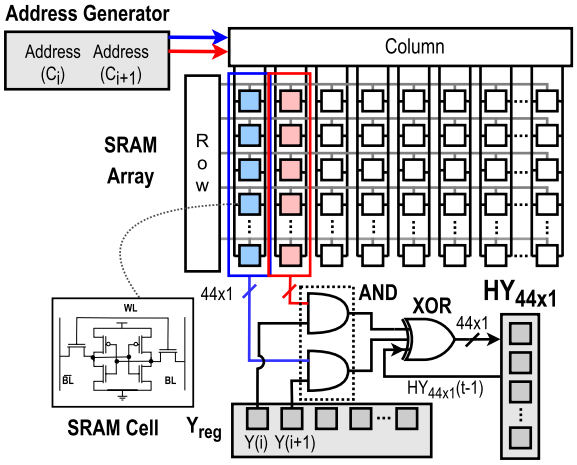


Fig. 5. Synthesized performance comparison of single-port SRAM, dual–port SRAM, and register-file-based syndrome multiplier (top) and syndrome multiplier architecture implemented using a dual-port SRAM (bottom). The parameter i is given as $2x+1$ where x starts as 0 and increments each cycle up to 63. Given that t is the current timestamp, $H.Y_{44\times1}(t-1)$ is the value at a previous timestamp.

2x parallelization in the primary EG. Higher parallelization reduces latency and increases throughput but comes at the cost of greater area and power consumption. The choice of parallelization factor in primary and secondary EGs is determined by the application requirements. The GRAND chip in [16] is designed to operate with low power consumption (less than 5 mW) and a small area (less than 1 mm^2) to meet

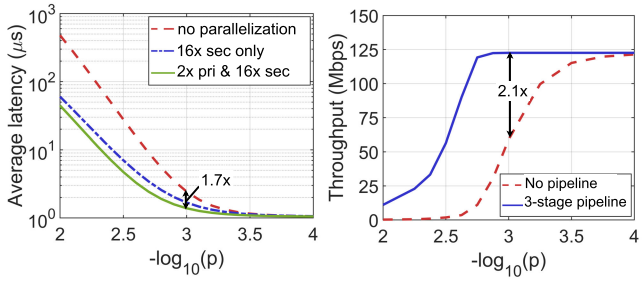


Fig. 6. Simulated average latency of the system when the primary and secondary EG generates 2x and 16x errors in parallel (left); Throughput of a 3-stage pipelined architecture compared to no pipelining at different channel conditions (right). At $p = 10^{-3}$, the average latency decreases by 1.7x, and the average throughput increases by 2.1x when 3-stage pipelining is utilized compared to a single-stage system.

the stringent requirements of IoT devices while delivering the necessary throughput and latency.

C. Pipelining Advantage

Having a three-stage pipelined architecture increases the throughput of the system by a factor of two at a bit flip probability of 10^{-3} compared to a single-stage architecture. The gain in throughput due to a 3-stage pipelined architecture at different channel noise conditions is shown in Fig. 6.

There will be no pipeline stalls on average as long as the average arrival time of codewords to the primary FIFO and secondary FIFO is greater than the average processing time of the primary EG and secondary EG. The graphs in Fig. 7(a) and (b) show the average processing rate of the primary EG for parallelization factors of 1, 2, and 4, and of the secondary EG for parallelization factors of 8, 16, and 32 for varying bit-flip probability p. The syndrome completes the membership check with a fixed latency of 71 cycles. The fastest arrival of codewords to the primary will be 10^6 per second. At higher SNR, the arrival rate to the primary will

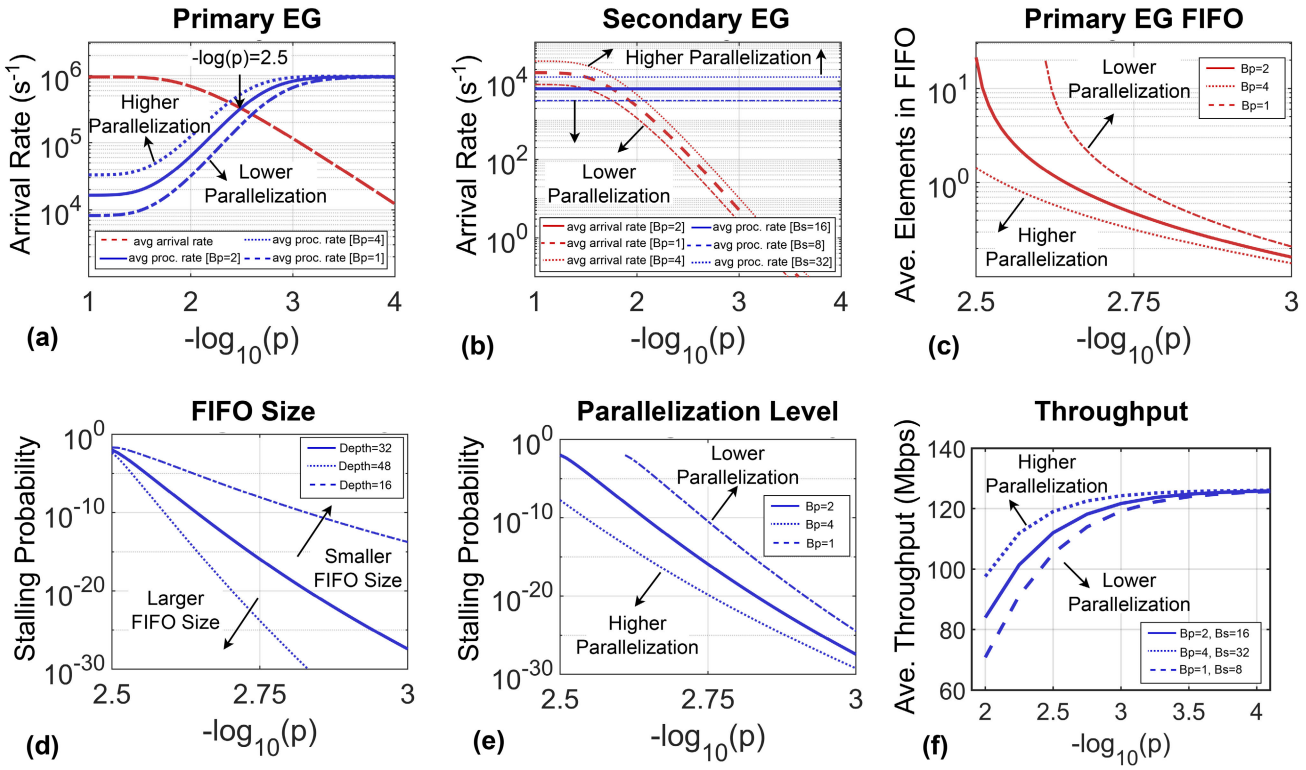


Fig. 7. (a) Average arrival rate and average processing rate of a codeword for primary EG with parallelization factors B_p of 1, 2, and 4; (b) average arrival rate and average processing rate of a codeword for secondary EG with parallelization factors B_s of 8, 16, and 32. The graph in (c) shows the average number of elements in the primary EG FIFO for a given p and B_p. The stalling probability of primary EG FIFO for a FIFO depth of 16, 32, and 48 is shown in (d) and the probability that a stall will occur when the maximum FIFO size is 32 with parallelization factors B_p of 1, 2, and 4 in primary EG is shown in (e). The average throughput obtained for different parallelization factors (B_p, B_s) in primary and secondary EGs is shown in (f) for a given p.

decrease as most of the codewords will pass the membership check by the syndrome. The average processing rate of the primary EG depends on the proportion of the codewords with a 1-bit error, 2-bit error, and greater than 2-bit error. Those with a greater than 2-bit error will be passed to the secondary EG. Each block will operate with maximum throughput when the average processing rate is greater than the average arrival rate. It can be seen that the primary EG FIFO determines the highest p where the chip can operate with negligible probability of a stall occurring. The plot in Fig. 7(c) shows the average number of elements in the FIFO for the primary EG, while Fig. 7(d) and (e) show the probability of the FIFO being full for varying FIFO sizes and parallelization factors. At $p = 10^{-2.66}$ the probability of the FIFO being full is less than 10^{-10} for a FIFO size of 32 with a parallelization factor of two. In the worst-case scenario, when the FIFO is full, the system waits until the primary or the secondary EG has finished decoding the current codeword before loading the next sequence onto the chip, which reduces the average throughput. The impact of stalling on the chip's throughput is shown in Fig. 7(f). The parallelization factors in the EGs are chosen to ensure that the chip operates at maximum throughput for the desired bit-flip probability. All three stages of the system operate in parallel and are clock-gated to save power. A hardware block consumes power only when it is actively processing a codeword. Therefore the average energy of the system depends on the average active time of each block. For a given p, the active time of the primary and

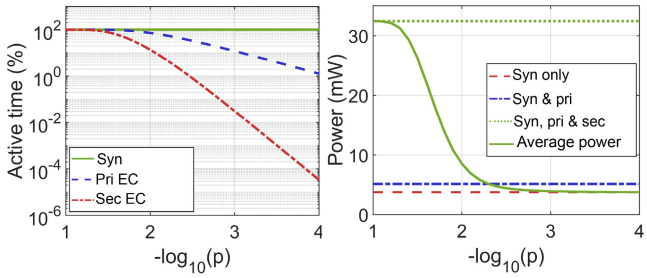


Fig. 8. Simulated active time of each block for different channel noise conditions (left); Measured average power consumption of the clock-gated system compared to power consumption of each block, i.e., syndrome, primary and secondary (right). The average power consumption of the decoder dynamically improves and approaches the syndrome power consumption when the channel noise conditions improve.

secondary EG depends on the probability of having 1- and 2-, and 3-bit errors, respectively, according to the channel noise conditions. The syndrome block is always active since all the codewords first go through the membership check. Fig. 8 shows the average active time of each block for different channel noise conditions. As the channel noise decreases, the system's average power consumption will approach the power consumption of the syndrome block only. Thus, the system energy consumption dynamically adapts when the channel improves, without requiring any feedback.

D. Sparse Matrix Multiplier

The matrix multiplier in the syndrome block takes 64 cycles to compute $H \cdot Y^n$. Using the same multiplier for computing

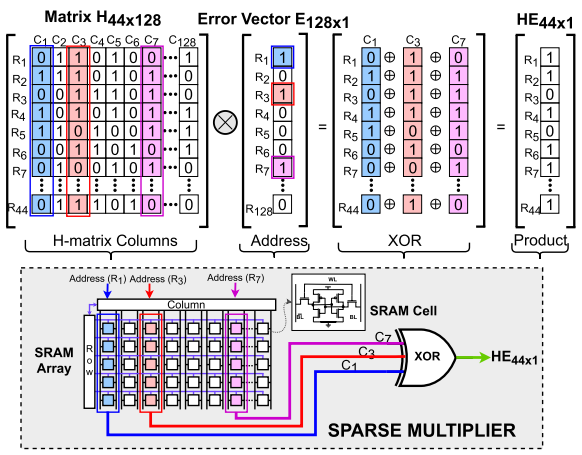


Fig. 9. Low-latency matrix-vector multiplication enabled by the sparsity of error vector [16].

TABLE I
COMPARISON OF PERFORMANCE WHEN THE SYNDROME MULTIPLIER IS USED FOR ERROR VECTOR MULTIPLICATION VERSUS THE PROPOSED SPARSE MULTIPLIER LEVERAGING THE SPARSITY OF THE ERROR VECTORS

Performance	Syndrome multiplier	Sparse multiplier
Power (nW)	74.7	45.2
Latency (cycles)	64	1
Energy (pJ/b)	0.55	0.005
Area (μm^2)	777	322

the $H \cdot E^n$ product will allow checking only one error every 64 cycles. To reduce the latency of calculating the product $H \cdot E^n$ to a single cycle, we leverage the sparsity of noise. The error vector E^n will have 3 active high bits at most since the abandonment threshold is set to 3. These high bits are equally likely to be anywhere in the 128-bit error vector. We take the index of high bits as the column address for the parity-check matrix H stored in the SRAM. These columns of H are XORed together to give the product in a single cycle, as shown in Fig. 9. For checking 1- or 2-bit error patterns in primary EG, one dual-port SRAM returns the respective columns of H depending on the indices of the high bits in E^n. For checking a 3-bit error pattern in secondary EG, two dual-port SRAMs are used to get 3 columns of H in a single cycle where one port of the second SRAM is disabled. A performance comparison of the syndrome multiplier and sparse matrix multiplier is shown in Table I. The sparse multiplier consumes 1.65x lower power, with a 64x lower latency, providing 106x higher energy efficiency.

E. Error Generator

The error generator in primary and secondary EG produces the 1-, 2-, and 3-bit error patterns to be multiplied with H by the sparse multiplier. Each error generator consists of error logic, pattern generator, and error shifter. Distance logic generates a distance pair (D1 and D2) that defines a specific error pattern that is translated into indices of active high bits by a pattern generator. The error shifter uses these indices to shift the pattern to the left to generate all possible errors for that given pattern. The error generator is shown in Fig. 10.

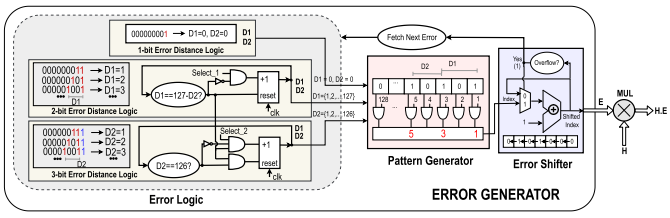


Fig. 10. Key blocks of the Error Generator: Error Logic for distance pair generation, Pattern Generator, and Error Shifter [16].

1) Error Logic: The 1-, 2-, and 3-bit error patterns are defined by a distance pair (D1 and D2). D1 indicates the distance between the first and second active high bit in an error sequence starting from the least significant bit (LSB). D2 indicates the distance between the second and third active high bits. For example, an error vector of 128-bits, $E^{128} = (00\ldots0100101)$, has a distance pair of D1 = 2 and D2 = 3. For a 1-bit error pattern, both D1 and D2 are zero. A 2-bit error pattern is defined by a non-zero D1 where D2 = 0. A 3-bit error pattern needs both D1 and D2 to be determined. The distance logic is implemented as counters. For a 3-bit error with a code length of n = 128, the maximum ranges for D1 and D2 are 1 to $127 - D2$ and 1 to 126, respectively.

2) Pattern Generator: The pattern generator uses the distance pair from distance logic to generate the indices of the active high bits in the error vector. The least significant bit of the error vector is always active high, so the first index is always 1. The second and third high-bit indices are given as $D1$ and $D1 + D2$, respectively. This set of indices is passed to the error shifter.

3) Error Shifter: The error shifter uses the indices provided by the pattern generator to produce all the error combinations associated with one given pattern by shifting to the left. It is implemented as a counter that adds one to the set of indices, essentially performing the shift left operation until an overflow occurs. Overflow is defined as either of the index values of D1 or D2 exceeding $n - 1$. The sparse multiplier uses each set of indices produced by the error shifter to select the columns of the H matrix to be XORed together to produce $H \cdot E^n$. When an overflow occurs, the error shifter requests a new distance pair as all the patterns for the previous pair have been exhausted. The error shifter in the primary EG generates two errors in a single cycle by using 1-bit and 2-bit logical shift left (LSL) operations. The error shifter in secondary uses a combination of 1-bit and 4-bit LSL operations to generate 16 errors in parallel, as shown in Fig. 11. The secondary error shifter is split into 4 branches of 4-bit LSL to reduce the critical path delay. By using the distance pair values D1 and D2 in the error logic, the chip can be configured to produce error patterns for a Hamming Weight of one, two, or three. This enables implementing the distance bound on IGRAND where only a certain number of bits are flipped in the first iteration and the distance bound increases only if necessary for further iterations.

F. Code-Interleaving for Asymmetric Product Codes

GRAND, being a universal decoder, allows the decoding of any binary linear codebook. To use a new codebook, we need a new H-matrix corresponding to the given codebook

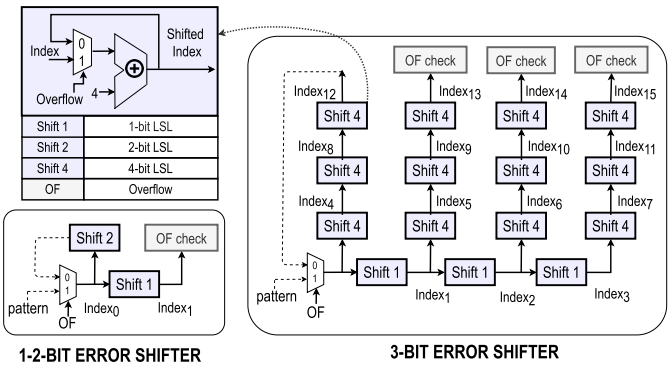


Fig. 11. Error shifter implementation of Primary and Secondary EG [16].

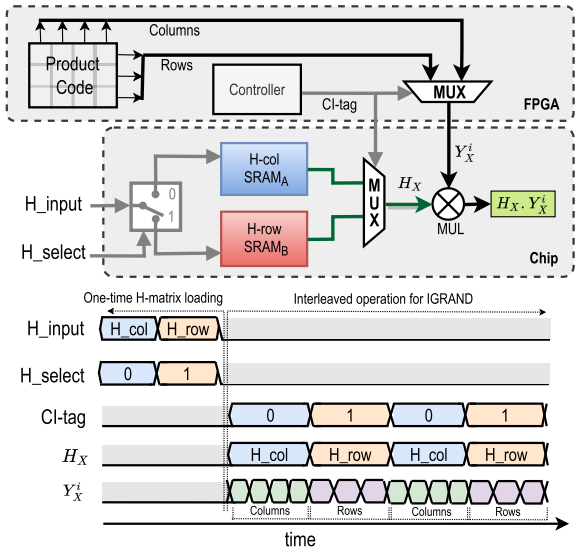


Fig. 12. The Code-Interleaved (CI) architecture block diagram when used for product codes. The H-matrices of the columns and rows are first loaded into the SRAMs A and B. When the columns and rows are iteratively decoded, a 1-bit CI-tag indicates the corresponding SRAM to be used. This allows seamless decoding of rows and columns without any downtime for changing the H-matrix.

for the syndrome, primary EG, and secondary EG blocks. We implemented a Code-Interleaved (CI) architecture to allow multiple codebook support simultaneously without waiting to load a new H-matrix. In the context of product codes, this enables decoding asymmetric product codes where the rows and columns can be encoded by using two distinct codebooks with different rates and code lengths. This will further increase the coding space and hence the system's flexibility in the choice of codebook, code length, and code rate. Each CI hardware chain has a pair of sets of SRAMs. At the beginning of the decoding, we can load two H-matrix belonging to those codebooks. Once the decoding starts, either of the set of SRAMs can be used to decode codewords belonging to the different codebooks. A single bit CI-tag is used to indicate which set of SRAMs has the corresponding H-matrix for the correct decoding. The CI tag is set to 0 for the column decoding and it is changed to 1 for the row decoding to specify the respective H-matrices. This process allows seamless decoding of the incoming codewords belonging to different codebooks without any downtime for loading the new H-matrix. A simplified block diagram is shown in Fig. 12 which shows how the product code can take advantage of this architecture to decode asymmetric product codes without any wait time to change the H-matrix.

IV. Experimental Results

The GRAND chip is fabricated using 40 nm low-power CMOS technology. The chip occupies an active core area of 0.83 mm^2. A die photo of the GRAND chip is shown in Fig. 13(a).

A. Measurement Setup

The measurement setup is shown in Fig. 13(b) and (c). The chip is wire bonded to a QFN package soldered to a test printed circuit board (PCB). The PCB is connected to an Opal Kelly FPGA (XEM-7360) through a breakout board. The FPGA is connected to a host PC through a serial USB port. The PC is only used to initialize and send the generator matrix, and the parity check matrix once before the start of measurements. The FPGA generates the codewords for the component codebook for testing using the generator matrix. An AWGN channel emulation is implemented on the FPGA using linear-feedback shift registers (LFSR) that generate the noise according to the given bit flip probability. An outer wrapper is implemented on the FPGA that constructs a product code matrix using the component codes and adds the generated noise effect to that matrix. The rows and columns of this matrix are the codewords with noise that will be the input of the decoder chip. The product code is then decoded iteratively as described in section II-D. First, all the columns of the product code are sent to the GRAND chip for decoding. IGRAND passes the distance bound as a parameter to the GRAND chip which allows GRAND to abandon its guessing process. A vector of status flags keeps track of columns indicating which columns need to be decoded, which are successfully decoded, and which failed the decoding for the given distance bound. Then the rows are decoded in the same way, and the next iteration begins. This process is repeated until the product code is fully decoded, or further error correction is not possible with the bounding distance set to a maximum of 3 bits. A new product code and noise effect are then generated, and the process is repeated. The FPGA records the measured performance metrics for each product code and outputs the final values to the host PC. To obtain an estimate of the power consumption and area of the IGRAND algorithm implemented on FPGA, we synthesized the IGRAND algorithm including the noise generator in 40 nm CMOS. Fig. 14(a) and (b) show the power and area breakdown of each of the blocks for the full experimental setup for decoding product codes. The GRAND chip area and power are based on the chip measurements, while the IGRAND wrapper area and power are estimated through synthesis. Fig. 14(c) shows the synthesis performance of the IGRAND algorithm in 40 nm CMOS excluding the GRAND chip. The GRAND chip measured results as a component code decoder with the FPGA-supported IGRAND experimental setup are demonstrated separately in section IV-B and IV-C. All the energy and power

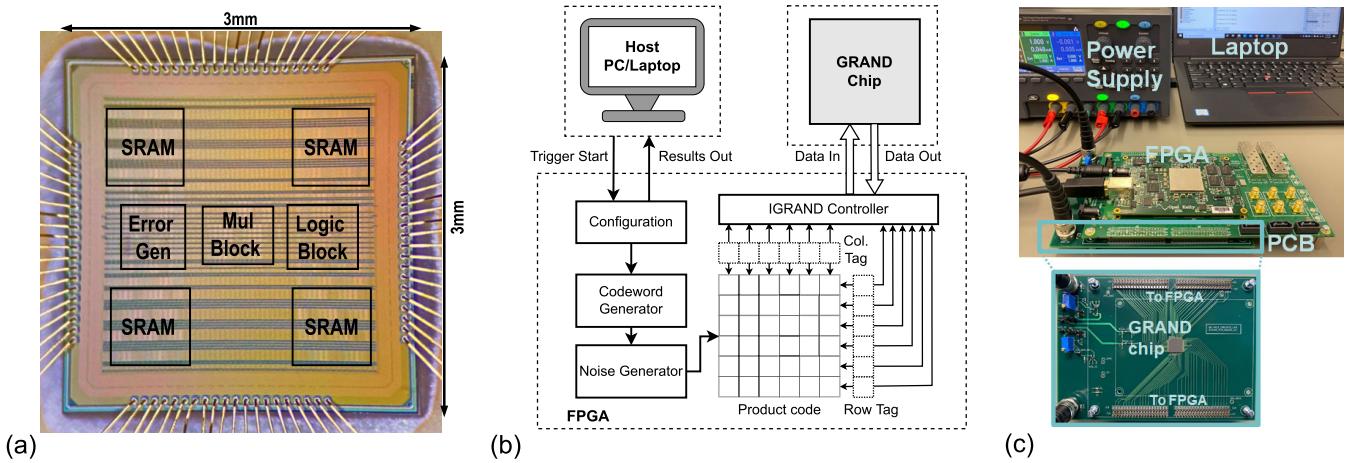
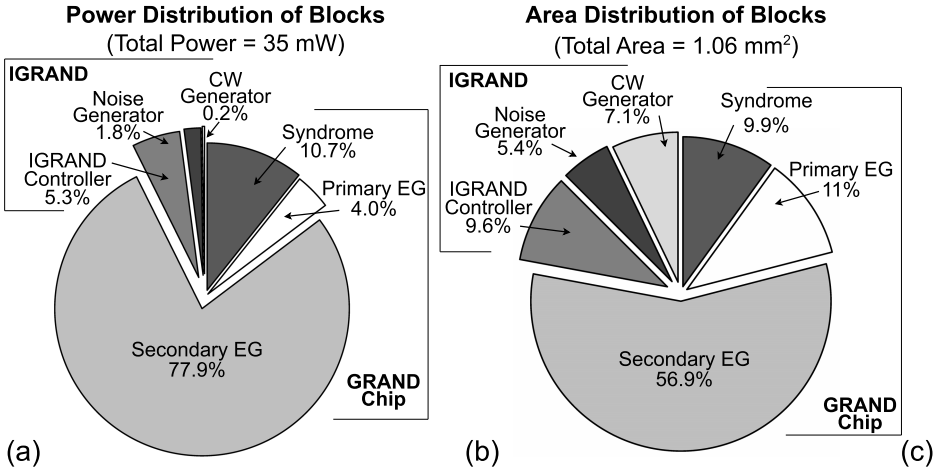


Fig. 13. (a) Die micrograph of the universal GRAND decoder implemented in 40 nm low-power CMOS technology [16] (b) Block diagram of the IGRAND setup (c) Measurement setup [16].



IGRAND Implementation	
Technology (nm)	40
Implementation	Synthesis
Voltage (V)	1.10
Code Length (n)	128x128
Sequential Logic Area (mm^2)	0.103 (44%)
Combinational Logic Area (mm^2)	0.132 (56%)
Total Area (mm^2)	0.235
Frequency (MHz)	70
Leakage power (mW)	0.02
Dynamic power (mW)	2.57

(c)

Fig. 14. (a) Power and (b) area breakdown for each of the blocks used in product decoder implementation. The GRAND chip power and area are based on the chip measurements, while the IGRAND power and area are estimated through synthesis; (c) area and power consumption of the IGRAND algorithm implemented in 40 nm CMOS through synthesis excluding the GRAND chip.

measurement of the chip includes the total leakage power of 0.18 mW.

B. Measured Decoding Performance

Fig. 15(a) shows the decoding performance of the GRAND chip for component codes up to 128-bit code length [16]. The BER performance of the GRAND chip is measured for Random Linear Code (RLC), CRCs, and systematized CA-Polar with varying code lengths and code rates. Fig. 15(b) depicts the measured decoding accuracy of IGRAND experimental setup to decode product codes constructed from CRCs [6], BCH codes [31], extended BCH (eBCH) codes [32], CA-Polar codes and RLCs. The chip can decode all of these codes without adjustment to the hardware; it is required only that the chip be loaded with the corresponding parity-check matrix of the target component code. We note that CRCs and RLCs do not have specialized decoders demonstrated before, and it is the universality of the GRAND chip that enables their decoding for the first time. Of particular note is that the CRC-based product codes achieve equal or better error correction performance compared to the BCH-based product codes, despite CRCs typically being used for error detection.

C. Measured Energy Consumption and Latency Performance

1) Component Code Decoding: Fig. 16(a) and (b) show the measured energy consumption and latency results of the GRAND chip when the component codes BCH(127, 106) and CRC(128, 105) are used to perform the decoding [16]. The GRAND chip provides a base latency of 1 μs with an energy per bit of 30.6 pJ and a throughput of 122.6 Mbps at a bit flip probability of 10^{-5} at 68 MHz operating frequency. At these operating conditions, the chip consumes 3.75 mW of power from a 1.1 V supply. The comparison of the GRAND chip with the state-of-the-art for decoding component codes using the GRAND algorithm is shown in Fig. 18(a). The GRAND chip [16] performance is compared with other hard-detection GRAND decoders [33], [34], [35]. The GRAND chip consumes 5.65× and 13.9× lower power but 5.3× and 3.33× higher energy per bit than the more recently proposed architectures [33], [34]. The lower energy consumption of [33] is

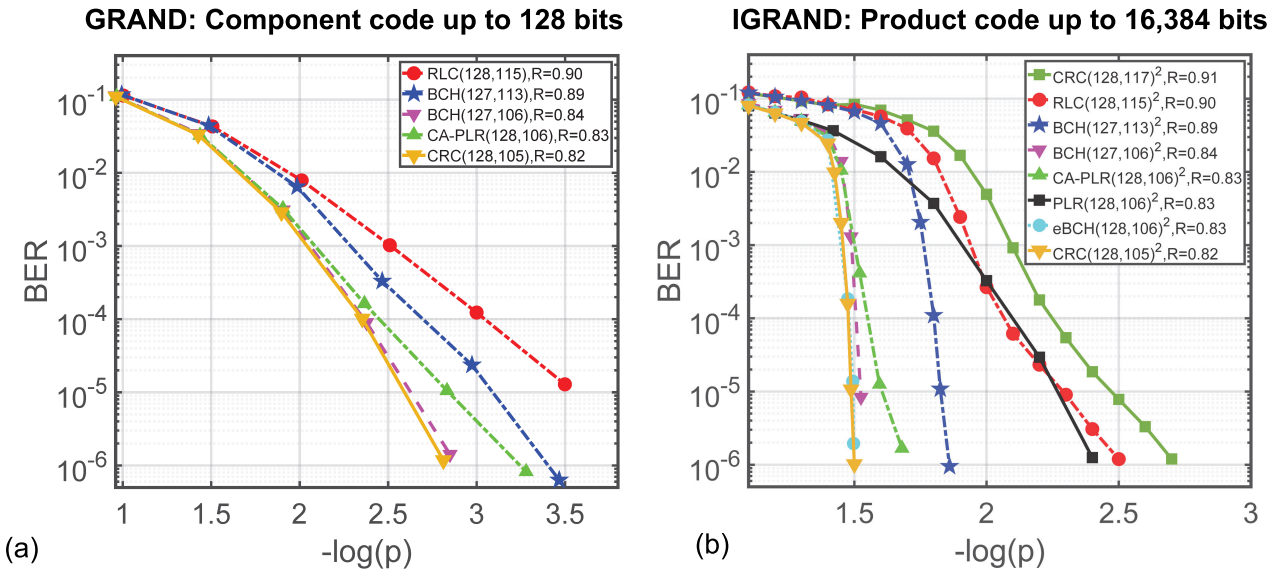


Fig. 15. (a) Measured bit error rates (BER) for component code decoding of RLC(128, 115), BCH(127, 113), BCH(127, 106), systematized CA-Polar(128,106), and CRC(128, 105) using the GRAND chip [16]; (b) measured BER of the following product codes when they are decoded by the IGRAND: CRC(128, 117)^2, RLC(128, 115)^2, BCH(127, 113)^2, BCH(127, 106)^2, CA-Polar(128, 106)^2, Polar(128, 106)^2, eBCH(128, 106)^2, and CRC(128, 105)^2. A single RLC was generated to represent the RLC(128, 115) class of codes, such that the non-systematic part of its generator matrix had no all-zero columns or rows. The CA-Polar code uses an 11-bit CRC, as in the 5G uplink scenario.

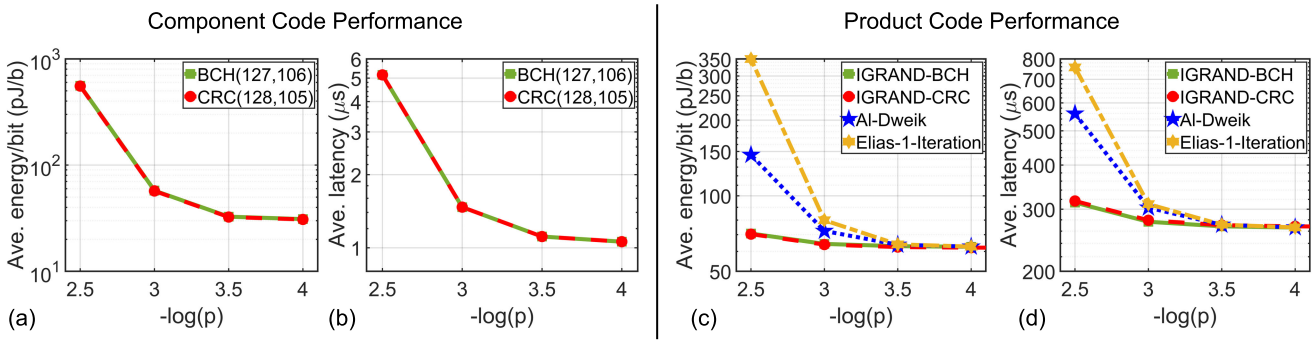


Fig. 16. The graphs in (a) and (b) show the measured average energy consumption per bit and measured average latency for the GRAND chip to decode the component codes BCH(127, 106) and CRC(128, 105). The graphs in (c) and (d) show the measured energy consumption per bit and measured average latency for the iterative decoding of BCH(127, 106)^2 and CRC(128, 105)^2 product codes using IGRAND. The simulated latency and energy of the Al-Dweik/Sharif and Elias algorithms are shown in (c) and (d) for comparison with IGRAND when applied to the BCH code. Here, the performance of IGRAND is measured for both CRC and BCH, whereas the Al-Dweik/Sharif and Elias algorithms are simulated only for BCH.

attributed to the lower supply voltage of 0.75 V, and the energy consumption of [34] is estimated based on synthesis using an ideal clock frequency of 500 MHz compared to the fabricated designs [16], [33] which include parasitics, performance degradation due to routing, and clock non-idealities (i.e., clock skew and jitter), which lower maximum operating frequency that degrades the performance significantly. Moreover, the work in [33] targets latency-critical applications that require constant latency and throughput, with trade-offs in area and power consumption. The GRAND chip in [16] is designed to meet the power and area constraints of IoT devices while meeting the latency and throughput requirements. This design can be modified to achieve higher throughput by increasing the parallelization of noise guessing in the primary and secondary EGs to meet application demands.

2) Product Code Decoding: Fig. 16 (c) and (d) provide the measured latency and energy results, respectively, for the chip when it is used with IGRAND to decode BCH(127, 106, 7)^2 and CRC(128, 105, 7)^2 product codes with code lengths of 16,129 and 16,384 bits and code rates of 0.70 and 0.67, respectively. IGRAND decoding consumes a base energy per bit of 61.2 pJ/b and a base latency of 18k cycles, regardless of the code type. Furthermore, using the column-only decoding technique discussed in section IV-D, the latency can be further reduced as shown in Fig. 17. To compare the IGRAND results with Al-Dweik/Sharif and Elias algorithms, we performed simulations for these algorithms using GRAND as the component code decoder and estimated the average latency and energy consumption. The IGRAND performs with better energy efficiency and lower latency, especially in a noisy channel. As the channel noise conditions improve, the energy consumption and latency approach the best-case scenario for the syndrome check only. The table in Fig. 18(b) shows the performance of product code decoding

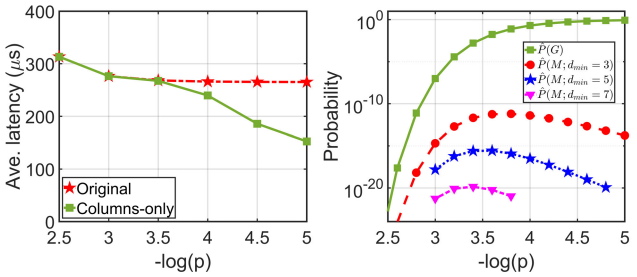


Fig. 17. IGRAND decoding latency of a BCH$(127, 106, 7)^2$ product code when a columns-only decoding strategy is applied. For a bit-flip probability p and a component code length of $n = 127$, the second plot shows the approximate probability $\hat{P}(M)$ of event M, that columns-only decoding will cause a performance compromise. Also included is the probability of G, the event that the columns will genuinely be error-free and that energy costs will be saved.

using the IGRAND algorithm. The rows and columns of the product codes are iteratively decoded using the GRAND chip as the component code decoder and the GRAND chip performance is measured for all the iterations of the product code. At a bit flip probability of $10^{-2.5}$, the IGRAND consumes $2\times$ lower energy per bit and $1.8\times$ lower latency than Al-Dweik/Sharif. At the same conditions, IGRAND consumes $5\times$ lower energy per bit and demonstrates $2.4\times$ lower latency than Elias. When product codes are decoded using a single GRAND chip as the component decoder for the iterative decoding, the throughput obtained is 52 Mbps at p of $10^{-2.5}$. Utilizing the property of the product code that its components are independent of each other and can be decoded in parallel, we can increase the throughput by employing multiple GRAND chips to decode the rows and columns of product code simultaneously. The number of chips operating in parallel can be chosen according to the throughput requirements of the application and the power budget.

D. Columns-Only Decoding

In columns-only decoding, row decoding is skipped whenever the decoding of the product code's columns does not identify any errors, resulting in the reduction of energy usage and latency by 50%. This method utilizes the syndrome check of columns to dynamically determine whether to mark the decoding as complete or proceed to row decoding. If all the columns pass the initial codebook membership check, the decoding is completed which provides gains in energy efficiency and reduction in latency without degrading the decoding performance at higher SNR conditions. Fig. 17 compares decoding latency with and without columns-only decoding. As channel conditions improve, the latency with columns-only decoding converges to ∼9k cycles, which is half of the latency that is achieved without enabling the columns-only decoding.

To study how columns-only decoding can impact the decoding performance, we consider two cases that can occur for any given p. In the first case, no errors occur in both rows and columns and decoding is halted after column decoding, consuming only half the energy cost of both row and column decoding. Let G denote this event. In a BSC with bit-flip probability p and given a product code with component codes of length n, we represent the probability of the event G as $P(G) = (1-p)^{n^2}$.

In the second case, the columns are decoded without detection of errors and hence the row decoding is skipped but unresolved errors still exist in the rows. Let M denote this event. Consider a product code with randomly constructed $[n, k]$ component codes, each with a minimum Hamming distance of $d_{\min}$. Let $N^{n,i}$ taking values in $\{0,1\}^n$ be an error pattern affecting the i-th column and let $B_i = \sum_{j=1}^{n} N_j^{n,i}$. Then,

$$
\begin{aligned}
&P(M)\\
&= \prod_{i=1}^{n} P(N^{n,i} \in \mathcal{C}) - \prod_{i=1}^{n} P(B_i = 0)\\
&= \prod_{i=1}^{n} \Big(P(B_i = 0) + P(B_i \geq d_{\min}, N^{n,i} \in \mathcal{C}) \Big) - (1-p)^{n^2}\\
&\approx \prod_{i=1}^{n} \left((1-p)^n + \left(1 - \sum_{j=0}^{d_{\min}-1} \binom{n}{j} p^j (1-p)^{n-j} \right) \frac{2^k}{2^n} \right)\\
&\quad - (1-p)^{n^2}.
\end{aligned}
$$

In the above, the first equality comes from the fact that a miscorrection occurs when all columns are codewords but not all of them have zero noise effect. In the second equality, the probability of an individual column being a codeword is broken down into the case where it is errorless and the case where it contains at least $d_{\min}$ errors and is a codeword; if $0 < B_i < d_{\min}$, then by the definition of $d_{\min}$ the column cannot be a codeword. For the final equality, the approximation is employed that, given $B_i \geq d_{\min}$, the probability of hitting a codeword is $2^k/2^n$, which is an accurate approximation for well-distributed codebooks [8].

Fig. 17 demonstrates that the estimated probability of a mistake is minimal, particularly for $d_{\min} \geq 5$. Thus, there is a high probability of saving energy, particularly in low-noise channels. This suggests that column-only decoding is a viable strategy for reducing decoding costs at a negligible ($<10^{-10}$) risk of decoding error especially at a higher SNR.

V. Discussion

Hardware decoders are typically limited to decoding product codes with a small collection of component codes. The GRAND chip is demonstrated to support universal decoding of both component codes and any product code constructed with moderate redundancy component codes, regardless of code type. We have presented its performance when the GRAND chip is used as a component code decoder for the IGRAND algorithm, demonstrating its efficacy through hardware experiments, as well as the performance gain of IGRAND over other iterative decoding algorithms.

We have considered BCH and CRC codebooks for performance evaluation. The CRC-based product codes are shown to have similar decoding accuracy to the BCH-based product codes, despite CRCs not typically being used for error correction. Both CRCs and BCH codes outperform CA-Polar

(a) Component Code Performance

References	This Work (ESSCIRC'21)[16]	ESSERC '24 [33]	TVLSI '23 [34]	SiPS '20 [35]
Technology (nm)	40	65	40	65
Implementation	Fabricated	Fabricated	Synthesized Only	Synthesized Only
Voltage (V)	1.10	0.75	0.99	0.90
Decoding Algorithm	Universal GRAND	Universal GRAND	Universal GRAND	Universal GRAND
Measured Codebooks	BCH, RM, CRC, RLC, Polar, CA-Polar, eBCH	BCH	BCH	CRC
Code Length (n)	127, 128[a]	127	127	128
Code Rate (R)	0.656 - 1.000[b]	0.83	0.88	0.75
Average power (mW)	3.75[c]	21.2	52.1	N.R.
Energy per Decoded Bit (pJ/b)	30.6[c]	5.7	9.18	N.R
Throughput (Mbps)	122.6[c]	3710	5670	9000
Frequency (MHz)	10-68	35	500	500
Average Latency (µs)	1.04[c]	0.012	0.019	0.002
Area (mm^2)	0.83	6.15	0.16	0.25[d]

(b) Product Code Performance using GRAND

Product Code Algorithm	IGRAND		Al-Dweik[e]	Elias[e]
Implementation	Measured		Simulated	Simulated
Component Code used	BCH	CRC	BCH	BCH
Code Length (n)	16,129	16,384	16,129	16,129
Code Rate (R)	0.70	0.67	0.70	0.70
BER	$10^{-2.5}$	$10^{-2.5}$	$10^{-2.5}$	$10^{-2.5}$
Average power (mW)	3.60	3.64	4.19	7.47
Energy per Decoded Bit (pJ/b)	70.7	70.3	145.3	350.6
Throughput (Mbps)	51.5	51.8	28.8	21.3
Frequency (MHz)	68	68	68	68
Average Latency (µs)	313	316	559	756

* N.R. = Not Reported
[a] BCH (n=127) and RLC (n=128) used for the measurements.
[b] Measured energy, latency, and throughput is for R = 0.88.
[c] Measured performance at BER = 10^{-5} and 68MHz for a BCH code.
[d] Reported area estimate from synthesized design.
[e] Simulated performance using GRAND as component code decoder.

Fig. 18. (a) Performance summary and comparison for the GRAND chip used as a component code decoder [16] with the other state-of-the-art hard-detection GRAND decoders [33], [34], [35]. (b) Performance summary and comparison for the product code measurements using the GRAND chip with IGRAND, and simulated Elias and Al-Dweik/Sharif.

codes. In addition to their powerful error correction capability, CRCs are also available at all code lengths and rates, making them more flexible than eBCH and Polar codes, which are available only at lengths of power of two. Evidently, CRCs merit more investigation as error correction codes. The same flexibility holds for RLCs, and product codes with random linear component codes (RLPCs) were proven to be capacity-achieving in [17]. We demonstrate the first-time decoding of RLPCs in hardware.

CRCs are ubiquitously used as error detection codes in standards such as Bluetooth Low Energy (BLE) and IEEE 802.15.4 IoT, but they could now be used as error correction codes. CA-Polar codes are used as error-correcting codes in 5G control channel communications [36]. BCH and eBCH product codes have been employed in optical transport networks [19]. The GRAND chip, coupled with an iterative decoding wrapper, could be used as a universal product code decoder in any of these contexts or simply to decode component codes. While we have demonstrated the scaling of the GRAND chip to decoder product codes up to 16,384 bits in code length using the IGRAND experimental setup, the same architecture can be extended to decode longer cubic tensor codes [37] that are constructed by encoding information bits in 3D resulting in a code with dimensions (n^3, k^3) with a rate R^3. As long as the component code is up to 128 bits in codelength, the GRAND chip can be employed as the component code decoder.

A natural progression from the hard-input iterative decoding of product codes investigated in this work is soft-input soft-output iterative decoding, also known as turbo decoding [24]. With the recent development of a high-throughput soft-detection variant of GRAND called Ordered Reliability Bits Guessing Random Additive Noise Decoding (ORBGRAND) [11], it should now be feasible to demonstrate efficient soft-input soft-output iterative decoding. Many efficient hardware architectures have already been developed for ORBGRAND [38], [39], [40] including its first chip realization [41] that consumes a low energy of sub-0.8 pJ/b, making it suitable for energy-efficient soft-detection iterative decoding. It has been shown that soft-output variant of GRAND (SOGRAND) [42] can provide accurate blockwise and bitwise soft output that can be used iteratively to improve the decoding performance significantly. In a soft-detection channel scenario, the received Log-Likelihood Ratios (LLRs) are updated at each iteration using the soft output from the decoder.

VI. Conclusion

We present the hardware implementation of the IGRAND algorithm by integrating the hard-detection GRAND chip [16], implemented in 40 nm CMOS, as a component code decoder with an FPGA-supported outer wrapper for iteratively decoding product codes of up to 16,384 bits in length. We describe the inherent properties of the GRAND chip architecture that enable efficient product code decoding with low power consumption and minimal area while supporting a reconfigurable system for a wide range of codebooks. We demonstrate the measured decoding performance, energy consumption, and latency for various product codes, including capacity-achieving random linear product codes. When decoding the component codes, the GRAND chip achieves an average energy consumption of 30.6 pJ/b with a latency of 1.04 μs at 68 MHz at a bit flip probability of 10^{-5} while consuming 3.75 mW of power from a 1.1 V supply. At the same operating conditions, the chip consumes an average energy of 61.2 pJ/b with an average latency of 265 μs for iteratively decoding product codes using the IGRAND experimental setup. This implementation provides a power- and area-efficient solution for product code decoding while meeting the necessary throughput and latency requirements of various low-power IoT applications.

Acknowledgment

The opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of these organizations.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[2] E. Berlekamp, R. J. McEliece, and H. C. A. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 3, pp. 384–386, May 1978.

[3] E. Berlekamp, "Nonbinary BCH decoding (Abstr.)," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 2, p. 242, Mar. 1968.

[4] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IEEE Trans. Inf. Theory*, vol. IT-4, no. 4, pp. 38–49, Sep. 1954.

[5] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[6] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *Proc. Int. Conf. Dependable Syst. Netw.*, Jul. 2004, pp. 145–154.

[7] P. Koopman, K. Driscoll, and B. Hall, "Selection of cyclic redundancy code and checksum algorithms to ensure critical data integrity," Federal Aviation Admin., NJ, USA, Tech. Rep. DOT/FAA/TC-14/49, 2015.

[8] K. R. Duffy, J. Li, and M. Médard, "Capacity-achieving guessing random additive noise decoding," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4023–4040, Jul. 2019.

[9] K. R. Duffy, A. Solomon, K. M. Konwar, and M. Médard, "5G NR CA-polar maximum likelihood decoding by GRAND," in *Proc. 54th Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2020, pp. 1–5.

[10] W. An, M. Médard, and K. R. Duffy, "Keep the bursts and ditch the interleavers," *IEEE Trans. Commun.*, vol. 70, no. 6, pp. 3655–3667, Jun. 2022.

[11] K. R. Duffy, W. An, and M. Médard, "Ordered reliability bits guessing random additive noise decoding," *IEEE Trans. Signal Process.*, vol. 70, pp. 4528–4542, 2022.

[12] K. R. Duffy, J. Li, and M. Médard, "Guessing noise, not code-words," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 671–675.

[13] K. R. Duffy, M. Médard, and W. An, "Guessing random additive noise decoding with symbol reliability information (SRGRAND)," *IEEE Trans. Commun.*, vol. 70, no. 1, pp. 3–18, Jan. 2022.

[14] W. An, M. Médard, and K. R. Duffy, "CRC codes as error correction codes," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6.

[15] A. Solomon, K. R. Duffy, and M. Médard, "Soft maximum likelihood decoding using GRAND," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[16] A. Riaz et al., "Multi-code multi-rate universal maximum likelihood decoder using GRAND," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2021, pp. 239–246.

[17] K. Galligan, A. Solomon, A. Riaz, M. Médard, R. T. Yazicigil, and K. R. Duffy, "IGRAND: Decode any product code," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.

[18] P. Elias, "Error-free coding," *IEEE Trans. Inf. Theory*, vol. IT-4, no. 4, pp. 29–37, Sep. 1954.

[19] J. Justesen, K. J. Larsen, and L. A. Pedersen, "Error correcting coding for OTN," *IEEE Commun. Mag.*, vol. 48, no. 9, pp. 70–75, Sep. 2010.

[20] M. Scholten, T. Coe, and J. Dillard, "Continuously-interleaved BCH (CI-BCH) FEC delivers best in class NECG for 40G and 100G metro applications," in *Proc. Nat. Fiber Optic Eng. Conf.*, Mar. 2010, pp. 1–3.

[21] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *J. Lightw. Technol.*, vol. 30, no. 1, pp. 110–117, Jan. 1, 2012.

[22] C. Yang, Y. Emre, and C. Chakrabarti, "Product code schemes for error correction in MLC NAND flash memories," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 12, pp. 2302–2314, Dec. 2012.

[23] T. V. Vo and S. Mita, "A novel error-correcting system based on product codes for future magnetic recording channels," *IEEE Trans. Magn.*, vol. 47, no. 10, pp. 3320–3323, Oct. 2011.

[24] R. M. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

[25] A. J. Al-Dweik and B. S. Sharif, "Non-sequential decoding algorithm for hard iterative turbo product codes," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1545–1549, Jun. 2009.

[26] J. Justesen, "Performance of product codes and related structures with iterated decoding," *IEEE Trans. Commun.*, vol. 59, no. 2, pp. 407–415, Feb. 2011.

[27] C. Häger and H. D. Pfister, "Approaching miscorrection-free performance of product codes with anchor decoding," *IEEE Trans. Commun.*, vol. 66, no. 7, pp. 2797–2808, Jul. 2018.

[28] N. Abramson, "Cascade decoding of cyclic product codes," *IEEE Trans. Commun. Technol.*, vol. CT-16, no. 3, pp. 398–402, Jun. 1968.

[29] M. Belkasmi, M. Lahmer, and F. Ayoub, "Iterative threshold decoding of product codes constructed from majority logic decodable codes," in *Proc. 2nd Int. Conf. Inf. Commun. Technol.*, vol. 2, Apr. 2006, pp. 2376–2381.

[30] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, vol. 1, Jun. 1994, pp. 339–343.

[31] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[32] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Hoboken, NJ, USA: Wiley, 2021.

[33] L. D. Blanc et al., "A GRANDAB decoder with 8.48 Gbps worst-case throughput in 65nm CMOS," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2024, pp. 685–688.

[34] S.-I. Chu, S.-A. Ke, S.-J. Liu, and Y.-W. Lin, "An efficient hard-detection GRAND decoder for systematic linear block codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 11, pp. 1852–1864, Nov. 2023.

[35] S. M. Abbas, T. Tonnellier, F. Ercan, and W. J. Gross, "High-throughput VLSI architecture for GRAND," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2020, pp. 1–6.

[36] *Technical Specification Group Radio Access Network; Study on Scenarios and Requirements for Next Generation Access Technologies; (Release 16)*, document TR 38.913, 3GPP, 2020.

[37] S. Khalifeh, K. R. Duffy, and M. Médard, "Turbo product decoding of cubic tensor codes," in *Proc. 59th Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2025, pp. 1–6.

[38] S. M. Abbas, T. Tonnellier, F. Ercan, M. Jalaleddine, and W. J. Gross, "High-throughput and energy-efficient VLSI architecture for ordered reliability bits GRAND," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 6, pp. 681–693, Jun. 2022.

[39] C. Condo, "A fixed latency ORBGRAND decoder architecture with LUT-aided error-pattern scheduling," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 2203–2211, May 2022.

[40] C. Ji, X. You, C. Zhang, and C. Studer, "Efficient ORBGRAND implementation with parallel noise sequence generation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 2, pp. 435–448, Feb. 2025.

[41] A. Riaz et al., "A Sub-0.8-pJ/bit universal soft-detection decoder using ORBGRAND," *IEEE J. Solid-State Circuits*, early access, Nov. 27, 2024, doi: 10.1109/JSSC.2024.3502240.

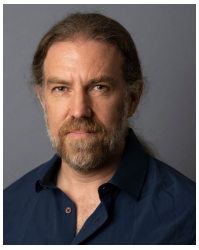
[42] P. Yuan, M. Médard, K. Galligan, and K. R. Duffy, "Soft-output (SO) GRAND and iterative decoding to outperform LDPC codes," *IEEE Trans. Wireless Commun.*, vol. 24, no. 4, pp. 3386–3399, Apr. 2025.

Arslan Riaz (Member, IEEE) received the B.S. degree in electrical engineering from the National University of Sciences and Technology (NUST), Islamabad, in 2018, and the M.S. and Ph.D. degrees from Boston University, MA, USA, in 2024, specializing in Guessing random additive noise decoding hardware implementation. He is a Research Scientist with the Electrical and Computer Engineering Department, Boston University. His research is oriented toward the development of energy-efficient and secure wireless communication systems in collaboration with MIT and Northeastern University. He was a recipient of several awards, including the IEEE COMSNETS'22 and COMSNETS'23 Best Research Demo Award, the CISE Best Presentation Award in 2022 and 2024, and the BU Doctoral Achievement Award in 2024.

Kevin Galligan (Graduate Student Member, IEEE) received the Ph.D. degree from the Hamilton Institute, Maynooth University, Ireland, in 2024. He has published several peer-reviewed papers, including one that received the Best Paper award at CISS 2023. His research has focused on decoding algorithms for channel codes.

Alperen Yasar (Graduate Student Member, IEEE) received the B.Sc. degree in electronics engineering from Sabanci University, Istanbul, Türkiye, in 2021. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Boston University, Boston, MA, USA. Since Fall 2021, he has been a member of the WISE-Circuits, Boston University. His research interests focus on ultra-low-power analog design for biomedical applications and analog/RF solutions for physical-layer security.

Vaibhav Bansal (Student Member, IEEE) received the Bachelor of Engineering degree in mechatronics engineering from the Thapar Institute of Engineering and Technology, Punjab, India, in 2018, and the M.Sc. degree in electrical and computer engineering from Boston University, MA, USA, in 2020. He is currently with Advanced Micro Devices (AMD), as a Senior Silicon Design Engineer, where his work majorly involves the design and implementation of various cache and floating point designs. He is interested in the design and implementation of high-speed ASICs.



Ken R. Duffy (Senior Member, IEEE) received the B.A. degree in mathematics and the Ph.D. degree in probability theory from Trinity College Dublin, in 1996 and 2000, respectively.

He is a Professor of electrical and computer engineering and a Professor of mathematics with Northeastern University, where he is affiliated to the Institute for the Wireless Internet of Things. He works in collaborative multi-disciplinary teams to design, analyze, and realize algorithms using tools from probability, statistics, and machine learning. Algorithms he has developed have been implemented in digital circuits and in DNA. He is a Co-Founder of the Royal Statistical Society's Applied Probability Section in 2011; co-authored a cover article of *Trends in Cell Biology* in 2012; a Co-Winner of the Best Paper Award at the IEEE International Conference on Communications in 2015; and received the Best Paper Award from IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING in 2019, the Best Research Demo Award from COMSNETS in 2022 and 2023, and the Ellersick Best Paper Award from IEEE MILCOM in 2024. He serves as an Associate Editor for IEEE TRANSACTIONS ON INFORMATION THEORY and IEEE TRANSACTIONS ON MOLECULAR, BIOLOGICAL AND MULTI-SCALE COMMUNICATIONS.

Muriel Medard (Fellow, IEEE) received the three bachelor's degrees in EECS, in mathematics, and in humanities, and the M.S. and Sc.D. degrees from MIT, in 1989, 1989, 1991, 1991, and 1995, respectively.

She is the NEC Professor of software science and engineering with the Electrical Engineering and Computer Science (EECS) Department, MIT, where she leads the Network Coding and Reliable Communications Group, Research Laboratory for Electronics; and a Chief Scientist with Steinwurf, which she has co-founded. She has over 80 U.S. and international patents awarded, the vast majority of which have been licensed or acquired. For technology transfer, she has co-founded CodeOn, for which she consults, and Steinwurf, for which she is a Chief Scientist. She has supervised over 40 master's students, over 20 doctoral students, and over 25 postdoctoral fellows.

Dr. Medard is a member of German National Academy of Sciences Leopoldina (elected in 2022), American Academy of Arts and Sciences (elected in 2021), U.S. National Academy of Engineering (elected in 2020); and a fellow of U.S. National Academy of Inventors (elected in 2018). She holds Honorary Doctorates from the Technical University of Munich in 2020, The University of Aalborg in 2022, and Budapest Institute of Technology and Economics (BME) in 2023. She was a Co-Winner of the MIT 2004 Harold E. Edgerton Faculty Achievement Award in 2007. She received the 2017 IEEE Communications Society Edwin Howard Armstrong Achievement Award, the 2016 IEEE Vehicular Technology James Evans Avant Garde Award, the 2022 IEEE Kobayashi Computers and Communications Award, the 2019 Best Paper Award for IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, the 2018 ACM SIGCOMM Test of Time Paper Award, and nine conference paper awards. She received the inaugural MIT Excellence in Postdoctoral Mentoring Award in 2022 and the inaugural MIT EECS Graduate Student Association Mentor Award in 2013, voted by the students. She set up the Women in the Information Theory Society (WithITS) and Information Theory Society Mentoring Program, for which she was recognized with the 2017 IEEE Aaron Wyner Distinguished Service Award. She serves on the Nokia Bell Labs Technical Advisory Board. She serves as the Editor-in-Chief for IEEE TRANSACTIONS ON INFORMATION THEORY. She was elected as the President of the IEEE Information Theory Society in 2012 and served on its board of governors for a dozen years. She has served as an Editor or a Guest Editor for many IEEE publications, including IEEE TRANSACTIONS ON INFORMATION THEORY, IEEE JOURNAL OF LIGHTWAVE TECHNOLOGY, and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY.

Rabia Tugce Yazicigil (Senior Member, IEEE) received the Ph.D. degree from Columbia University in 2016. She is an Assistant Professor with the ECE Department, Boston University, and the Network Faculty, Sabanci University. She was a Post-Doctoral Associate with MIT. Her research interests lie at the interface of integrated circuits, bio-sensing, signal processing, security, and wireless communications to innovate system-level solutions for future energy-constrained applications. She has received several awards, including the NSF CAREER Award in 2024, the Early Career Excellence in Research Award for the Boston University College of Engineering in 2024, the Catalyst Foundation Award in 2021, the Boston University ENG Dean Catalyst Award in 2021, and the Electrical Engineering Collaborative Research Award for the Ph.D. research in 2016. She is an Active Member of the Solid-State Circuits Society (SSCS) Women-in-Circuits Committee and a member of the 2015 MIT EECS Rising Stars Cohort. She was selected as an IEEE SSCS Distinguished Lecturer for the 2024–2026 term and elected to the IEEE SSCS AdCom as the Member-at-Large in 2024. She was selected as a member of the 2024 National Academy of Engineering (NAE) U.S. Frontiers of Engineering (USFOE) Cohort. Additionally, she is the Workshop Co-Chair of IEEE ESSERC 2024; and a Technical Program Committee Member of IEEE ISSCC, RFIC, and DAC. She serves as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR ARTIFICIAL INTELLIGENCE.