

High-Throughput and Energy-Efficient VLSI Architecture for Ordered Reliability Bits GRAND

Syed Mohsin Abbas, Thibaud Tonnellier, Furkan Ercan, *Member, IEEE*
 Marwan Jalaleddine and Warren J. Gross, *Senior Member, IEEE*

Abstract—Ultra-reliable low-latency communication (URLLC), a major 5G New-Radio use case, is the key enabler for applications with strict reliability and latency requirements. These applications necessitate the use of short-length and high-rate channel codes. Guessing Random Additive Noise Decoding (GRAND) is a recently proposed Maximum Likelihood (ML) decoding technique for these short-length and high-rate codes. Rather than decoding the received vector, GRAND tries to infer the noise that corrupted the transmitted codeword during transmission through the communication channel. As a result, GRAND can decode any code, structured or unstructured. GRAND has hard-input as well as soft-input variants. Among these variants, Ordered Reliability Bits GRAND (ORBGRAND) is a soft-input variant that outperforms hard-input GRAND and is suitable for parallel hardware implementation. This work reports the first hardware architecture for ORBGRAND, which achieves an average throughput of up to 42.5 Gbps for a code length of 128 at a target FER of 10^{-7} . Furthermore, the proposed hardware can be used to decode any code as long as the length and rate constraints are met. In comparison to the GRANDAB, a hard-input variant of GRAND, the proposed architecture enhances decoding performance by at least 2 dB. When compared to the state-of-the-art fast dynamic successive cancellation flip decoder (Fast-DSCF) using a 5G polar code (128, 105), the proposed ORBGRAND VLSI implementation has $49\times$ higher average throughput, $32\times$ times more energy efficiency, and $5\times$ more area efficiency while maintaining similar decoding performance.

Index Terms—Area efficiency, Energy efficiency, Error Correcting Code (ECC), Guessing Random Additive Noise Decoding (GRAND), Maximum Likelihood Decoding (MLD), Ordered Reliability Bits GRAND (ORBGRAND), VLSI architecture

I. INTRODUCTION

FOLLOWING Shannon's seminal 1948 paper [3], much work was directed toward developing practical coding schemes that could approach channel capacity (named “the Shannon limit”). Early proposed channel coding techniques mainly focused on designing error correcting codes where the number of correctable errors is selected by design. This approach towards channel coding brought rise to Hamming codes [4] and Bose–Chaudhuri–Hocquenghem (BCH) codes [5], [6]. Recently, researchers have been trying to discover new capacity achieving and capacity-approaching codes. Most notably, Turbo codes [7] and LDPC codes [8], were proposed

S. M. Abbas, T. Tonnellier, M. Jalaleddine and W. J. Gross are with the Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada. F. Ercan is affiliated with the Department of Electrical and Computer Engineering, Boston University, Boston, MA, 02115. (email: syed.abbas@mail.mcgill.ca, thibaud.tonnellier@mail.mcgill.ca, fercan@bu.edu, marwan.jalaleddine@mail.mcgill.ca, warren.gross@mcgill.ca.). A part of this work has been presented in ICASSP 2021 [1].

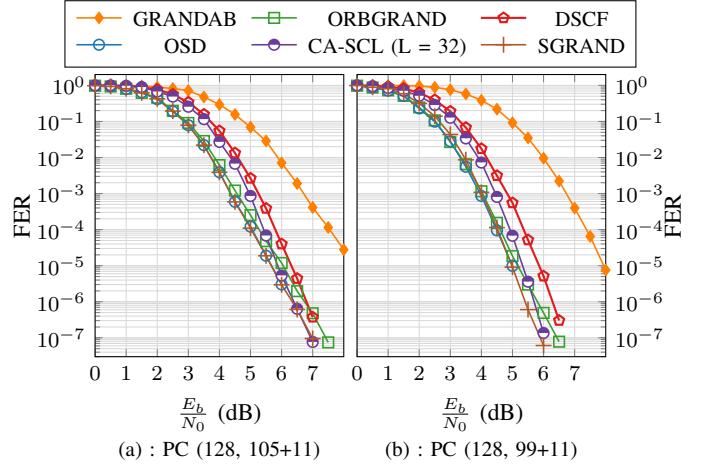


Fig. 1. Comparison of the decoding performance of different GRAND variants with OSD (Order = 2), CA-SCL and DSCF ($\omega = 2$, $T_{\max} = 50$) decoder for 5G-NR Polar Codes.

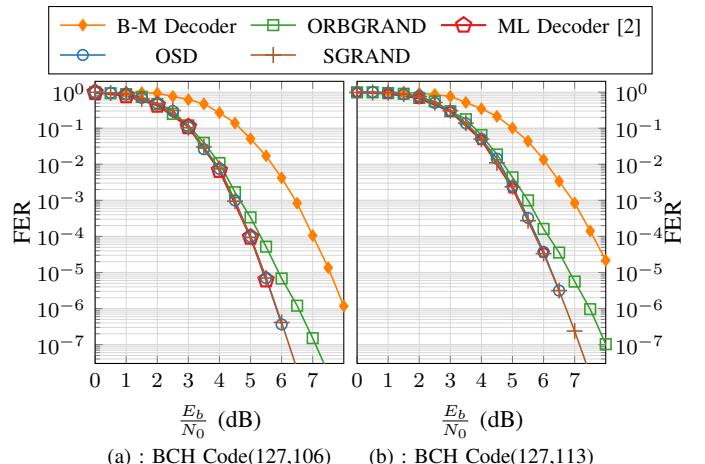


Fig. 2. Comparison of the decoding performance of different GRAND variants with OSD (Order = 2) and ML decoder for BCH Codes.

over time as capacity-approaching codes. On the other hand, Polar codes [9], proposed in 2008, are the first proven class of codes that asymptotically reach the Shannon limit for binary-input symmetric memory-less channels, as well as discrete and continuous memory-less channels [10]. However, designing codes that perform well in the short-to-medium block length regime is challenging [11].

The Guessing Random Additive Noise Decoding (GRAND) [12] is a recently proposed Maximum Likelihood (ML) decod-

ing technique for short-length and high-rate channel codes. Emerging applications such as intelligent transportation systems (ITS) [13], the internet of things (IoT) [14], [15], augmented and virtual reality, machine to machine communication (M2M) [16], and ultra-reliable low-latency communication (URLLC) [17] (5G New-Radio (NR) use case) necessitate short data packets with high reliability (target FER of $10^{-5} \sim 10^{-9}$) and ultra low latency to deal with critical events. GRAND is a maximum likelihood decoder for these channel codes with short lengths and high rates. Furthermore, because GRAND focuses on noise, transmissions with low noise are quickly decoded. GRAND is therefore suitable for applications that require high reliability and ultra-low latency.

In addition to GRAND, other universal decoders for (n, k) linear block codes, where n is the code length and k is the code dimension, include brute-force ML decoding, Ordered Statistic Decoding (OSD) [18] and its variants [19], [20], [21], [22]. Due to the high complexity requirement, the ML decoder is unsuitable for high-rate codes. Whereas the OSD and its variants permute the columns of the generator matrix (G) of the underlying code in a way that is dependent on the received vector of channel observation values (y), and the G matrix is transformed to a systematic form using Gaussian elimination. OSD is unsuitable for parallel hardware implementation [23] due to the complexity of Gaussian elimination ($\mathcal{O}(n^3)$) and the reliance of column permutations of G on the received vector (y). GRAND, on the other hand, provides a low-complexity ML decoding solution for short-length and high-rate codes because it does not require Gaussian elimination or G column permutations. GRAND instead requires simple bit-flipping and syndrome check (codebook membership verification) operations.

GRAND generates test error patterns, which are successively applied to the vector of channel observation values (y) and the resulting vector is evaluated for codebook membership. GRAND can be used with any codebook as long as there is a method for validating a vector's codebook membership. For linear codebooks (C), the codebook membership for a vector can be verified using the parity check matrix H . For other non-structured codebooks, stored in a dictionary, the codebook membership of a vector can be checked with a dictionary lookup. For the rest of the discussion, we restrict ourselves to (n, k) linear block codes.

The order in which these test error patterns are generated is the primary distinction between GRAND variants. GRAND with ABandonment (GRANDAB) [12] is a hard decision input version of GRAND that produces test error patterns in increasing Hamming weight order up to weight AB . Ordered Reliability Bits GRAND (ORBGRAND) [24] and Soft GRAND (SGRAND) [25] are soft-input variants that efficiently utilize soft information (channel observation values), resulting in improved decoding performance over hard-input GRANDAB.

Figure 1 compares the decoding performance of various GRAND variants for decoding 5G NR CRC-aided polar code (128,105+11) and polar code (128,99+11). Furthermore, the decoding performance of state-of-the-art soft-input decoders such as the CRC-Aided Successive Cancellation List (CA-

SCL) decoder [26], [27], Dynamic SC-Flip (DSCF) [28], [29] decoder and OSD [18], [19], [20] is included for reference. The numerical simulation results presented in this work are based on BPSK modulation over an AWGN channel. The ORBGRAND and SGRAND soft-input decoders outperform the hard-input GRANDAB variant in decoding performance, and the SGRAND variant achieves ML decoding performance similar to OSD, as shown in Fig. 1. Figure 2 compares the decoding performance of different GRAND variants with OSD and ML decoding of BCH codes (127, 106) and (127, 113), respectively. The ML decoding results are obtained from [2]. Similar trends in decoding performance can be seen in the BCH codes depicted in Fig. 2, where soft-input variants of GRAND (ORBGRAND and SGRAND) outperform the hard-input traditional Berlekamp-Massey (B-M) [30], [31] decoder and the SGRAND achieves ML performance comparable to OSD.

The SGRAND outperforms the other GRAND variants in terms of decoding performance; however, SGRAND is not suitable for parallel hardware implementation. The generated test error patterns in SGRAND are interdependent, and their query order changes for each received vector of channel observation values (y) [25]. As a result, SGRAND does not lend itself to efficient parallel hardware implementation, and a sequential hardware implementation will result in a high decoding latency, which is unsuitable for applications that require ultra-low latency. The ORBGRAND, on the other hand, generates test error patterns in a predetermined logistic weight order based on integer partitioning. Furthermore, the test error patterns generated are mutually independent and can be generated in parallel. ORBGRAND is thus highly parallelizable and well suited to parallel hardware implementation.

In this paper, we investigate the parameters that affect the decoding performance as well as the computational complexity of the ORBGRAND algorithm, and we propose modifications to the ORBGRAND algorithm to aid hardware implementation and reduce complexity. In addition, we present a high-throughput and energy-efficient ORBGRAND VLSI architecture. The VLSI implementation results show that, considering a code of length 128 and a target FER of 10^{-7} , the proposed hardware architecture can achieve an average information throughput of up to 42.5 Gbps. Furthermore, the proposed hardware can be used to decode any code as long as the length and rate constraints are met. When compared to the state-of-the-art fast dynamic successive cancellation flip decoder (Fast-DSCF) [28], [29] using a 5G NR polar code (128, 105), the proposed ORBGRAND implementation results in $49\times$ more average throughput, $32\times$ more energy efficiency, and $5\times$ more area efficiency.

It should be noted that a part of this work was previously discussed in [1]. This paper builds on the earlier work [1] and extends the proposed ORBGRAND in following ways:

- For various classes of channel codes, including Bose-Chaudhuri-Hocquenghem (BCH) codes, Cyclic Redundancy Check (CRC) codes, Random Linear Codes (RLCs), and Polar codes, the proposed ORBGRAND is evaluated in terms of decoding performance and compared with state-of-the-art decoding approaches.

- The ORBGRAND VLSI architecture [1] can only support a limited set of parameters ($LW \leq 64$ and $P \leq 6$). The proposed ORBGRAND VLSI architecture is expanded to support the additional parameters ($LW \leq 96$ and $P \leq 8$) in this work. Furthermore, detailed VLSI implementation results for area, power, throughput, hardware efficiency, and energy efficiency are presented.
- This paper describes the proposed test error pattern generation scheme for the ORBGRAND hardware and presents a step-by-step procedure for leveraging shift registers to generate error patterns corresponding to logistic weight order based on the integer partitioning procedure.
- This paper provides a comprehensive analysis of worst-case latency and throughput for selecting different parameters for the proposed ORBGRAND hardware. This analysis assists in the selection of optimal ORBGRAND hardware parameters to meet the throughput and latency requirements of a target application.
- For the proposed ORBGRAND VLSI hardware, we propose segmenting the sorter module into multiple partitions. The effect of partition number on the displacement of LLR elements $|y_i|$ ($\forall i \in [1, n]$) from their correct locations and their effect on ORBGRAND decoding performance is thoroughly investigated. Finally, the ORBGRAND with segmented sorter approach is implemented and compared to the baseline ORBGRAND hardware with non-segmented sorter, and the effect of varying the number of sorter-segments on area overhead is investigated.

The remainder of this work is structured as follows: Section II includes preliminary information on GRAND and ORBGRAND. Section III investigates ORBGRAND parameters and proposes modifications for simple hardware implementation as well as complexity reduction of ORBGRAND. The proposed hardware architecture for ORBGRAND is detailed in Section IV. Section V presents implementation results and compares them to the Fast-DSCF and GRANDAB decoders. Finally, in Section VI, concluding remarks are made.

II. PRELIMINARIES

A. Notations

Matrices are denoted by a bold upper-case letter (M), while vectors are denoted with bold lower-case letters (v). The transpose operator is represented by $^\top$. The number of k -combinations from a given set of n elements is noted by $\binom{n}{k}$. $\mathbb{1}_n$ is the indicator vector where all locations except the n^{th} are 0 and the n^{th} is 1. Similarly, $\mathbb{1}_v$ is the indicator vector in which all locations v_i ($\forall i \in [1, n]$) are 1. All the indices start at 1. For this work, all operations are restricted to the Galois field with 2 elements, noted \mathbb{F}_2 . The symbols \therefore and \because denote *therefore* and *because* respectively. $a \circ b$ denotes permuting elements in a according to the permutation order in b .

B. GRAND decoding of linear block codes

A linear block code is a linear mapping $g : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$, where $k < n$. In this mapping, a vector u of size k maps to a vector c of size n and the ratio $R \triangleq \frac{k}{n}$ is called the code-rate. For

every linear block code, there exists a $k \times n$ matrix G called generator matrix and a $(n - k) \times n$ matrix H called parity check matrix. The set of the 2^k vectors c is called a code \mathcal{C} , whose elements c are called codewords and each codeword verifies the following property:

$$\forall c \in \mathcal{C}, H \cdot c^\top = 0. \quad (1)$$

Consider the case where c was sent via a noisy channel and r was received at the channel's output. Because of the channel noise, r and c might differ. As a result, the relationship between r and c may be deduced as follows: $r = c \oplus e$, where e represents the channel-induced noise.

GRAND sequentially generates the test error patterns (e), and applies them to r and checks for codebook membership of r by verifying that

$$H \cdot (r \oplus e)^\top \quad (2)$$

is equal to zero. If so, e is the guessed noise and $\hat{c} \triangleq r \oplus e$ is the estimated codeword. GRAND's focus is on noise, thus it can be used with any codebook as long as there is a method for validating a vector's codebook membership. For linear codebooks, the codebook membership for a vector can be verified using H .

C. ORBGRAND decoding

Algorithm 1 summarizes the steps of the ORBGRAND. The inputs to the algorithm are the vector of channel observation values (log-likelihood ratios (LLRs)) y of size n , a $(n - k) \times n$ parity check matrix of the code H , an $n \times k$ matrix G^{-1} such that $G \cdot G^{-1}$ is the $k \times k$ identity matrix, with G a generator matrix of the code, and the maximum logistic weight considered LW_{\max} ($LW_{\max} \leq \frac{n(n+1)}{2}$). The logistic weight (LW) corresponds to the sum of the indices of non zero elements in the test error patterns (e)[24]. For example, $e = [0, 1, 0, 0, 1, 0]$ has a Hamming weight of 2, whereas the logistic weight is $2 + 5 = 7$.

ORBGRAND begins by sorting y in ascending order of absolute value of LLRs ($|y|$), and the corresponding indices are recorded in a permutation vector denoted by ind (line 1). Following that, all integer partitions ($\lambda = (\lambda_1, \lambda_2, \dots, \lambda_P) \vdash i$ where $P \in [1, P_{\max}]$ and $i \in [0, LW_{\max}]$; explained in section III) are generated for each logistic weight (line 3). The integer partition (λ) is used to generate a test error pattern (e), which is then ordered using the permutation vector ind (line 5-6). The generated test error patterns are then applied sequentially to the hard decision vector (\hat{y}), which is obtained from the input soft channel observation values (y). The resulting vector is then queried for codebook membership (line 7). If the codebook membership criterion (2) is met, then e is the guessed noise and $\hat{c} \triangleq \hat{y} \oplus e$ is the estimated codeword. Otherwise, either larger logistic weights or the remaining error patterns for that logistic weight are considered. Finally, using G^{-1} (line 8), the original message (\hat{u}) is retrieved from the estimated codeword, and the decoding process is terminated.

The frame error rate (FER) performance of ORBGRAND, a soft decision decoder, is compared to a hard decision variant GRANDAB for cyclic redundancy check (CRC) codes [32] and Random Linear Codes (RLCs) [33], [34] in Fig. 3. CRC

Algorithm 1: ORBGRAND Algorithm

Input: y , H , G^{-1} , LW_{\max}

Output: \hat{u}

- 1 $[ind, \bar{y}] \leftarrow \text{Sort}(|y|)$ // $\bar{y}_i \leq \bar{y}_j$ ($\forall i < j$)
- 2 **for** $i \leftarrow 0$ to LW_{\max} **do**
- 3 $S \leftarrow (\lambda_1, \lambda_2, \dots, \lambda_P) \vdash i$ // $P \in [1, P_{\max}]$)
- 4 **forall** j in S **do**
- 5 $e \leftarrow 0$
- 6 $e \leftarrow (e \oplus \mathbf{1}_j) \circ ind$
- 7 **if** $H \cdot (\hat{y} \oplus e)^T == 0$ **then**
- 8 $\hat{u} \leftarrow (\hat{y} \oplus e) \cdot G^{-1}$
- 9 **return** \hat{u}

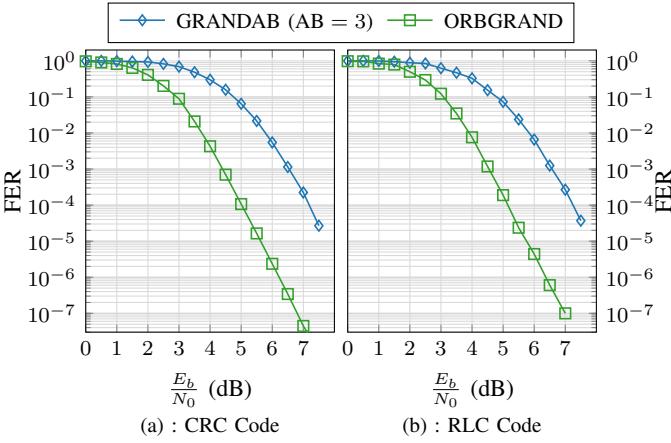


Fig. 3. Comparison of the GRANDAB and ORBGRAND decoding performance using (a) Cyclic Redundancy Check (CRC) codes and (b) Random Linear Codes (RLCs) for $n = 128$ and $k = 104$.

codes [32] are typically used to detect errors in communication systems and to assist list-based channel code decoders in selecting the final candidate codeword. On the other hand, CRC codes can also be used for error correction using the GRAND algorithm. The concept of using CRC codes for error correction with GRAND decoding was presented in [35] and expanded on in [36]. RLCs [33], [34] are linear block codes that are theoretically high-performing [33], [34] but are not considered realistic in terms of decodability. For CRC code (128,104), generator polynomial 0xB2B117 is used. As seen in Fig. 3, ORBGRAND outperforms GRANDAB by at least 2 dB for a target $FER \geq 10^{-5}$ for both CRC codes and RLCs.

To conclude, both GRAND and ORBGRAND can be used to decode any linear block code (n, k) , structured or unstructured, as long as the underlying code's parity check matrix (H) is provided. Furthermore, as a soft-input decoder, ORBGRAND outperforms hard-input GRANDAB.

III. ORBGRAND DESIGN CONSIDERATIONS

ORBGRAND is centered around generating distinct integer partition of a particular logistic weight (LW), and these integer partitions are then used to generate test error patterns (e).

An integer partition λ of a positive integer m , noted $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_P) \vdash m$ where $\lambda_1 > \lambda_2 > \dots > \lambda_P$, is the

multiset of positive integers λ_i ($\forall i \in [1, P]$) that sum to m . If all parts λ_i ($\forall i \in [1, P]$) of the integer partition are different, the partition is called distinct. Note that the generated test error pattern obtained from an integer partition with P elements has a Hamming weight of P . The ORBGRAND considers $\frac{n(n+1)}{2}$ as the maximum logistic weight for a (n, k) linear block code ($LW_{\max} = \frac{n(n+1)}{2}$). Furthermore, the generated TEPs have a maximum Hamming weight of n ($HW_{\max} = n$). It should be noted that only distinct integer partitions are taken into account when generating TEPs, and all parts (λ_i) of the partition are less than or equal to n ($\lambda_i \leq n$ ($\forall i \in [1, P]$)).

A. Parametric analysis of ORBGRAND

As seen in Algorithm 1, LW_{\max} is an important parameter for the ORBGRAND. LW_{\max} impacts ORBGRAND's decoding performance as well as its computational complexity. The computational complexity of GRAND and its variants can be expressed in terms of the number of codebook membership queries required. Furthermore, the complexity can be further subdivided into worst-case complexity, which corresponds to the maximum number of codebook membership queries required, and average complexity, which corresponds to the average number of codebook membership queries required. It should be noted that with improved channel conditions, the average complexity of GRAND and its variants decreases sharply as transmissions subject to light noise are quickly decoded [12][25][24].

In addition to LW_{\max} , another important parameter of ORBGRAND is the number of elements (P) in the generated integer partition (λ). Furthermore, P denotes the Hamming weight of the generated test error pattern. ORBGRAND's LW_{\max} and P parameters can be appropriately chosen to reduce worst-case complexity while having a minimal impact on decoding performance.

Fig. 4(a) depicts the impact of different parameters (LW_{\max} , P) on ORBGRAND decoding performance for 5G CRC-aided polar code (128,105+11) [37], [38] with BPSK modulation over an AWGN channel. Furthermore, the decoding performance of state-of-the-art soft-input decoders such as CA-SCL decoder [26], [27] and DSCF [28], [29] decoder is included for reference. The number of DSCF attempts (T_{\max}) parameter is set to 50, and the maximum bit-flipping order (ω) is set to 2. As seen in Fig. 4(a), the FER performance of ORBGRAND outperforms the GRANDAB (AB=3) [12] decoder by at least 2 dB for target FERs $\leq 10^{-5}$. Furthermore, the ORBGRAND decoder outperforms the CA-SCL decoder [26], [27] and the DSCF decoder [28], [29] for decoding polar code (128,105+11) at target FER of 10^{-4} and 10^{-6} , respectively.

The maximum number of codebook membership queries for LW_{\max} values of 128, 96, and 64 is 5.33×10^7 , 3.69×10^6 , and 1.5×10^5 , respectively. When LW_{\max} is decreased from 128 to 64, a performance degradation of 0.2 dB is observed at $FER = 10^{-7}$, as shown in Fig. 4(a). The complexity, on the other hand, is reduced by $355 \times$ as a result of this reduction. Similarly, the degradation in ORBGRAND decoding performance for the considered polar code with $LW_{\max} = 64$ is negligible when

$P = 6$ is used instead of an unbounded P . As a consequence, with $LW_{\max} = 64$ and $P = 6$, the maximum number of queries is limited to 1.16×10^5 , and the ORBGRAND (for target FER of 10^{-5}) performs similarly to the state-of-the-art DSCF polar code decoder [28], [29].

Similarly, Fig. 4(b) presents a comparison of ORBGRAND decoding performance for the BCH (127,106) code. The ORBGRAND decoding performance is compared to that of the hard decision input Berlekamp-Massey (B-M) decoder [30], [31] and the soft-input OSD [18], [19], [20] decoder. ORBGRAND decoding of BCH (127, 106) code results in a 1.7dB performance gain at a target FER of 10^{-5} when compared to B-M decoder. For target FER of 10^{-6} , the OSD decoder outperforms the ORBGRAND decoder by 0.7dB. However, due to the complexity of Gaussian elimination ($\mathcal{O}(n^3)$) and the reliance of column permutation of the \mathbf{G} matrix on the received vector from the channel (\mathbf{y}), OSD is unsuitable for parallel hardware implementation [23]. ORBGRAND, on the other hand, requires only simple bit-flipping and syndrome check (codebook membership verification) operations, making it ideal for applications requiring ultra-low decoding latency.

To conclude, the appropriate selection of ORBGRAND parameters LW_{\max} and P results not only in a reduction in computational complexity but also in the design of simple hardware, as seen in the section IV.

B. Proposed simplified generation of integer partitions (λ)

A hardware implementation for the generation of integer partitions was proposed in [39]. However, since the generated partitions are not distinct, their approach cannot be directly applied to our proposed ORBGRAND architecture. Furthermore, their integer partition generation is sequential, which is unsuitable for use in a parallelized, high-throughput hardware architecture. In this section, we propose a method for generating integer partitions of a particular logistic weight using a specific arrangement of shift registers and XOR gates.

For generating integer partitions of a specific logistic weight m , we noticed that a breakdown of m generates convenient patterns. For example, for $m = 12$ the distinct integer partitions are $\lambda = \{(12); (11, 1); (10, 2); (9, 3); (8, 4); (7, 5); (9, 2, 1); (8, 3, 1); (7, 4, 1); (6, 5, 1); (7, 3, 2); (6, 4, 2); (5, 4, 3); (6, 3, 2, 1); (5, 4, 2, 1)\}$. If a listing order is followed for $P = 2$ (i.e. the subset $\{(11, 1); (10, 2); (9, 3); (8, 4); (7, 5)\}$), the first integer descends while the second ascends. Therefore for a particular logistic weight m , integer partitions of size 2 ($P = 2$) can be generated as $(\lambda_1, \lambda_2) \vdash m$ where $\lambda_2 \in [1, \lfloor \frac{m}{2} \rfloor - 1]$ and $\lambda_1 = m - \lambda_2$.

Similar trends can be observed for higher-order partitions such as $P = 3$ (i.e. the subset $\{(9, 2, 1); (8, 3, 1); (7, 4, 1); (6, 5, 1); (7, 3, 2); (6, 4, 2); (5, 4, 3)\}$), the first integer descends while the second ascends as the third integer remains fixed until all iterations for the first two integers are complete. Hence, integer partitions of size 3 ($P = 3$) can be generated as $(\lambda_1, \lambda_2, \lambda_3) \vdash m$ where, $\lambda_3 \in [1, \lambda_3^{\max}]$, $\lambda_2 \in [\lambda_3 + 1, \lambda_{2,\lambda_3}^{\max}]$ and $\lambda_1 = m - \lambda_2 - \lambda_3$. Moreover, λ_3^{\max} is the maximum value of λ_3 , and $\lambda_{2,\lambda_3}^{\max}$ is the maximum value of λ_2 for a specific value of λ_3 ($\lambda_3 \in [1, \lambda_3^{\max}]$).

In general, an integer partition of size P can be generated as $(\lambda_1, \lambda_2, \dots, \lambda_P) \vdash m$ where $\lambda_P \in [1, \lambda_P^{\max}]$, $\lambda_i \in [\lambda_{i+1} + 1, \lambda_{i,\lambda_{i+1},\dots,\lambda_{P-1}}^{\max}] \forall i \in [2, P - 1]$ and $\lambda_1 = m - \sum_{i=2}^P \lambda_i$. Moreover, λ_P^{\max} is the maximum value of λ_P , and $\lambda_{i,\lambda_{i+1},\dots,\lambda_{P-1}}^{\max}$ is the maximum value of λ_i for specific values of λ_j ($\forall j \in [i + 1, P - 1]$). For simplicity, we will denote $\lambda_{i,\lambda_{i+1},\dots,\lambda_{P-1}}^{\max}$ as λ_i^{\max} . The maximum value for each λ_i ($\forall i \in [2, P]$) is bounded by (3).

Lemma III.1. *If a positive integer m is partitioned into P distinct parts, $\forall i \in [2, P]$, and assuming that λ_i are ordered, the maximum value for each λ_i is bounded by*

$$\lambda_i^{\max} < \frac{2 \times m - (i \times (i - 1)) + 2 - 2 \times \sum_{j=i+1}^P \lambda_j}{2 \times i}. \quad (3)$$

whereas the first value of λ is given as $\lambda_1 = m - \sum_{j=2}^P \lambda_j$.

Proof. The proof is provided in Appendix A. \square

IV. VLSI ARCHITECTURE FOR ORBGRAND

A VLSI architecture for GRANDAB (AB=3) was proposed in [35] for (n, k) linear block codes. Shift registers are used in [35] to store syndrome of error patterns with a Hamming weight of 1 (denoted as $s_i = \mathbf{H} \cdot \mathbf{1}_i^\top$, $i \in [1 \dots n]$). Furthermore, [35] employs the underlying code's linearity property to combine multiple s_i to generate syndrome of an error pattern with a Hamming weight of $l > 1$ ($s_{1,2,\dots,l} = \mathbf{H} \cdot \mathbf{1}_1^\top \oplus \mathbf{H} \cdot \mathbf{1}_2^\top \dots \oplus \mathbf{H} \cdot \mathbf{1}_l^\top$). We refer the reader to [35] for more details. The VLSI architecture for GRANDAB (AB=3) [35] forms the basis for the proposed ORBGRAND architecture. The GRANDAB decoder [35] can only generate test error patterns with Hamming weights ≤ 3 . As a result, significant improvements are needed to cater to soft-inputs, to generate error patterns in increasing order of their logistic weight, and to consider larger Hamming weights as required by ORBGRAND.

A. Scheduling and details

As explained in section III, ORBGRAND is based on generating test error patterns corresponding to integer partitions of a specific logistic weight m ($\forall m \in [3, LW_{\max}]$). Moreover, for each m , integer partitions are generated with size P ($\forall P \in [2, P_{\max}]$). We propose to generate these integer partitions in ascending order of their size (P). This modification does not impact the FER performance, however, it helps in designing a simpler hardware implementation.

In this section, we propose an arrangement and interconnection of shift registers and XOR gates to generate test error patterns corresponding to a specific logistic weight m . The shift registers store the syndromes that correspond to error patterns with a Hamming weight of 1 (s_i). To check for error patterns of hamming weight P , these syndromes are combined using an array of XOR gates.

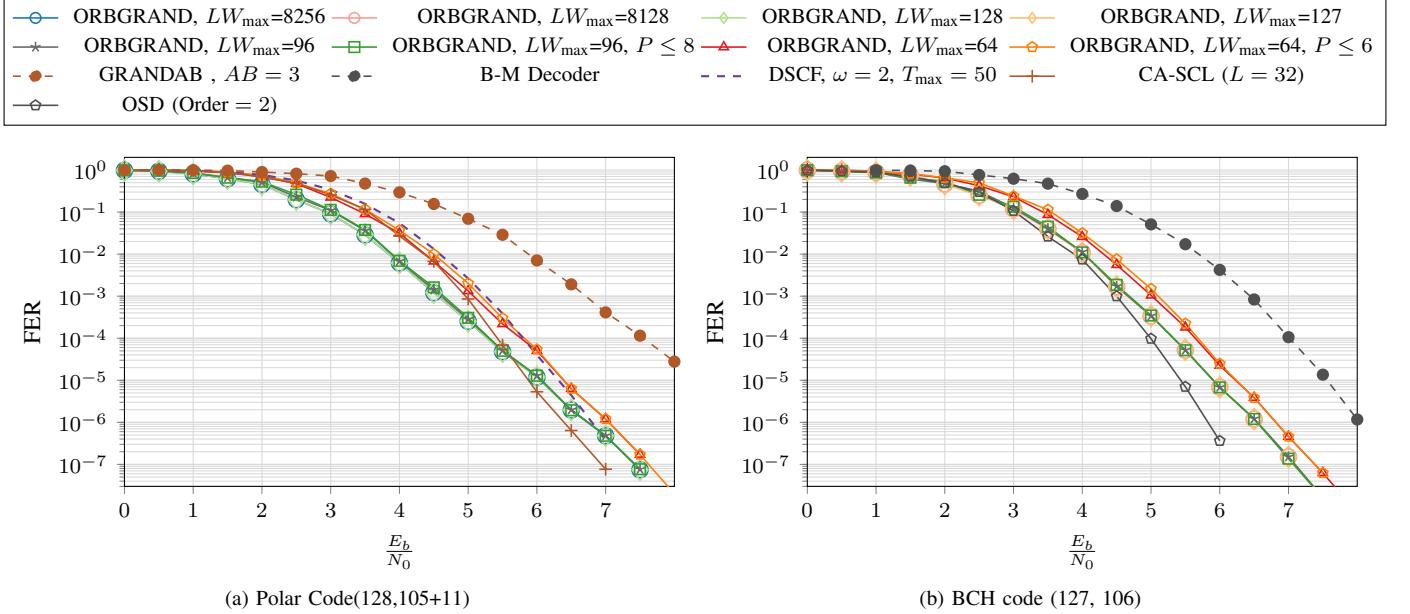


Fig. 4. Comparison of decoding performance of ORBGRAND decoding with different parameters (LW_{\max} , P) for 5G CRC-aided Polar Code (128,105+11) and BCH code (127, 106).

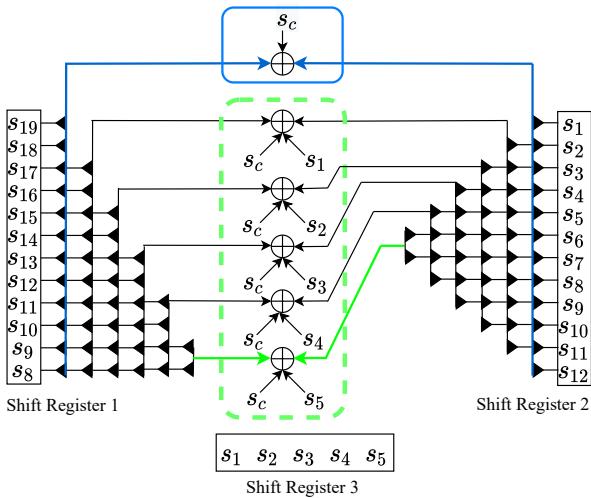


Fig. 5. Example of the shift registers content and interconnection for logistic weight $m = 20$ for checking error patterns of Hamming weights 2 and 3.

B. Generating test error patterns for $P \leq 3$

The size and number of shift registers used in [35] have a direct impact on the Hamming weight of the error patterns that can be evaluated in parallel. For example, in [35], two $n \times (n - k)$ shift registers are used to evaluate n test error patterns in parallel with a Hamming weight of 2. However, if more shift registers are added, the number of interconnections becomes a problem. As a result, for the proposed ORBGRAND architecture, we choose three shift registers that correspond to an integer partition of size 3 ($P = 3$).

In the proposed ORBGRAND VLSI architecture, λ_1 , λ_2 , λ_3 ($(\lambda_1, \lambda_2, \lambda_3) \vdash m$) are mapped to first, second and third shift register respectively. The third shift register is a $\lambda_3^{\max} \times (n - k)$ bit shift register, where λ_3^{\max} value is given by (3)

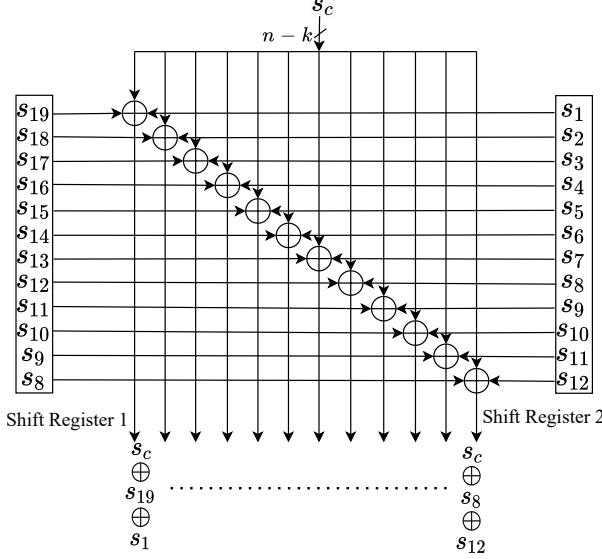
corresponding to $P = 3$. Whereas the first and second shift registers are each $2 \times (\lambda_3^{\max} + 1) \times (n - k)$ bits in size. Since we have $\lambda_1 = m - \sum_{i=2}^3 \lambda_i$, corresponding to $P = 3$, s_{m-i} is stored at the i^{th} index of the first shift register, while for the second and third shift registers s_i is stored at the i^{th} index.

Fig. 5 shows an example of the content and interconnection of three shift registers for logistic weight $m = 20$. The elements of the three shift registers are syndromes (s_i) of the error pattern with Hamming weight of 1. These syndromes (s_i) of the error pattern with Hamming weight of 1 are combined using an array of $(n - k)$ -wide XOR gates to check for error patterns with Hamming weights 2 and 3.

A collection of these connections is defined as a *bus*. Since there are numerous connections and XOR gates involved, we used a single XOR gate and a single *bus* symbol to illustrate these interconnections in Fig. 5. As seen in Fig. 5, there are 6 buses ($\lambda_3^{\max} + 1$, where $\lambda_3^{\max} = 5$ for $P = 3$ (3)) for $m = 20$. The first bus (highlighted by solid rectangle) is used to check error patterns with Hamming weight of 2, and the remaining buses (highlighted by the dashed rectangle in Fig. 5) are used to check for error patterns of Hamming weight 3.

To check the error patterns corresponding to a Hamming weight 2 ($P = 2$), the first bus (highlighted by solid rectangle) is used to combine all the elements of shift register 1 with all the elements of the shift register 2 using an array of XOR gates. These results are again combined with the syndromes of the received vector (s_c) to check for the error patterns with Hamming weight of 2. The detailed interconnections and the associated XOR gates for the first bus are shown in Fig. 6a.

Similarly, to check the error patterns corresponding to a Hamming weight of 3 ($P = 3$), the selected elements of the shift register 1 and 2 are again combined with s_c , but also with the elements of the shift register 3. We use a single bus and a single XOR gate to illustrate these interconnections, which are depicted in Fig. 5 by the dashed rectangle. The detailed



(a) Interconnections and the associated XOR gates for the first bus for checking error patterns of Hamming weight of 2 ($P = 2$).

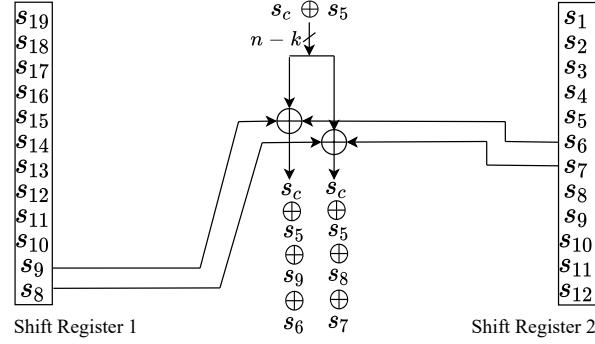


Fig. 6. Example of interconnections and the associated XOR gates for the first and last bus for logistic weight $m = 20$.

interconnections and the associated XOR gates for the last (6^{th}) bus are shown in Fig. 6b.

Due to the described arrangement and interconnection of the shift registers and XOR gates, all the error patterns corresponding to an integer partition of sizes 2 and 3 for a specific logistic weight m are checked in one time-step. In general, to check the error patterns corresponding to an integer partition of sizes 2 and 3 for any logistic weight m , the content and the interconnection of the three shift registers are depicted in Fig. 7.

C. Generating test error patterns for $P > 3$

To check all the test error patterns corresponding to integer partitions of sizes $P > 3$, a controller is used in conjunction with the shift registers. The controller combines $P_{\max} - 3$ syndromes together with the syndromes of the received vector, noted s_{comp} . Hence, when s_{comp} is fixed, only one time-step is required to generate all possible combinations of $\{\lambda_1, \lambda_2, \lambda_3\}$ using the shift registers with adequately chosen shift values.

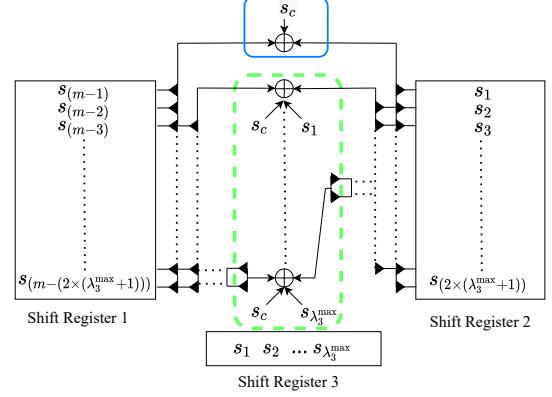


Fig. 7. Shift registers contents for checking error patterns corresponding to a Hamming weight of 2 and 3 for any logistic weight m .

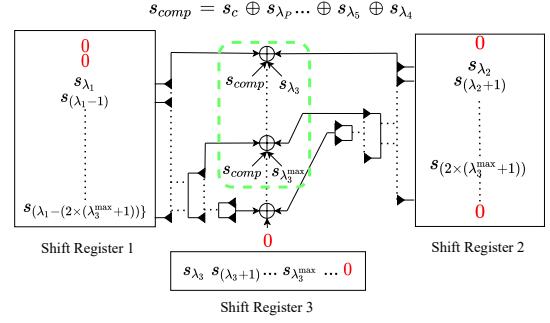
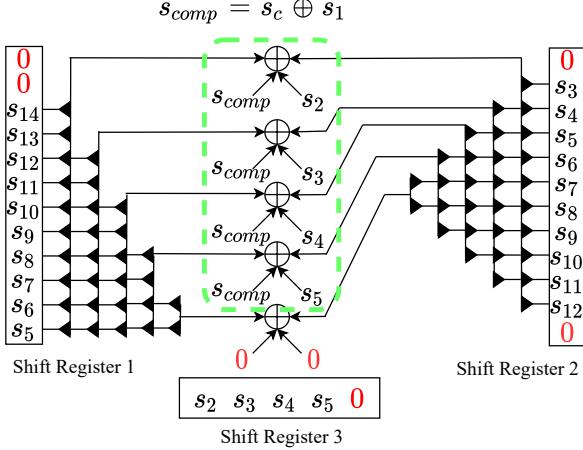


Fig. 8. Shift registers contents for checking error patterns corresponding to $P > 3$ for any logistic weight m ($\lambda_1 = m - \sum_{i=2}^P \lambda_i$).

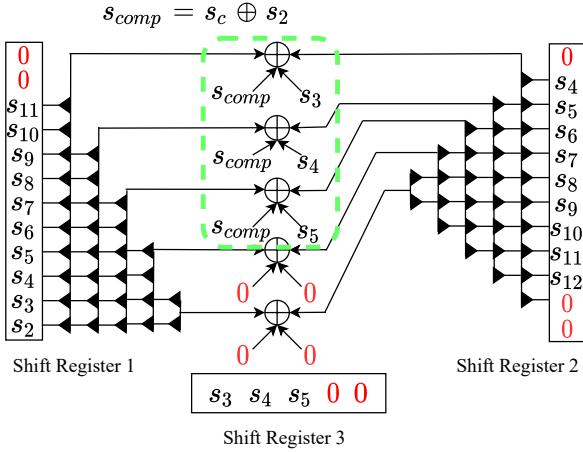
The content and the interconnection of the three shift registers, which are used to check the test error patterns corresponding to integer partitions of sizes $P > 3$, are depicted in Fig. 8. Since the first bus is only used to check error patterns with Hamming weight of 2 ($P = 2$), it is disabled for $P > 3$ and not shown in Fig. 8. A 0 corresponds to a disabled connection, which means the respective elements of the bus, do not take part in the final computations. Fig. 9 illustrates testing error patterns corresponding to $P = 4$. At each time step, the controller outputs $s_{comp} = s_c \oplus s_{\lambda_4}$ ($\lambda_4 \in [1, \lambda_4^{\max}]$) and $\{\lambda_1, \lambda_2, \lambda_3\}$ are computed and mapped to their corresponding shift registers.

At the first time step, having received $s_{comp} = s_c \oplus s_1$ ($\lambda_4 = 1$) as an output from the controller, λ_3 ($\lambda_3 \in [2, \lambda_3^{\max}]$ where $\lambda_{3,\lambda_4}^{\max} = 5$ with $\lambda_4 = 1$ (3)) is computed and mapped to the third shift register. Similarly, λ_2 ($\lambda_2 \in [\lambda_3 + 1, \lambda_{(2 \times (\lambda_3^{\max} + 1))}]$) and λ_1 ($\lambda_1 = m - \sum_{i=2}^4 \lambda_i$) are computed and mapped to their corresponding shift registers. The test error patterns with $\lambda_4 = 1$ are checked as shown in Fig. 9a.

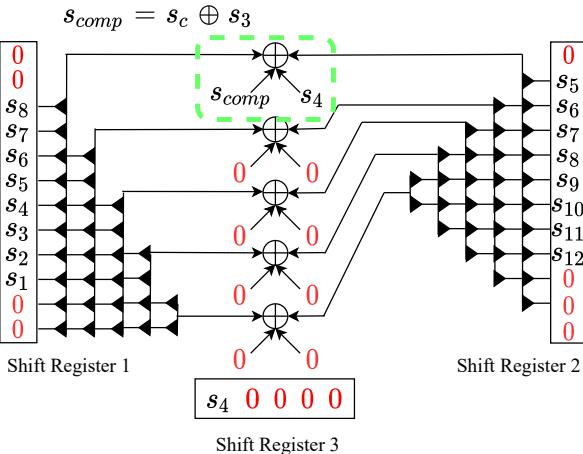
At the next time step, the controller outputs $s_{comp} = s_c \oplus s_2$ ($\lambda_4 = 2$) and λ_3 ($\lambda_3 \in [3, \lambda_{3,\lambda_4}^{\max}]$ where $\lambda_{3,\lambda_4}^{\max} = 5$ with $\lambda_4 = 2$ (3)) is computed. Shift register 2 is shifted up by 1 position and shift register 1 outputs λ_1 ($\lambda_1 = m - \sum_{i=2}^4 \lambda_i$) as shown in Fig. 9b. Hence, the test error patterns with $\lambda_4 = 2$ are checked in the second time-step. Similarly, at third time step, the controller outputs $s_{comp} = s_c \oplus s_3$, ($\lambda_4 = 3$) λ_3 ($\lambda_3 \in$



(a) Shift registers contents for checking test error patterns corresponding to $P = 4$ at time step 1.



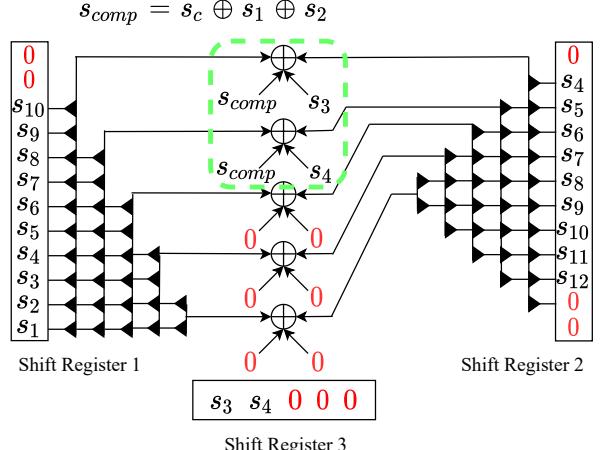
(b) Shift registers contents for checking test error patterns corresponding to $P = 4$ at time step 2.



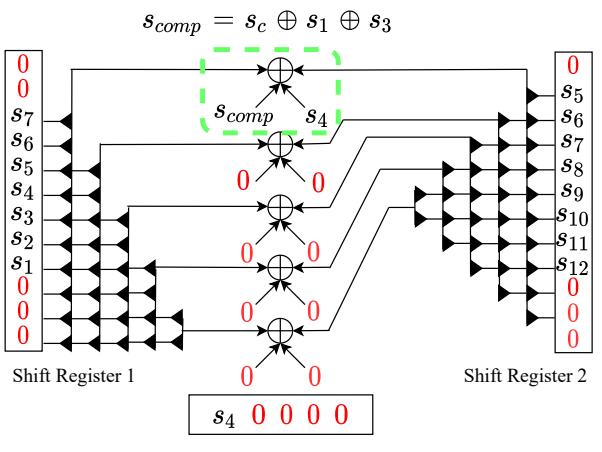
(c) Shift registers contents for checking test error patterns corresponding to $P = 4$ at time step 3.

Fig. 9. Shift registers contents for checking test error patterns corresponding to $P = 4$ ($m = 20$).

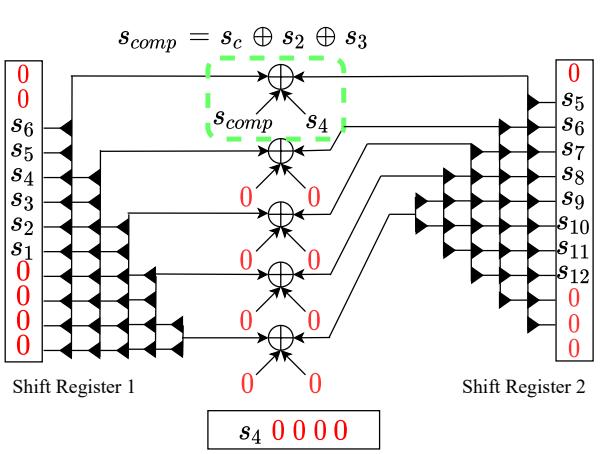
$[4, \lambda_{3,\lambda_4}^{max}]$ where $\lambda_{3,\lambda_4}^{max} = 4$ with $\lambda_4 = 3$ (3)) is computed as shown in Fig. 9c. Therefore, a total of 3 time steps ($\lambda_4^{max} = 3$, Eq. 3), are required to check for error patterns corresponding



(a) Shift registers contents for checking test error patterns corresponding to $P = 5$ at time step 1.



(b) Shift registers contents for checking test error patterns corresponding to $P = 5$ at time step 2.



(c) Shift registers contents for checking test error patterns corresponding to $P = 5$ at time step 3.

Fig. 10. Shift registers contents for checking test error patterns corresponding to $P = 5$ ($m = 20$).

to $P = 4$ and $m = 20$.

Fig. 10 depicts the use of shift registers to check the error

patterns corresponding to $P = 5$ and $m = 20$. At each time step, the controller outputs $s_{comp} = s_c \oplus s_{\lambda_5} \oplus s_{\lambda_4}$. For each value of λ_5 ($\lambda_5 \in [1, \lambda_5^{max}]$), λ_4 ($\lambda_4 \in [\lambda_5 + 1, \lambda_{4,\lambda_5}^{max}]$) is computed. Similarly, for each value of λ_4 , λ_3 ($\lambda_3 \in [\lambda_4 + 1, \lambda_{3,\lambda_5,\lambda_4}^{max}]$), λ_2 ($\lambda_2 \in [\lambda_3 + 1, \lambda_{(2 \times (\lambda_3^{max} + 1))}^{max}]$) and λ_1 ($\lambda_1 = m - \sum_{i=2}^5 \lambda_i$) are computed and mapped to their corresponding shift registers. Hence, a total of 3 time steps ($\sum_{\lambda_5=1}^{\lambda_P^{max}} \left(\sum_{\lambda_4=\lambda_5+1}^{\lambda_{P-1,\lambda_P}^{max}} (1) \right)$, where $\lambda_5^{max} = 2$, $\lambda_{4,\lambda_5=1}^{max} = 3$ and $\lambda_{4,\lambda_5=2}^{max} = 3$ (3)) are required to check for error patterns corresponding to $P = 5$ and $m = 20$ as shown in Fig. 10. In general, the number of time steps required to generate all integer partitions of size $P > 3$ for a specific logistic weight (LW) is given by:

$$\sum_{\lambda_P=1}^{\lambda_P^{max}} \left(\sum_{\lambda_{P-1}=\lambda_P+1}^{\lambda_{P-1,\lambda_P}^{max}} \left(\dots \sum_{\lambda_4=\lambda_5+1}^{\lambda_{4,\lambda_5,\dots,\lambda_P}^{max}} (1) \right) \right). \quad (4)$$

D. Proposed VLSI architecture

Figure 11 depicts the proposed VLSI architecture for ORBGRAND which can be used to decode any linear block code of length n . For clarity, the control and clock signals are not shown. To support different codes and rates, any H matrix can be loaded into the H memory of size $(n - k) \times n$ -bit at any time. The hard decided vector \hat{y} is subjected to a syndrome check (2) in the first phase of decoding. Decoding is assumed to be successful if the syndrome is verified. Otherwise, the LLRs values are sorted in ascending order of their absolute value $|y|$.

As depicted in Fig. 11, the sorted syndromes of error patterns with Hamming weight of 1 (s_i) are passed to the *decoder core*, while the indices of the sorted LLRs are forwarded to the multiplexers for later use by the *word generator* module. Following the sorting process, all syndromes of error patterns with Hamming weight of 1 (s_i) are tested for codebook membership (2) in a single time-step. Following that, error patterns are tested for codebook membership in ascending logistic weight (LW) order as explained in section II-C.

The test error pattern syndromes corresponding to integer partitions of a given logistic weight m are generated using the shift register and XOR gate arrangement proposed in section IV-A. The rows of shift registers are combined with the controller's output (s_{comp}), and the resulting test syndromes are NOR-reduced and fed to a 2D priority encoder. Each NOR-reduce output is 1 if and only if all of the bits of the syndromes computed by (2) are 0. If any of the tested syndrome combinations satisfy the parity check constraint (NOR-reduced output is 1), the 2D priority encoder is used in conjunction with the *controller* module to forward the respective indices to the word generator module, where P multiplexers are used to convert the sorted index values to their appropriate bit-flip locations.

V. ORBGRAND DESIGN EXPLORATION

In this section, we present the VLSI implementation results for ORBGRAND (LW, P). Initially, implementation results

for baseline ORBGRAND, which can support parameters $LW \leq 64$ and $P \leq 6$, are presented. Please note that a subset of these implementation results were previously presented in [1]. In this work, baseline implementation results are supplemented with power consumption, area, and energy efficiency results. Furthermore, the proposed ORBGRAND VLSI architecture is extended to support the parameters $LW \leq 96$ and $P \leq 8$, and a comprehensive analysis of worst-case latency and worst-case throughput for selecting different parameters (LW, P) for the proposed ORBGRAND hardware is presented. Finally, the sorter module for the proposed ORBGRAND is segmented into multiple partitions to reduce the area overhead. The effect of the number of partitions on ORBGRAND decoding performance as well as area overhead is also presented and compared to the ORBGRAND with a non-segmented sorter approach.

A. ORBGRAND baseline implementation

The proposed ORBGRAND VLSI architecture with parameters $LW \leq 64$ and $P \leq 6$ has been implemented in Verilog HDL and synthesized using Synopsys Design Compiler with general-purpose TSMC 65 nm CMOS technology. The design has been verified using test benches generated via the bit-true C model of the proposed hardware. Table I presents the synthesis results for the proposed decoder with $n = 128$ and the proposed architecture can support code rates between 0.75 and 1. Input channel LLRs are quantized on 5 bits, including 1 sign bit and 3 bits for the fractional part. To ensure accuracy in power measurements, switching activities from real test vectors are extracted for all of the VLSI architectures presented in Table I.

The maximum frequency supported by the ORBGRAND implementation is 454 MHz. Since there is no pipelining technique for the decoder core, one clock cycle corresponds to one time-step. The average decoding latency of the proposed hardware is calculated using the bit-true C model after taking account for at least 100 frames in error for each $\frac{E_b}{N_0}$ point. At target FER of 10^{-7} ($\frac{E_b}{N_0} > 7.5$ dB), the average latency is only 2.47ns, resulting in an average decoding information throughput of 42.5 Gbps for a (128,105) polar code. However, the worst-case (W.C.) scenario needs 4226 cycles with $n = 128$ and parameters $LW \leq 64$ and $P \leq 6$, culminating in a W.C. latency of $9.3\mu s$.

As seen in Fig. 4 (a), ORBGRAND with parameters $LW \leq 64$ and $P \leq 6$ has similar decoding performance (target FER of 10^{-5}) to the Dynamic SC-Flip (DSCF) [28] decoder for (128,105) 5G-NR CRC-Aided (CA) polar code. The proposed ORBGRAND VLSI implementation ($LW \leq 64$ and $P \leq 6$) is compared to VLSI architecture for DSCF polar code decoder ($\omega = 2$, $T_{max} = 50$) [29], which employs 7 and 6 bit internal and channel LLR quantizations, respectively. Compared to DSCF [29], ORBGRAND ($LW \leq 64$, $P \leq 6$) has a $8 \times$ area overhead, as well as a 52% increase in the worst-case latency. However, at a target FER of 10^{-7} , the proposed ORBGRAND results in $49 \times$ higher average throughput than the DSCF [29]. Furthermore, as compared to DSCF [29], ORBGRAND ($LW \leq 64$, $P \leq 6$) is $5 \times$ more area efficient and $32 \times$ more energy efficient. Moreover, the proposed ORBGRAND hardware

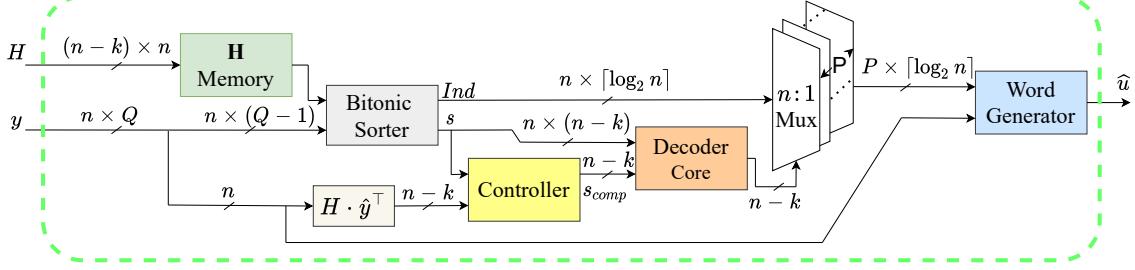
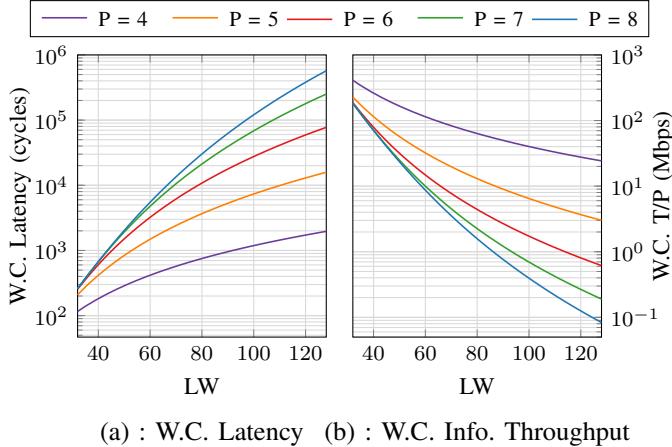


Fig. 11. Proposed VLSI Architecture for ORBGRAND.

TABLE I
TSMC 65 NM CMOS SYNTHESIS COMPARISON FOR ORBGRAND WITH GRANDAB AND DSCF FOR $n = 128$.

	GRANDAB [35]	ORBGRAND				DSCF [29]
Parameters	AB=3	LW≤64, P≤6	LW≤96, P≤8	LW≤96, P≤8, S = 2	LW≤96, P≤8, S = 4	$\omega = 2, T_{max} = 50$
Technology (nm)	65	65	65	65	65	65
Supply (V)	0.9	0.9	0.9	0.9	0.9	0.9
Max. Frequency (MHz)	500	454	454	454	454	426
Area (mm ²)	0.25	1.82	2.25	2.08	1.85	0.22
W.C. Latency (μs)	8.196	9.30	205.76	205.76	205.76	6.103
Avg. Latency (ns)	2	2.47	2.47	2.47	2.47	122
W.C. T/P (Mbps) ^a	12.8	11.3	0.51	0.51	0.51	17.2
Avg. T/P (Gbps) ^a	52.5	42.5	42.5	42.5	42.5	0.86
Power (mW)	46	104.3	133	131.3	130	68.51
Energy per Bit (pJ/bit) ^b	0.87	2.45	3.13	3.09	3.0	79.6
Area Efficiency (Gbps/mm ²) ^c	210	23.3	18.9	20.4	23	3.9
Code compatible	Yes	Yes	Yes	Yes	Yes	No

^a Information Throughput (Gbps) = $\frac{k}{\text{Decoding Latency (ns)}}$, ^b Energy per Bit (pJ/bit) = $\frac{\text{Power (mW)}}{\text{Avg. Throughput (Gbps)}}$, ^c Area Efficiency (Gbps/mm²) = $\frac{\text{Avg. Throughput (Gbps)}}{\text{Area (mm²)}}$

Fig. 12. Worst-Case (W.C.) latency and W.C. information throughput for the proposed ORBGRAND architecture with various parameters (LW, P).

is code and rate compatible, while the DSCF [29] can only decode polar codes.

In comparison to the hard-input GRANDAB decoder (AB=3) [35], ORBGRAND ($LW \leq 64, P \leq 6$) has a 7× area overhead, as well as a 13.5% higher W.C. and a 23.5% higher average latency. Furthermore, as compared to [35], ORBGRAND ($LW \leq 64, P \leq 6$) is 2× less energy efficient and 9× less area efficient. However, as seen in Fig. 4, the FER performance of ORBGRAND ($LW \leq 64, P \leq 6$), a soft decision decoder, outperforms hard-input counterpart decoders by at least 1.3 ~ 2dB for target FERs $\leq 10^{-5}$.

TABLE II
DISPLACEMENT OF LLR ELEMENTS y_i ($\forall i \in [1, n]$) FROM THEIR CORRECT LOCATIONS WITH SEGMENTED SORTER

Displacement	# of segments for bitonic sorter			
	S = 2	S = 4	S = 8	S = 16
= 0	10.31%	5.98%	3.87%	2.59%
≤ 1	29.40%	17.42%	11.40%	7.67%
≤ 2	45.50%	28.18%	18.67%	12.65%
≤ 3	58.76%	38.05%	25.67%	17.50%
≤ 5	77.84%	54.62%	38.65%	26.85%
≤ 10	96.89%	81.64%	63.82%	47.68%
≤ 20	99.99%	98.34%	90.09%	75.95%
≤ 30	100%	99.94%	98.10%	90.58%

B. Design expansion and latency analysis

As illustrated in Fig. 4, the ORBGRAND with parameters $LW \leq 96$ and $P \leq 8$ has similar decoding performance to the ORBGRAND with parameters $LW_{max} = 8256$ and $LW_{max} = 8128$ ($LW_{max} \leq \frac{n(n+1)}{2}$ [24]) for both (128, 105) Polar code and (127, 106) BCH code. Furthermore, at a target FER of $\leq 10^{-7}$, ORBGRAND with parameters $LW \leq 96$ and $P \leq 8$ results in a 0.2 ~ 0.3dB gain in decoding performance when compared to ORBGRAND with parameters $LW \leq 64$ and $P \leq 6$.

Table I presents the VLSI implementation results for the proposed ORBGRAND VLSI architecture with parameters $LW \leq 96$ and $P \leq 8$ using the same implementation settings described in section V-A. The proposed ORBGRAND can support a maximum frequency of 454 MHz. As shown in Table I, the ORBGRAND implementation with parameters $LW \leq 96$

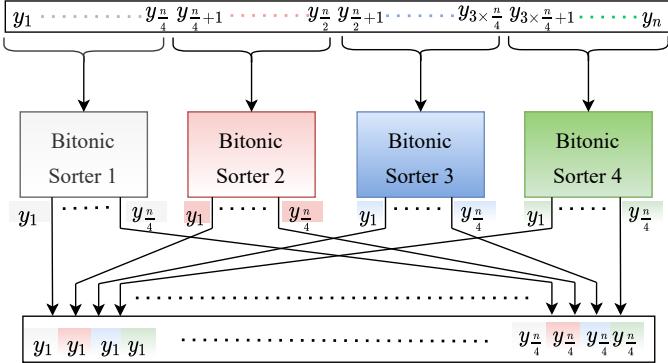


Fig. 13. Proposed segmented sorter ($S = 4$) for ORBGRAND.

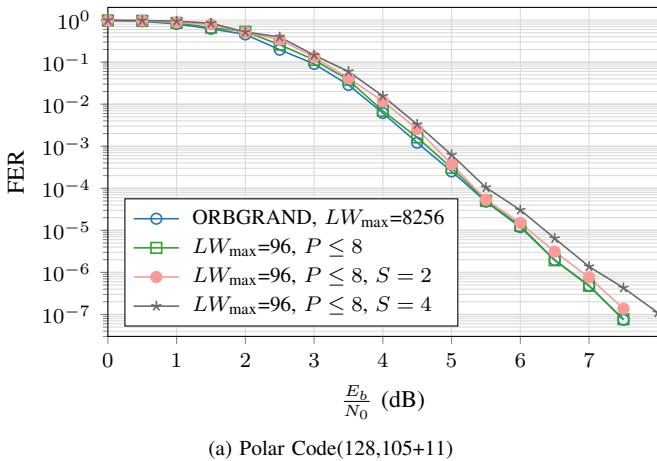


Fig. 14. Comparison of decoding performance of ORBGRAND decoding with different parameters (LW_{max} , P , S) for Polar Code(128,105+11)

and $P \leq 8$ incurs a 23.6% area overhead when compared to the ORBGRAND implementation with parameters $LW \leq 64$ and $P \leq 6$. Furthermore, the ORBGRAND implementation with parameters $LW \leq 96$ and $P \leq 8$ is 18.8% less area efficient and 27.7% less energy efficient than the ORBGRAND implementation with parameters $LW \leq 64$ and $P \leq 6$.

The ORBGRAND parameters LW and P influence the decoding performance as well as the worst-case decoding latency of the proposed ORBGRAND VLSI hardware. In the worst-case scenario, the ORBGRAND with parameters $LW \leq 64$ and $P \leq 6$ requires 4226 cycles ($n = 128$), whereas the ORBGRAND with parameters $LW \leq 96$ and $P \leq 8$ requires 93417 cycles, resulting in a worst-case latency of $205.76\mu s$. Figure 12 (a) depicts the worst-case latency (in clock cycles (4)) of the proposed ORBGRAND hardware for various LW and P parameter values. Fig. 12 (b) depicts the information throughout corresponding to $k = 105$ and a maximum frequency of 454 MHz for the proposed ORBGRAND (LW, P) hardware.

To conclude, the ORBGRAND implementation parameters (LW, P) can be appropriately chosen to strike a balance between area overhead, energy budget, and decoding performance requirements for a target application.

C. ORBGRAND area optimization

In this section, we investigate the sorter module in the proposed ORGRAND (LW, P) VLSI implementation and propose segmenting the sorter module into multiple partitions to reduce the area overhead of ORBGRAND hardware. The ORBGRAND decoding procedure, as described in Algorithm 1, begins by sorting channel LLRs (y) in ascending order of their absolute value ($|y|$). Any sorter [40] (*insertion sorter*, *merge sorter*, *bubble sorter*) may be used to sort $|y|$ for the proposed ORBGRAND hardware. The sorter choice is determined by the target application's budget in terms of decoding latency and hardware overhead. On one end of the spectrum, we have sequential sorters with high latency but low hardware implementation cost, while on the other, we have parallel sorters with low latency but high hardware implementation cost.

The proposed ORBGRAND VLSI implementation employs a *bitonic sorter* [41] of length n that is pipelined to $\log_2(n)$ stages. As a result, the sorting procedure requires just $\log_2(n)$ clock cycles. The *bitonic sorter* module of the proposed ORBGRAND VLSI implementation can be partitioned into multiple segments to reduce the ORBGRAND implementation's area overhead. The size and number of partitions influence ORBGRAND decoding performance as well as the area overhead of the proposed ORBGRAND VLSI implementation. A *bitonic sorter* module of length n is segmented into S segments, each having a size $\frac{n}{S}$, for the proposed segmented sorter approach. Please note that the number of segments S should be chosen in such a way that the size of each segment $\frac{n}{S}$ is an integer. The segmented *bitonic sorter* is depicted in Fig. 13, which employs four sorters ($S = 4$) of length $\frac{n}{4}$, each of which receives an unique subset of channel LLRs (y). To generate the final LLRs, the sorted LLRs from individual sorters are concatenated. The first four elements of the final sorted LLRs are comprised of the first element of the output of each sorter. Similarly, the second element of each sorter's output occupies the following four positions of the final sorted LLRs. This procedure is continued until the last elements of each sorter's output are placed in the last four positions of the sorted LLRs, as shown in Fig. 13.

A non-segmented sorter will sort the LLR elements $|y_i|$ ($\forall i \in [1, n]$) to their correct location. However, compared to a non-segmented sorter, the sorted LLRs using a segmented sorter ($S > 1$) will have elements that are displaced from their correct locations. To investigate the effect of segments on the displacement of LLR elements from their correct locations, we performed Monte-Carlo simulations and measured the percentage of LLR elements that were within a specified distance of their correct location. Table II compares the displacement of LLR elements from their correct locations with varying numbers of segments employed in the segmented-sorter approach. Table II shows that as the number of segments decreases, more elements are concentrated closer to their correct locations, but as the number of segments increases, LLR elements are concentrated further away from their correct locations.

Fig. 14 depicts the FER performance of implementing a

segmented sorter approach for ORBGRAND ($LW \leq 96$, $P \leq 8$) decoding of (128,105) polar code. As seen in the Figure 14, the ORBGRAND using the segmented sorter with $S = 2$ and $S = 4$ suffers from a FER performance degradation of 0.1dB and 0.3dB respectively, at the target FER of 10^{-6} , as compared to ORBGRAND with a non-segmented sorter. Table I compares VLSI implementation results for the ORBGRAND ($LW \leq 96$, $P \leq 8$) using the non-segmented sorter to the proposed ORBGRAND ($LW \leq 96$, $P \leq 8$) using segmented sorter approach. As shown in Table I, the proposed ORBGRAND ($LW \leq 96$, $P \leq 8$) with non-segmented sorter incurs an area overhead of 8% and 21.6%, respectively, when compared to ORBGRAND with segmented sorter parameters $S = 2$ and $S = 4$. To conclude, the number of sorter segments influences both decoding performance and the area overhead of the ORBGRAND hardware; they can be chosen appropriately to strike a balance between decoding performance requirements and area overhead for a target application.

VI. CONCLUSION

In this work, we present a hardware architecture for the ORBGRAND algorithm. ORBGRAND is a soft input GRAND variant that generates test error patterns in a fixed logistic weight order, rendering it suitable for parallel hardware implementation. Due to the code-agnostic nature of the GRAND and its variants, the proposed ORBGRAND architecture can decode any code as long as the length and rate constraints are met. We suggest modifications in the ORBGRAND algorithm to simplify the hardware implementation and reduce the decoding complexity. Furthermore, the proposed ORBGRAND VLSI architecture uses parameters that can be tweaked to meet the optimal decoding performance as well as the decoding latency for a specific application. According to ASIC synthesis results, an average decoding throughput of 42.5 Gbps can be achieved for a code length of 128 and a target FER of 10^{-7} . The proposed VLSI architecture improves decoding performance by at least 2 dB over the GRANDAB, a hard-input variant of GRAND. In comparison to the state-of-the-art DSCF hardware decoder for 5G (128, 105) polar code, the proposed VLSI implementation achieves $49 \times$ higher decoding throughput, $32 \times$ higher energy efficiency and $5 \times$ higher area efficiency. Finally, the proposed architecture is the first step toward implementing GRAND family soft-input decoders in hardware.

APPENDIX A PROOF OF LEMMA 1

Proof. It is sufficient to show that for all i ($i \in [2, P]$)

$$\lambda_i^{\max} < \frac{2 \times m - (i \times (i-1)) + 2 - 2 \times \sum_{j=i+1}^P \lambda_j}{2 \times i}. \text{ We use induction on } i.$$

Base case ($i = 2$):

$$\lambda_2^{\max} < \frac{m - \sum_{j=3}^P \lambda_j}{2}$$

Since the λ_i are ordered $\therefore \lambda_2 < \lambda_1$

$$\begin{aligned} \Rightarrow \lambda_2^{\max} &< m - \sum_{j=2}^P \lambda_j \quad (\because \lambda_1 = m - \sum_{j=2}^P \lambda_j) \\ \Rightarrow \lambda_2^{\max} &< m - \sum_{j=3}^P \lambda_j - \lambda_2^{\max} \\ \therefore \lambda_2^{\max} &< \frac{m - \sum_{j=3}^P \lambda_j}{2} \end{aligned}$$

Inductive hypothesis ($i = k$):

$$\lambda_k^{\max} < \frac{2 \times m - (k \times (k-1)) + 2 - 2 \times \sum_{j=k+1}^P \lambda_j}{2 \times k}$$

Inductive step ($i = k+1$):

$$\lambda_{k+1}^{\max} < \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times (k+1)}$$

\therefore Inductive hypothesis

$$\begin{aligned} \lambda_k^{\max} &< \frac{2 \times m - (k \times (k-1)) + 2 - 2 \times \sum_{j=k+1}^P \lambda_j}{2 \times k} \\ \Rightarrow \lambda_k^{\max} &< \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+1}^P \lambda_j}{2 \times k} + 1 \\ &\quad (\because k \times (k-1) = k \times (k+1) + 2 \times (k)) \\ \Rightarrow \lambda_k^{\max} &< \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times k} \\ &\quad + 1 - \frac{\lambda_{k+1}^{\max}}{k} \\ &\quad (\because \sum_{j=k+1}^P \lambda_j = \lambda_{k+1}^{\max} + \sum_{j=k+2}^P \lambda_j) \\ \Rightarrow \lambda_{k+1}^{\max} &+ \frac{\lambda_k^{\max} - k}{k} \\ &< \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times k} \end{aligned}$$

$$\begin{aligned} \Rightarrow \frac{\lambda_{k+1}^{\max} + \lambda_k^{\max} - k}{k+1} \\ &< \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times (k+1)} \quad (5) \end{aligned}$$

Since λ_i are ordered $\therefore \lambda_{k+1} < \lambda_k$

$$\Rightarrow \lambda_{k+1}^{\max} + 1 \leq \lambda_k^{\max} \quad (\because \lambda_{k+1}^{\max} + 1 \leq \lambda_k \leq \lambda_k^{\max})$$

$$\begin{aligned}
&\Rightarrow k \times \lambda_{k+1}^{\max} + k + \lambda_{k+1}^{\max} \leq k \times \lambda_k^{\max} + \lambda_{k+1}^{\max} \\
&\Rightarrow \lambda_{k+1}^{\max} \leq \frac{\lambda_{k+1}^{\max} + \lambda_k^{\max} - k}{k+1} \\
&\text{Using (5): } \lambda_{k+1}^{\max} \leq \frac{\lambda_{k+1}^{\max} + \lambda_k^{\max} - k}{k+1} \\
&< \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times (k+1)} \\
&\Rightarrow \lambda_{k+1}^{\max} < \frac{2 \times m - (k \times (k+1)) + 2 - 2 \times \sum_{j=k+2}^P \lambda_j}{2 \times (k+1)} \\
&\quad (\because a \leq b < c \Rightarrow a < c)
\end{aligned}$$

- [17] G. Durisi, T. Koch, and P. Popovski, "Toward massive, ultrareliable, and low-latency wireless communication with short packets," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1711–1726, 2016.
- [18] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," vol. 41, no. 5, pp. 1379–1396, 1995.
- [19] C. Yue, M. Shirvanimoghaddam, Y. Li, and B. Vucetic, "Segmentation-discarding ordered-statistic decoding for linear block codes," in *2019 IEEE Global Communications Conference, GLOBECOM 2019, Waikoloa, HI, USA, December 9-13, 2019*, 2019, pp. 1–6, IEEE.
- [20] Johannes Van Wonterghem, Amira Alloum, Joseph Jean Boutros, and Marc Moeneclaey, "On performance and complexity of osd for short error correcting codes in 5g-nr," 2017.
- [21] J.V. Wonterghem, A. Alloum, J.J. Boutros, and M. Moeneclaey, "On short-length error-correcting codes for 5g-nr," *Ad Hoc Networks*, vol. 79, pp. 53–62, 2018.
- [22] W. Jin and M. Fossorier, "Efficient box and match algorithm for reliability-based soft-decision decoding of linear block codes," in *2007 Information Theory and Applications Workshop*, 2007, pp. 160–169.
- [23] S. Scholl, C. Stumm, and N. Wehn, "Hardware implementations of Gaussian elimination over GF(2) for channel decoding algorithms," in *2013 Africon*, 2013, pp. 1–5.
- [24] Ken R. Duffy, "Ordered reliability bits guessing random additive noise decoding," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8268–8272.
- [25] A. Solomon, K. R. Duffy, and M. Médard, "Soft maximum likelihood decoding using grand," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.
- [26] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.
- [27] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, 2015.
- [28] L. Chandresris, V. Savin, and D. Declercq, "Dynamic-scflip decoding of polar codes," *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2333–2345, 2018.
- [29] F. Ercan, T. Tonnellier, N. Doan, and W. J. Gross, "Practical dynamic SC-flip polar decoders: Algorithm and implementation," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5441–5456, 2020.
- [30] E. Berlekamp, "Nonbinary BCH decoding (abstr.)," *IEEE Transactions on Information Theory*, vol. 14, no. 2, pp. 242–242, 1968.
- [31] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.
- [32] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.
- [33] Robert G. Gallager, "Information theory and reliable communication," 1968.
- [34] J.T. Coffey and R.M. Goodman, "Any code of which we cannot think is good," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1453–1461, 1990.
- [35] S. M. Abbas, T. Tonnellier, F. Ercan, and W. J. Gross, "High-throughput VLSI architecture for GRAND," in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, 2020, pp. 1–6.
- [36] A. Wei, M. Médard, and K. R. Duffy, "Crc codes as error correction codes," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.
- [37] K. R. Duffy, A. Solomon, K. M. Konwar, and M. Médard, "5G NR CA-Polar maximum likelihood decoding by GRAND," in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2020, pp. 1–5.
- [38] 3GPP, "NR; Multiplexing and Channel Coding," Tech. Rep. TS 38.212, April 2020, Rel. 16.1.
- [39] J. Butler and T. Sasao, "High-speed hardware partition generation," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, pp. 1–17, 01 2014.
- [40] Thomas H. Cormen, *Algorithms for Sorting and Searching*, pp. 25–59, 2013.
- [41] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, New York, NY, USA, 1968, AFIPS '68 (Spring), p. 307–314, Association for Computing Machinery.