Abstract—Due to the low-latency and high-reliability requirements of 5G, low-complexity node-based successive cancellation list (SCL) decoding has received considerable attention for use in 5G communications systems. By identifying special constituent codes in the decoding tree and immediately decoding these, node-based SCL decoding provides a significant reduction in decoding latency compared to conventional SCL decoding. However, while there exists many types of nodes, the current node-based SCL decoders are limited by the lack of a more generalized node that can efficiently decode a larger number of different constituent codes to further reduce the decoding time. In this paper, we extend a recent generalized node, the sequence repetition (SR) node, to SCL decoding, and describe the first implementation of an SR-List decoder. By merging certain SR-List decoding operations and applying various optimizations for 5G New Radio (NR) polar codes, our optimized SR-List decoding algorithm increases the throughput by almost 2× compared to a similar state-of-the-art node-based SCL decoder. We also present our hardware implementation of the optimized SR-List decoding algorithm which supports all 5G NR polar codes. Synthesis results show that our SR-List decoder can achieve a 2.94 Gbps throughput and 6.70 Gbps/mm² area efficiency for \( L = 8 \).

Index Terms—Polar codes, successive cancellation decoding, list decoding, node-based SCL, sequence repetition (SR), hardware implementation, low-latency, 5G NR, wireless communications.

I. INTRODUCTION

POLAR codes, proposed in Arıkan’s seminal work [1], are the first class of error-correcting codes with an explicit construction that provably achieves channel capacity for binary-input discrete memoryless channels. Based on their outstanding error-correcting performance, polar codes were ratified as the standard code for the control channel of 5G enhanced mobile broadband (eMBB) [2]. While the polar code encoding schemes were released by 3GPP in 2018 [3], 3GPP did not specify a decoding scheme for polar codes. Although the original low-complexity successive cancellation (SC) decoding algorithm allows polar codes to achieve channel capacity at infinite code lengths, SC decoding has mediocre error-correcting performance for moderate code lengths and high decoding latency, which is insufficient for 5G scenarios.

To improve the error-correcting performance, Tal and Vardy proposed SC-List (SCL) decoding [4], which explores both hypotheses for each message bit and maintains a list of up to \( L \) candidate codewords in parallel during the decoding. The 5G standard also includes concatenated cyclic redundancy check (CRC) codes, which allow the SCL decoder to select the most reliable candidate codeword that satisfies the CRC, which further improves the SCL decoding performance [5]. However, SCL decoding with medium to large list sizes (e.g., \( L = 4 \) or \( L = 8 \)) suffers from high hardware complexity as an SCL decoder replicates the SC decoder structure \( L \) times. Nonetheless, owing to the hardware-friendly log-likelihood ratio (LLR) based SCL decoding algorithm [6], an SCL decoder with \( L = 8 \) provides a good trade-off between error-correcting performance and hardware complexity. It has therefore been chosen as the error-correcting performance baseline during the 5G standardization process [7].

Driven by the strict low-latency and high throughput requirements of modern communication systems, numerous SCL decoder hardware implementations have been presented in the literature [6], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. Most of these process more than one bit at a time to accelerate the decoding, since the conventional SCL decoder [6] suffers from a long latency due to the serial decoding of each bit. In [8], [9], [10], [11], [12], multi-bit SCL decoding is used in which a fixed number of bits are processed simultaneously, and all possible candidates for these bits are explored in parallel without exploiting a particular code structure. Alternatively, due to the recursive construction of polar codes, where every polar code of length \( N \) is formed...
from two constituent codes of length $N/2$, there exist constituent codes which have special bit patterns that allow for a more efficient direct decoding of all their bits in parallel. Rate-0 (R0) and Rate-1 (R1) nodes were initially proposed in [24] for simplified SC decoding with many other node types later identified, such as repetition (REP) and single parity-check (SPC) [25] nodes for Fast-SC decoding, TYPE-I to TYPE-V [26] nodes, and the generalized REP (G-REP) and generalized parity-check (G-PC) nodes [27]. These node-based techniques were later extended with list update techniques for node-based SCL decoding [14], [28], [29].

These special nodes often represent large constituent codes, especially for high- and low-rate polar codes, allowing a node-based SCL decoder to generally achieve much lower latency than multi-bit SCL decoders at the cost of making the decoding schedule more complex due to the need for detecting these nodes in the decoding tree. However, all existing node-based SCL decoders [13], [14], [15], [16], [17] suffer from either only supporting a few special nodes, which impairs their decoding latency, or they instantiate separate sub-decoders for each node type when supporting several of these, which complicates their hardware implementation. What is lacking is a more general node type implementation that supports most special node patterns with maximum hardware re-use and the ability to optimize the decoder throughput and area for a specific standard.

Recently, the sequence repetition (SR) node was proposed for SC decoding [30], [31]. The SR node is a generalized node that includes most aforementioned nodes as special cases. An SR-based decoder can thus support many node patterns in both low- and high-rate codes to reduce the decoding effort without limiting the range of supported codes. SR decoding can also potentially provide low-latency decoding as it is highly parallelizable, making it promising for SCL decoding in 5G. However, for SCL decoding, with a list of $L$ paths, the strategy from [31] of decoding with all the SR node candidates in parallel is highly computationally complex. Therefore, to make SR-List decoding more feasible in hardware, it is critical to develop a general low-complexity SR-List decoding algorithm and architecture. This architecture can then be customized to the specific codes employed by a standard to provide the best trade-off between area and decoding latency.

A. Contributions and Paper Outline

This work is an extension of our work in [32], in which the general SR-List decoding algorithm was initially proposed. However, due to the high implementation complexity when utilizing the large parallelism of SR-List decoding, it is difficult to fully realize the potential of the SR-List decoding algorithm in hardware. In this paper, we thus perform a joint optimization of the SR-List decoding algorithm with its hardware implementation. To better reach a low-latency implementation, we optimize the SR node structure by dividing it into a low-rate SR-I part combined with a rate-one SR-II part, and perform their maximum likelihood (ML) decoding and Wagner-based list decoding by merging or simplifying internal decoding steps. We also analyze the 5G NR polar codes to derive several constraints on the special node types, which helps us to significantly decrease the decoding latency and computational complexity. Our contributions on the algorithmic level comprise the following:

- We propose an optimized SR-List decoding algorithm which makes more operations execute in parallel and less complex to implement in hardware. Additionally, by investigating the node types encountered in the 5G NR polar codes, we can derive several constraints which further help to reduce the SR-List decoding complexity.
- For our optimized SR-List decoding, we also propose to combine it with two general latency reduction techniques which simplify the path update and node traversal, respectively. The worst-case latency is significantly reduced when compared to the state-of-the-art (SOA) node-based SCL [14] with negligible to no performance degradation.

On the architecture level, we design the SR-List decoder implementation, which is the first node-based SCL decoder that is fully compatible with and specifically optimized for all 5G NR polar codes. Our contributions in this area include:

- We propose flexible multi-stage decoding to combine multi-stage decoding [12] and node-based list decoding [14], which decreases the LLR memory and the LLR calculation latency.
- We provide an efficient architecture for SR-List decoding that efficiently re-uses internal modules to improve the hardware efficiency.
- Based on the rank-order sorter [33], we propose a new partial rank-order sorter which has a 45% area reduction compared to a full rank-order sorter and improves the maximum clock frequency.
- By exploring the SR-List decoder design space, we find the best algorithm/architecture configurations that satisfy different requirements for area and throughput. Synthesis results show that our decoder can outperform other similar polar decoders in throughput and area efficiency.

The remainder of this paper is organized as follows: Section II provides symbol definitions and a background on polar codes and decoding. Section III introduces the proposed SR-List decoding algorithm with both general optimizations and specifically for 5G NR polar codes. In Section IV, we present our SR-List decoder architecture which is compatible with all 5G NR polar codes. Section V discusses the implementation results and presents our design space exploration. Finally, Section VI concludes the paper.

II. PRELIMINARIES

Notation: In this paper, we use the following definitions. Boldface lowercase letters $u$ denote vectors, where $u[i]$ refers to the $i$-th element of $u$ and $u[i:j]$ is the sub-vector $(u[i], u[i+1], \ldots, u[j]), i \leq j$ and the null vector otherwise. Boldface uppercase letters $B$ represent matrices, with $B[i][j]$ denoting the element in the $i$-th row of the $j$-th column. Note that all indices start from 0. Blackboard letters $S$ denote sets with $|S|$ being the cardinality. The function $\text{HD}(x) := 1_{x \neq 0}$ defines the hard decision function. We adopt the following parameters from the 5G NR standard [3]: the length of the mother polar code is denoted as $N = 2^n$, $A$ is the number of message bits, $P$ the number of CRC bits, $K$ the number of messages bits with the CRC bits attached (i.e., information bits for polar codes), $E$ the length of the codeword after rate-matching, and $G$ the encoded block length. The $i$-th node at stage $s$ is denoted as $N_{s,i}$, the code length of the node as $N_{i}$ or $2^i$, and the number of information bits in the node is $K_{s,i}$. The frozen and information bit set indices are denoted as $\mathbb{P}$ and $\mathbb{A}$, respectively. The code-rate $R$ is $R = A/E$, and we refer to a code after rate-matching as an $(E, A)$ polar code. We use uplink (UL) and downlink (DL) codes from the Physical Uplink Control Channel (PUCCH), the Physical Uplink Shared Channel (PUSCH), and the Physical
Downlink Control Channel (PDCCH) from 5G NR. Further details on 5G NR are given in [3] and the 5G NR polar code construction is described in [34]. A summary of key symbols and function definitions is provided in Table A.1 of Appendix A.

A. Construction and Encoding

Given an input bit sequence \( \mathbf{u} \) of length \( N \) with \( A \) information bits, a polar encoder applies a linear transformation \( \mathbf{x} = \mathbf{uG} \) to get the codeword \( \mathbf{x} \). The generator matrix \( \mathbf{G} = \mathbf{F}^{s/n} \) is constructed from the kernel \( \mathbf{F} = [1 \quad 0] \). Based on the principle of channel polarization [1], the \( N \) bits in \( \mathbf{u} \) correspond to \( N \) individual bit channels with different reliabilities, where the \( K \) most reliable bit channels transmit information bits with CRC attached and the remaining \( N - K \) bit channels transmit frozen bits, typically set to a value 0. After encoding, rate-matching is applied to the codeword \( \mathbf{x} \) to get a codeword of length \( E \) for both UL and DL channels.

B. Successive Cancellation (SC) Decoding

At the receiver, a channel LLR vector of length \( N \) is obtained after rate-recovery and \( \mathbf{u} \) can be estimated by an SC decoder. SC decoding can be represented as the traversal of a binary tree with \( n + 1 \) stages (including the root node) and \( 2^{n-s} \) nodes at the \( s \)-th stage as illustrated in Fig. 1. Each node represents a constituent code of length \( 2^s \). For the \( i \)-th node at the \( s \)-th stage, \( \mathcal{N}_{s,i} \) with \( i \in [0, \ldots, 2^{s-1}-1] \), a length \( 2^s \) LLR vector \( \lambda_{s,i} \) is received and after traversing all the child nodes, the node returns a length \( 2^s \) partial sum (PSUM) vector \( \beta_{s,i} \) to its parent node. The \( \beta_{s,i} \) update equations are

\[
\begin{align*}
\lambda_{s,2i}[j] &= f(\lambda_{s+1,i}[j], \lambda_{s+1,i}[j + 2^s]), \\
\lambda_{s,2i+1}[j] &= g(\lambda_{s+1,i}[j], \lambda_{s+1,i}[j + 2^s], \beta_{s,2i}[j]),
\end{align*}
\]

where \( j \in [0, 2^s - 1] \) indexes the \( j \)-th value of an LLR vector. The \( f \) - and \( g \)-functions are defined as

\[
\begin{align*}
f(x,y) &= \text{sgn}(x)\text{sgn}(y) \min\{|x|, |y|\}, \\
g(x,y,z) &= (1 - 2z)x + y.
\end{align*}
\]

The PSUM vector, \( \beta_{s+1,i} \), is updated by

\[
\begin{align*}
\beta_{s+1,i}[j] &= \beta_{s,2i}[j] \oplus \beta_{s,2i+1}[j], \\
\beta_{s+1,i}[j + 2^s] &= \beta_{s,2i+1}[j].
\end{align*}
\]

C. Successive Cancellation List (SCL) Decoding

SC decoding only selects the locally optimal estimate for each information bit, and any wrong choice makes the entire estimated codeword incorrect. Contrary to SC decoding, SCL decoding maintains a list of up to \( L \) bit sequences (paths) by examining both hypotheses for each information bit [4]. Each path is forked into two new paths. The list is initialized with one path starting at the root. Once the number of generated paths exceeds the list size, only the \( L \) most reliable paths are kept by measuring their (approximate) path metrics (PMs) as

\[
\begin{align*}
\text{PM}_{l_{0,i}} &= \begin{cases} 
\text{PM}_{l_{0,i-1}} + |\lambda_{l_{0,i}}|, & \text{if } 0 < i \\
0, & \text{otherwise},
\end{cases}
\end{align*}
\]

where \( \lambda_{l_{0,i}} \) is the LLR of path \( l \) at leaf node \( \mathcal{N}_{l_{0,i}} \). HD(\( \lambda_{l_{0,i}} \)) is the hard decision based on \( \lambda_{l_{0,i}} \) and \( \beta_{0,i} \) is the \( i \)-th bit estimate for path \( l \).

To differentiate between paths considered at each information bit, we refer to the \( L \) paths before decoding an information bit as parent paths, the \( 2L \) paths created from considering both hypotheses at an information bit as candidate paths, and the \( L \) selected paths with the smallest PMs as the surviving paths. When the next information bit is encountered, the previously selected paths are used as parent paths. In (4), \( \text{PM}_{l_{0,i-1}} \) is the PM of a parent path and \( \text{PM}_{l_{0,i}} \) is the PM of a candidate path.

D. Node-Based Decoding

SC and SCL decoding suffer from long decoding latencies as the entire decoding tree is traversed. However, some special types of nodes in the decoding tree have frozen bit patterns that allow for directly calculating the PSUMs using special decoding algorithms. To label whether a leaf bit in the fanout tree of a node \( \mathcal{N}_{s,i} \) is frozen, we use a vector \( \mathbf{d}_{s,i} \), where \( \mathbf{d}_{s,i} \) indicates a frozen bit and \( \mathbf{d}_{s,i} = 1 \) indicates an information bit, respectively. In the following, we omit the node index \( i \) for brevity.

Using this notation, the initial four special nodes proposed in [24] and [25] are:

1) \( \text{R0} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, 0) \)
2) \( \text{R1} \) \( \mathbf{d}_{s} = (1, 1, \ldots, 1, 1) \)
3) \( \text{REP} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, 1) \)
4) \( \text{SPC} \) \( \mathbf{d}_{s} = (0, 1, \ldots, 1, 1) \)

Examples of these nodes are shown in Fig. 1. Note that some special nodes can be decomposed into other special nodes, e.g., the REP node \( \mathcal{N}_{S,0} \) contains both an R0 and an REP node.

The five additional nodes defined in [26] have the patterns:

5) \( \text{TYPE-I} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, 0, 0, 1, 1) \)
6) \( \text{TYPE-II} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, 0, 1, 1) \)
7) \( \text{TYPE-III} \) \( \mathbf{d}_{s} = (0, 0, 1, 1, \ldots, 1, 1) \)
8) \( \text{TYPE-IV} \) \( \mathbf{d}_{s} = (0, 0, 1, 0, \ldots, 1, 1, 1) \)
9) \( \text{TYPE-V} \) \( \mathbf{d}_{s} = (0, 0, 0, 1, 0, 1, 1, 1) \)

The top node in Fig. 1, \( \mathcal{N}_{S,0} \), is an example of a TYPE-V node, which is composed of an REP node and an SPC node.

The G-REP and G-PC nodes were proposed in [27] and are more general nodes for low- and high-rate codes, respectively. Their patterns are shown below, where \( N_p \) indicates the code length of a node at stage \( p \) with \( p < s \) and each “X” can either be a frozen or an information bit.

10) \( \text{G-REP} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, X, \ldots, X) \)
11) \( \text{G-PC} \) \( \mathbf{d}_{s} = (0, 0, \ldots, 0, 0, 0, \ldots, X) \)
Note that \( N_p \) can be zero, meaning there are no \( X_s \) for the G-REP node and no \( 0 \)s for the G-PC node. This new definition varies from [27], but it allows the G-REP node to include the R0 node and the G-PC node to include the R1 node.

The more general SR node [31], shown in Fig. 2, is an extended form of the G-REP node, with multiple groups of R0/REP nodes and a source node located at stage \( r \) that can be of any type. The pattern of an SR node is expressed as:

\[
12) \quad \text{SR} \quad d_s = (0, \ldots, 0, X, \ldots, 0, \ldots, 0, X, \ldots, X)_{N_s-1} \oplus (\eta[1])_{N_s+1} \oplus \eta_s_{N_s}.
\]

E. Sequence Repetition (SR) Node Decoding

The SR node is a generalized node that covers most of the aforementioned nodes as special cases. An SR-based decoder can thus support a wide range of frozen bit patterns. An SR node, as shown in Fig. 2, is any node \( N_s \) at stage \( s \), whose descendants are all either R0 or REP nodes, except for the rightmost node at stage \( r \), \( 0 \leq r \leq s \). Node \( N_r \) is called the source node, and is a generic node of any rate. The SR node structure is described by three parameters as SR(\( v, SNT, r \)) [31]. The length \((s-r)\) binary vector \( v \) describes the distribution of R0 and REP nodes towards \( N_r \). For \( 0 \leq t < s-r \), \( v[t] = 0 \) and \( v[t] = 1 \) indicates that the left node at stage \( (s-t-1) \) is an R0 node and an REP node, respectively. As such, the sum of \( v \) is the number of REP nodes, denoted as \( W_v \). The parameter \( SNT \) describes the source node type. As an example, the node \( N_{3,0} \) in Fig. 1 is an SR((1), SPC, 2) node. Additionally, we use a vector \( \eta \) to denote the last bit value (shown by an \( X \) in the special node description of the SR node) of each R0/REP node, that is, \( \eta[t] = 0 \) when \( v[t] = 0 \) and \( \eta[t] \in \{0, 1\} \) when \( v[t] = 1 \).

On a high-level, an SR node is decoded using different candidate LLR vectors, \( \lambda^k \), at the source node. Each candidate LLR vector is generated as shown in (7) from \( \lambda_s \), by repeatedly applying (2b) using repetition sequences, \( S^k \), with each sequence generated from a case of \( \eta \) [31]. Let \( S = \{S^0, S^1, \ldots, S^{W_v-1}\} \) denote the set of all possible repetition sequences for a given SR node. For each \( k \)-th case of \( \eta \), the sequence \( S^k \) is derived as

\[
S^k = (\eta[0, 0] \oplus (\eta[1, 0]) \oplus \cdots \oplus (\eta[s-r-1, 0]),
\]

with \( \oplus \) describing the operation

\[
(a[0], a[1], \ldots, a[i]) \oplus (b[0], b[1]) = (a[0] + b[0], a[0] + b[1], \ldots, a[i] + b[0], a[i] + b[1]).
\]

The SR node decoding process can then be divided into three main steps as:

1) The \( |S| \) LLR vectors to the source node are calculated as

\[
\lambda^k[j] = \sum_{m=0}^{2^s-1} (1 - 2^{S_k[m]}) \lambda_s[2^s m + j],
\]

where \( 0 \leq j < 2^s \) and \( 0 \leq k < |S| \).

2a) Given the \( |S| \) LLR vectors, the source node is decoded using \(|S|\) copies of the conventional SC decoder in parallel.

2b) Meanwhile, the ML candidate of the \(|S|\) candidates is identified as

\[
\hat{k} = \text{argmax}_{0 \leq k < |S|} \sum_{j=0}^{2^s-1} |\lambda^k[j]|.
\]

3) The SR node PSUM vector, \( \beta_s \), is computed by repeating \( \beta^k_r \), based on the corresponding \( \hat{k} \)-th repetition sequence

\[
\beta_s[2^s m + j] = \beta^k_r[j] \oplus S^k[m].
\]

where \( 0 \leq j < 2^s \) and \( 0 \leq m < 2^s-r \).

Node-based decoding techniques can be used in step 2a when the source node is a special node and step 2b can be skipped when there are only R0 nodes as left-descendants since \(|S| = 1 \).

Note that the LLRs for all the left R0/REP nodes under the SR node, \( \lambda^{s-1}, \lambda^{s-2}, \ldots, \lambda_s \), are never calculated and are grayed out in Fig. 2 along with the PSUMs.

III. PROPOSED SR-LIST DECODING ALGORITHMS

In this section, we propose several node-based list decoding algorithms for a low-latency SCL decoder using a single generalized node, the SR node. First, we present a general SR-List decoding algorithm to extend the SR node for list decoding with no error-correcting performance loss. Then, we propose an optimized low-complexity SR-List decoding algorithm based on the 5G NR node distribution and the merging or simplification of some SR-List decoding operations. We also provide a strategy for empirically reducing the number of path forks for SR-List decoding and we show how some 5G NR rate-matching schemes can reduce the decoding latency without any loss in error-correcting performance.

A. General SR-List Decoding

As described in Section II-E, the SR node provides a high degree of parallelization by considering all repetition sequences in parallel. While this can provide a reduction in decoding latency compared to other node types, all parent paths and their path forks have to be considered for all repetition sequences during list decoding. For an SR node with a source node of size \( N_s \) and information length \( K_r \), the total number of information bits is \( W_v + K_r \), and the \( L \) parent paths can each be forked \(|S|2^{W_v} (|S| = 2^{W_v}) \) times to account for all choices of information bits, which significantly increases the decoding and sorting complexity.

To alleviate this issue, we simplify the SR-List decoding process by dividing it into two parts, described in Section III-B and Section III-C, respectively. In the first part, for the R0 and REP nodes, we enumerate all information bit possibilities to...
generate \(|S|\) candidates from each of the \(L\) parent paths, calculate the PMs of all \(|S|/L\) candidate paths, and select \(L\) survivors. Then, in the second part, sequential node-based list decoding is performed for the source node using the previously selected \(L\) candidates. Note that in the hardware, we simplify the decoding for basic nodes such as R0, R1, REP, SPC, and TYPE-III as explained in Section IV-C.

**B. General SR-List Decoding Part 1: R0/REP Node Decoding**

In conventional node-based list decoding such as Fast-SCL decoding [14], SR node decoding starts by decoding each of the left-descendant R0 and REP nodes in order, and the PMs for the \(L\) parent paths at the source node are based on the input LLRs of all the individual R0 and REP nodes. However, in SR-List decoding, these LLRs are unknown to the decoder. Instead, we propose a four step process for efficiently calculating the PMs for \(L\) parent paths at the source nodes:

1) The LLR vectors at the source node, \(\lambda_r^{l,k}\), can be calculated for each path as

\[
\lambda_r^{l,k}[j] = \sum_{m=0}^{2^{r-r_l}-1} (1 - 2S^k[m]) \lambda_r^l[2^r m + j],
\]

where \(0 \leq l < L, 0 \leq k < |S|, \) and \(0 \leq j < 2^r\).

2) To calculate the corresponding PMs of the source node, we propose a method which does not directly decode the R0 and REP nodes. Let \(\beta_r^{l,k} = \text{HD}(\lambda_r^{l,k}),\) to distinguish from \(\beta_r^{l,k}\), which is the source node output after decoding it. Then, we calculate \(\beta_r^{l,k}\) as

\[
\beta_r^{l,k}[2^r m + j] = \lambda_r^{l,k}[j] \oplus S^k[m],
\]

where \(0 \leq j < 2^r\) and \(0 \leq m < 2^{r-r_l}\).

3) With the hard decision PSUMs at the source node and SR node, we then calculate PM\(_{l,k}\) at the source node for each sequence and for each parent path as

\[
\text{PM}_{l,k} = \sum_{j=0}^{2^{r-r_l}-1} [\beta_r^{l,k}[j] - \beta_r^{l,k}[j]]|\lambda_r^{l,k}[j]|,
\]

where PM\(_{l,k}\) indicates the PM of the \(l\)-th parent path at the SR node. Note that PM\(_{l,k}\) is identical with the results obtained from conventional Fast-SCL decoding where the LLRs and PSUMs from the R0 and REP nodes are calculated.

4) With PM\(_{l,k}\), we select the \(L\) best of the \(|S|L\) candidate paths and use these \(L\) paths for the source node list decoding.

Note that for each of the \(L\) paths that survived, the selected repetition sequence is recorded as \(k_l\) for \(0 \leq l < L\). In the next part of the general SR-List decoding, described in Section III-C, \(l\) refers to a candidate that survived in step 4 above, and not to the parent paths at the SR node input.

**C. General SR-List Decoding Part 2: Source Node Decoding**

With the \(L\) surviving paths as the parent paths for the source node, the decoder can start the source node list decoding. Unfortunately, the source node of the SR node can be any type of node, which significantly complicates the hardware implementation. However, if we constrain the source node to a G-PC node, whose first \(N_p\) bits are frozen bits as shown in Fig. 3, the source node can be viewed as a group of \(N_p\) SPC nodes. In Lemma 2 of [27], the list decoding strategy for the G-PC node is presented where \(N_p\) SPC node are decoded in parallel. However, such a parallel strategy explores many redundant candidates. Therefore, inspired by the SPC and TYPE-III list decoding processes in [14] and [29], respectively, we propose an efficient sequential list decoding algorithm for G-PC nodes.

1) For each path, split the LLRs and PSUMs into \(N_p\) groups and find each group’s minimum magnitude LLR index as

\[
e^l_q = \arg\min_{0 \leq q < N_p} |\lambda_r^l[N_p i + q]|, \quad (13)
\]

where \(0 \leq q < N_p\) is the group index.

2) For each path and group, compute the initial parity

\[
\gamma^l_q = \bigoplus_{i=0}^{N_r/N_p} \beta_r^l[N_p i + q], \quad (14)
\]

with \(\beta_r^l\) obtained at the source node after calculating the source node LLRs using (10) as described in Section III-B.

3) For each path, generate the ML candidate by computing the PSUMs, \(\beta_r^l\), which satisfy the even-parity constraint of each group as shown in (15) and calculate the corresponding candidate PM, \(\text{PM}_r^l\), as shown in (16):

\[
\tilde{\beta}_r^l[N_p i + q] = \begin{cases} \\
\beta_r^l[N_p i + q] + \gamma^l_q, & \text{if } i = e^l_q; \\
\beta_r^l[N_p i + q], & \text{otherwise.}
\end{cases}
\]

\[
\tilde{\text{PM}}_r^l = \text{PM}_r^l + \sum_{q=0}^{N_p-1} \gamma^l_q|\lambda_r^l[N_p e^l_q + q]|, \quad (16)
\]

with \(\text{PM}_r^l\) as the PM of a candidate allowed to survive after decoding the R0 and REP nodes, as described in Section III-B.

4) To reduce the exploration of redundant candidates, we modify the LLRs of each sub-group as follows. Let \(\zeta_r^l = \lambda_r^l\) except for \(K_r = N_r - N_p\) bits which are defined as

\[
\zeta_r^l[N_p i + q] = |\lambda_r^l[N_p i + q]| + (1 - 2\gamma_q^l)|\lambda_r^l[N_p e^l_q + q]|, \quad (17)
\]

where \(0 \leq i < N_r/N_p\) and \(i \neq e^l_q\).
5) We then identify the bit-flipping candidates for further path forking by sorting \( |a^1| \) in ascending order with sorted index \( j^l \), i.e., \( j^l \) is the index of the \( j \)-th minimum in \( |a^1| \). Note that the LLRs found in (13) are excluded from the sorting as these are used to satisfy the parity constraint of all candidates.

6) Then, we serially fork the paths from sorted index \( 0^l \) to \( (K_r - 1)^l \) for the \( l \) paths, that is, the number of path fork operations is \( K_r \). The PMs for the generated codewords are

\[
\hat{\text{PM}}_{r,j}^{l} = \begin{cases} 
\text{PM}_{r,j-1}^{l}, & \text{if } \gamma_r^{l}[j] = \tilde{\beta}_r^{l}[j], \\
\text{PM}_{r,j-1}^{l} + \Delta_r^{l,j}, & \text{otherwise},
\end{cases}
\]

where \( \text{PM}_{r,j-1}^{l} = \text{PM}_{r}^{l} \) and \( \Delta_r^{l,j} \) is given as

\[
\Delta_r^{l,j} = |\lambda_r^{l}[j]| + (1 - 2q_r^{l})|N_p\epsilon^{l}_{q} + q_j^{l}|,
\]

with \( q_j^{l} = j \mod N_p \), i.e., the group index of the LLR with sorting index \( j^l \). Note that if \( \text{PM}_{r,j-1}^{l} \) is incremented with \( \Delta_r^{l,j} \), the corresponding \( \gamma_r^{l} \) should be flipped. The \( L \) paths with the lowest PMs are selected for the next path forking and this process is repeated \( \min(L - 1, K_r) \)-times.

7) Finally, the SR node PMUs can be obtained as

\[
\beta_s^{l}[2^s m + j] = \beta_r^{l}[j] \oplus \hat{\Delta}_s^{l,j}[m],
\]

where \( 0 \leq j < 2^s \), \( 0 \leq m < 2^{s-r} \), and \( k_l \) is the \( k \)-th sequence used for path \( l \) found in Section III-B.

Modifying the LLRs in (17) results in a set of LLRs where the sub-groups with a parity of 1 (i.e., \( \gamma_q^{l} = 1 \) in (14)) are penalized and all LLR magnitudes in that group are reduced, making it more likely that those indices will be amongst the first to be explored, thus reducing the number of redundant candidates compared to [27]. Note that if the G-PC node is an R1 node, we skip steps 1-4 and directly perform the path forking. If the G-PC node is an SP node, the LLRs in (17) are unnecessary and the process simplifies to SPC list decoding.

D. Optimized SR-List Decoding

For the overall SR-List decoding, the source node list decoding waits until the list decoding of the R0/REP groups is complete. To reduce the time complexity of this process, we propose an optimized SR-List decoding algorithm where we redefine the SR node representation and merge some SR-List decoding steps. Moreover, by taking the specific 5G NR node distribution into account, we apply some constraints on the special nodes to significantly simplify the implementation.

1) Optimized SR Node Representation: As described in Section III-C, the G-PC node is a good generalized form of the source node in the SR node, whose frozen bits are distributed in the front and thus do not cause path forking. If we only use the G-PC node as the source node type, the decoding can be simplified by considering the source node parity check together with the list decoding of the R0 and REP segments in the SR node. With this approach, the SR node is divided into two parts.

\[\text{PM}_{r,k}^{l} = \text{PM}_{r}^{l} + \sum_{j=0}^{2^s-1} |\beta_s^{l}[j] - \hat{\beta}_s^{l}[j]| |\lambda_r^{l}[j]| + \sum_{q=0}^{N_p-1} |\lambda_r^{l}[k_q^{l}] N_p + q|,\]

where \( \Delta_r^{l,k} \) denotes the PM penalty from the parity constraints of the source node (G-PC node) for path \( l \) with repetition sequence \( k \). Note that all terms in (21) can be calculated in parallel (after 10). Moreover, the best \( L \) paths that survive from the \( |S|/L \) paths of (21) slightly improve the error-correcting compared to (12) (without considering the parity), due to the more global (direct) selection of the \( L \) best paths from the \( |S|/L \) candidate paths.

The full SR-List decoding algorithm is given in Algorithm 1. Note that in Algorithm 1 we use (23) which we introduce in Section III-D as a simplification of (21). In contrast to the SR-I part, the SR-II list decoding is performed by sequentially selecting \( L \) paths from \( 2L \) generated candidate paths. After list decoding the source node, the PSUM vectors to the SR parent node are calculated according to (20). In the remaining sections of this paper, we only use this optimized SR node representation.

2) SR and G-PC node distribution in 5G NR: For 5G NR polar codes, we can potentially find some constraints on the SR and G-PC node parameters which simplify the hardware with little to no increase in worst-case decoding latency. In particular, we note that more complex, potentially unsupported SR or G-PC

---

The text above is a natural representation of the content in the image. It has been formatted to ensure readability and coherence, with the mathematical expressions and algorithms presented in a clear and understandable manner. The content has been condensed and simplified where appropriate, while retaining the core information and structure of the original document.
nodes can still be decomposed into other nodes, which require slightly more time to decode, but where the reduced decoding time does not justify the additional hardware resources. To find constraints which reduce the overall complexity, we check all valid 5G NR polar code configurations, i.e., combinations of $A$ and $G$. For PDCCH we use $A$ ∈ {12, 140} and $G$ ∈ {36, 8192}. For PUCCH/PUSCH we use $A$ ∈ {12, 1706} and all the supported $G$ as shown below (taken from (1) in [35])

$$G \in \begin{cases} [A + 9, 8192] & \text{if } A \in [12, 19] \\ [A + 11, 8192] & \text{if } A \in [20, 359] \\ [A + 11, 16385] & \text{if } A \in [360, 1012] \\ [2 \cdot A/2] + 22, 16385 & \text{if } A \in [1013, 1706]. \end{cases}$$  (22)

In Table I, we show the percentages of the different SR node sizes and G-PC node types depending on the number of repetition sequences $|S|$ in the SR node and the number of frozen bits $N_{p}$ in the G-PC node. For SR nodes in PDCCH, the number of valid 5G NR polar code configurations, i.e., combinations of $A$ and $G$. For PDCCH/PUSCH, the number of SR nodes with $|S| \leq 4$ and $|S| \leq 8$ accounts for 92.16% and 98.12%, respectively. For PDCCH and PUSCH, the number of 5G NR polar code configurations, i.e., combinations of $A$ and $G$. For PDCCH/PUSCH, the number of SR nodes with $|S| \leq 4$ and $|S| \leq 8$ is 95.7% and 99.03%, respectively. Additionally, we note that SR nodes with $|S| > 32$ never occur. As described in Section III-A, the computational complexity of decoding the SR node part with the R0 and REP nodes is linearly related to $|S| L$. Therefore, as SR nodes with large $|S|$ rarely occur, we only consider $|S|_{\text{max}} \in \{2, 4, 8\}$ as interesting options for further evaluation in the hardware implementation.

For the G-PC node in PDCCH and PUCCH/PUSCH, we only observe $N_{p} \in \{0, 1, 2\}$, where $N_{p} = 0$ corresponds to having a R1 node, $N_{p} = 1$ is an SPC node, and $N_{p} = 2$ is a TYPE-III node. G-PC nodes with $N_{p} > 2$ never appear. Therefore, even though the G-PC node is a generalized node, only a limited number of cases occur in valid 5G NR codes and we therefore restrict our decoder to $N_{p_{\text{max}}} = 2$.

Consequently, in the following, all simulated SR-List decoding results are based on the optimized SR-List decoding with the constraints $|S|_{\text{max}} = 8$ and $N_{p_{\text{max}}} = 2$. We can then simplify (21) into three explicit formulas for the three supported G-PC cases, as shown in (23) and (24)

$$\bar{\mathbf{P}}_{r}^{l,k} = \mathbf{P}_{r}^{l} + \sum_{j=0}^{2^{N_{p}}-1} |\gamma_{s}^{l}[j] - \beta_{s}^{l,k}[j]|^{\lambda_{s}^{l}[j]} + \Delta_{s}^{l,k}, \quad (23)$$

where $\Delta_{s}^{l,k}$ from (23) is expanded as

$$\Delta_{s}^{l,k} = \begin{cases} 0 & \text{if } N_{p} = 0 \\ \gamma_{s}^{l,k}[\lambda_{s}^{l,k}[\epsilon_{s}^{l,k}]] & \text{if } N_{p} = 1 \\ \gamma_{s}^{l,k}[\lambda_{s}^{l,k}[\epsilon_{s}^{l,k}]] + \gamma_{0}^{l,k}[\lambda_{0}^{l,k}[\epsilon_{0}^{l,k}]] & \text{if } N_{p} = 2 \end{cases}$$  (24)

E. General Latency Optimizations

In this section, we describe two additional decoding optimizations: rate-matching adaptation and empirical path forking, which can be used in any node-based SCL decoder.

1) Rate-Matching Adaptation: In 5G NR, rate-matching is applied to a codeword using either puncturing, shortening, or bit-repetition [35]. For puncturing, the $N - E$ first codeword bits are removed and the LLRs after rate-recovery have value zero. With shortening, the $N - E$ last codeword bits are removed, and the LLRs are set to $\infty$ after rate-recovery. In our decoder, we use puncturing and shortening bits along with the first information bit index to speed up the decoding.

In Fig. 4 we show an example of using the first information bit and the shortening bits in the decoding. As the PSUMs at frozen bits are always 0, all R0 nodes have an output PSUM vector of all 0s and the first R0 nodes can thus be skipped. Decoding can start directly at the green encircled node. For the shortening bits, we treat a node with only information and shortening bits as a R1 node, like the red encircled node in Fig. 4. The decoder can immediately decode this node instead of first traversing the corresponding subtree. We note that this method does not affect the error-correcting performance and it is cheap to implement in hardware. Note that if puncturing is used instead of shortening, the puncturing bits can be combined together with the frozen bits at the front to potentially skip even more R0 nodes.

2) Empirical Path FORking: All aforementioned algorithms for SR-List decoding work without any error-correcting performance loss. However, some of the min$(L - 1, K_{s})$ path forks are often redundant for high-rate nodes (i.e., R1, SPC, and TYPE-III nodes), where the bit-channels have high reliability. In practice, the number of path forks can be empirically reduced to a constant value $T_{\text{SNT}}$ for a general node, as first proposed

\begin{table}[H]
\centering
\caption{SR and G-PC Node Distribution in 5G NR}
\begin{tabular}{|c|c|c|}
\hline
Name & PDCCH [%] & PUCCH/PUSCH [%] \\
\hline
$|S|$ & 1 & 62.78 \hspace{1cm} 68.85 \\
& 2 & 19.07 \hspace{1cm} 19.12 \\
& 4 & 10.31 \hspace{1cm} 7.73 \\
& 8 & 5.96 \hspace{1cm} 3.33 \\
& 16 & 1.88 \hspace{1cm} 0.93 \\
& 32 & $4.31 \times 10^{-4}$ \hspace{1cm} $4.28 \times 10^{-2}$ \\
\hline
$N_{p}$ & 0 & 8.47 \hspace{1cm} 17.42 \\
& 1 & 83.91 \hspace{1cm} 76.31 \\
& 2 & 7.62 \hspace{1cm} 6.27 \\
\hline
\end{tabular}
\end{table}

Fig. 4. SC decoder tree with two frozen bits, three information bits, and three shortening bits. An arbitrary LLR value is indicated as $?, a \text{PSUM value} X$ which can be either 0 or 1, and $\infty$ is for an LLR value for the shortening. The green and red encircled nodes are the first and last nodes, respectively.
in [14], which only negligibly degrades the error-correcting performance and increases the throughput by reducing the latency from path forking. With this optimization, the number of path forks changes as follows

$$\min(L - 1, K_s) \rightarrow \min(T_{SNT}, K_s), \quad (25)$$

where $T_{SNT}$ in our decoder is either $T_{R1}$, $T_{SPC}$, or $T_{TYPE-III}$, depending on the node type.

Using different values of $T_{SNT}$ as an upper limit on the number of path forks provides further flexibility to trade-off error-correcting performance for the speed of SR-List decoding. Numerical results show that for $L = 4$, we can select $T_{R1} = 1$, $T_{SPC} = 2$, and $T_{TYPE-III} = 2$. For $L = 8$ we can select $T_{R1} = 2$, $T_{SPC} = 3$, and $T_{TYPE-III} = 3$. These values result in a negligible reduction in FER, as illustrated in Section III-F, and provide a large latency reduction. As each path fork generally requires one clock cycle (CC), this technique helps to significantly reduce the decoding latency.

### F. Error-Correcting Performance of Proposed Algorithms

In this section, the error-correcting performance of the proposed algorithms are provided by simulating 5G NR polar codes. For UL channels (PUCCH/PUCSH), we generally only use $E = 1024$ with different $R$. For DL channels (PDCCH), we consider typical values of $E \in \{108, 216, 432, 864, 1728\}$ and $A \in [12, 140]$, as described in [35]. Note that most $E \in \{864, 1728\}$ bit patterns overlap with $E = 432$ due to bit-repetition and we thus only use $E \in \{108, 216, 432\}$ for DL. For both DL and UL channels we limit $L_{max}$ to 8 as this is what has been selected as the baseline for the maximum list size by 3GPP during the standardization process [7]. All data is modulated by binary phase-shift keying and sent over an additive white Gaussian noise channel.

Fig. 5 illustrates the FER performance for Fast-SCL decoding [14], for our SR-List decoding algorithm, and for the approximate SR-List decoding with different $T_{SNT}$ for UL and DL codes with $L \in \{4, 8\}$. We note that due to the selection of the globally optimal $L$ paths from the $[S/L]$ paths, our SR-List decoding algorithm show a slight performance improvement compared with Fast-SCL [14]. For the empirical path forking, the results in Fig. 5 show that for $L = 4$, the constants can be selected as $T_{R1} = 1$, $T_{SPC} = 2$, and $T_{TYPE-III} = 2$ with minimal error-correcting performance degradation. For $L = 8$ we can select $T_{R1} = 2$, $T_{SPC} = 3$, and $T_{TYPE-III} = 3$.

### IV. NODE-BASED POLAR DECODER ARCHITECTURE

In this section, we describe the implementation of our node-based SCL decoder, which is compatible with all 5G NR polar codes and with the algorithm optimizations described above. More specifically, we give a detailed description of each component, which, at a high-level, comprises three types of functions: memory, decoder cores, and control logic. High-level illustrations of the architecture and timing schedule are shown in Fig. 6 and Fig. 7, respectively.

For the memory, our decoder has channel LLR and internal LLR memories, Us memory, PSUM memory, PM memory, a pointer memory, and an instruction FIFO. The decoder cores are used to process the LLRs to get the codeword. They comprise the SC unit (SCU), the node-processing unit (NPU), and the PSUM unit (PSU). As illustrated in Fig. 7, the SCU first calculates the internal LLRs until a special node is encountered. Then, the NPU decodes the special node using the LLRs calculated by the SCU. Note that the NPU is the generalized node implementation that supports list decoding for all SR nodes with the constraints from Section III-D2. Finally, the PSU is used for the PSUM calculations and for copying the PSUMs as required by the list decoding. In parallel to the PSU, the decoder updates the message bits in the Us memory and the corresponding indices in the pointer memory. Afterwards, the SCU runs again until the next node is encountered and the procedures repeat until the final node is decoded. This process is orchestrated by the controller, which reads from a list of instructions and coordinates the components of the decoder.

#### A. Decoder Memories

The largest memories are the channel LLR memory and the internal LLR memory, shown on the left side of Fig. 6. These store the received channel LLRs and the internal LLRs which are generated in the decoder. Note that while the number of
bits used for channel and internal LLR values is the same, the internal LLR memory is instantiated $L$ times to store the internal LLRs for the $L$ paths. The internal LLR memory of each path is divided into two parts: an upper part for stages closer to the root where almost no special nodes can be identified and a lower part where special nodes are frequent. The upper part of the memory can be pruned to only store LLRs for some stages as described in Section IV-B, while the lower part provides storage for all remaining stages to avoid limiting the optimal use of the NPU. The internal LLR memory makes use of another memory, the pointer memory, shown at the bottom of Fig. 6. This pointer memory tracks the physical location of the internal LLRs for the individual paths in the $L$ LLR memories based on the path forking history [36].

The decoder also contains various smaller memories. The PSUM memory, shown in the PSU in the upper right corner of Fig. 6, holds the PSUM values generated after running the NPU, and is further explained in Section IV-E. Note that to support the optimizations described in Section III-E the PSUM memory is always cleared to 0 before decoding a new codeword. The Us memory, shown in the lower left corner of Fig. 6, stores the decoded message bits which are generated by re-encoding the PSUMs. At the end of the decoding process, once the CRC has been calculated for all final paths, the decoded message can be read out from the Us memory. The PM memory, shown above the pointer memory in Fig. 6, holds the PMs used during the decoding. Finally, the instruction FIFO holds the instructions which are fed to the controller.

B. SCU With Flexible Multi-Stage Decoding

The SCU, shown at the top of Fig. 6, calculates the internal LLRs using processing-elements (PEs) that implement the $f$- and $g$-functions in (2a) and (2b), respectively. The SCU calculates the LLRs for multiple stages of the decoding tree in a single cycle by connecting PEs to process multiple stages in a longer combinatorial path.

With multi-stage decoding, our decoder can significantly reduce the internal LLR memory size since intermediate LLRs are re-calculated combinatorially (i.e., without the need for dedicated CCs) rather than stored. This is especially important in list decoding, as the internal LLR memory, which is the largest of the memories, is instantiated $L$ times. The concept of multi-stage decoding with memory reduction is shown in Fig. 8. For an SCU with 3 stages and 64 PEs in the first stage, $\lambda_{7,0}$ can be read to directly calculate $\lambda_{4,0}$ in one CC. However, if the SCU has less than 64 PEs, a cycle penalty compared to single stage decoding with no memory optimization is incurred. This penalty arises since it takes multiple CCs to read $\lambda_{7,0}$ and those multiple CCs are required repeatedly for all LLRs in Fig. 8 that are directly calculated from $\lambda_{7,0}$. The SCU also calculates the LLRs $\lambda_{5,0}$ and $\lambda_{6,0}$ but these are not saved as the internal LLR memory in
is either an R1. For the low-rate nodes, i.e., R0 and REP, an adder tree
may vary, the full adder tree is not always used, and the MUX
s
r,
which represents the
for part 1 of the SR-
allows for finding larger nodes
N
max
= 4
L
PMs in the
log
N
s
s
= r
and
g
L
N
λ
= 4
N
N
N
N
λ
could be decoded directly in only
with decoding tree and
the NPU can decode most SR nodes encountered in practical
(BNU) for the SR-II part. As a generalized node implementation,
sequence unit (RSU) for the SR-I part and the basic node unit
the SR node structure described in Section III-D: the repetition
C. Repetition Sequence Unit (RSU) of the NPU

In our polar decoder, the multi-stage decoding needs to be
refined further since special nodes may be encountered before
the last stage calculated inside the SCU. In the example in
Fig. 8, a straightforward 3-stage SCU would calculate the LLRs
up to \( \mathcal{N}_{4,0} \), \( \mathcal{N}_{4,1} \), \( \mathcal{N}_{4,2} \), and \( \mathcal{N}_{4,3} \), which are SR, SPC, and
two R1 nodes, respectively, with a cost of 27 CCs. However,
the larger R1 node \( \mathcal{N}_{5,1} \) could be decoded directly in only
21 CCs (28.6% less CCs) if the intermediate LLRs \( \lambda_{5,1} \) were
available. Therefore, we employ flexible multi-stage decoding,
which stops at the first special node that the NPU can decode,
even in the intermediate stages of the SCU, and only saves
the LLRs for this node (in the example in Fig. 8 \( \lambda_{5,1} \)).
The flexibility of returning the LLRs from intermediate stages in
the SCU increases the number of opportunities for identifying
special nodes and results in an overall cycle count reduction by
using the NPU earlier. However, flexible multi-stage decoding
requires that the internal LLR memory can save the LLRs for all
stages where the NPU can decode nodes. As most of the LLR
memory is needed for the higher stages of the decoding tree,
where node-based decoding is not used, the memory reduction
is still significant, even though these lower stages can no longer
be pruned.

While multi-stage decoding has been used in some previous
works such as [12] and [37], our work improves on these in
several ways. In [12], multi-bit decoding is used at a fixed stage
instead of node-based decoding which our decoder can do whenever
a special node is encountered. In [37], multi-stage decoding is
only used at lower stages where node-based decoding is often more efficient and their decoder can do at most two combined
f-operations and not multiple g-operations. This limitation is
not present in our decoder.

C. Repetition Sequence Unit (RSU) of the NPU

The NPU, shown at the bottom of Fig. 6, performs the
node-based list decoding and comprises two parts based on
the SR node structure described in Section III-D: the repetition
sequence unit (RSU) for the SR-I part and the basic node unit
(BNU) for the SR-II part. As a generalized node implementation,
the NPU can decode most SR nodes encountered in practical
5G scenarios with maximum hardware re-use to enhance the
hardware efficiency. Note that for SR nodes where \( s = r \), i.e.,
the SR node and the source node are the same, the SR node is
decoded as a standard R0, REP, or G-PC node and the BNU is
directly activated, skipping the RSU. Similar to the special node
constraints described in Section III-D, we fix the maximum node
size that the NPU can decode as \( N_{\text{max}} \), which represents the
trade-off between hardware complexity and decoding latency.
Specifically, increasing \( N_{\text{max}} \) allows for finding larger nodes
which helps reduce the decoding time. However, this also leads
to a significantly more complex NPU implementation which can
potentially be slower overall due to an increase in the length of the
critical path.

The RSU architecture is shown in Fig. 9. To implement (10),
the LLR values are first put in bit-reversed order to route the
LLRs correctly for different SR node sizes. Then the repetition
sequences are XOR-ed with the LLR sign-bits and the resulting
LLRs are added together to generate the source node LLRs. For
adding the LLRs, a single adder tree of depth \( \log_2 (|S|_{\text{max}}) \)
where all internal results and the final result are routed to a MUX. As \( s \) and \( r \) vary, the full adder tree is not always used,
and the MUX selects which internal results from the adder tree are used based on \( s \) and \( r \).

After calculating the source node LLRs, the penalty unit
updates the PMs according to (12) with the PSUMs from (11),
the comp unit finds the indices of the minimum magnitude LLRs
per (13), and the G-PC group parities are calculated based on (14)
in the parity unit. These three steps are all done in parallel. As
the G-PC node with the constraint \( N_{\text{max}} = 2 \) is either an R1 node,
an SPC node, or a TYPE-III node, we re-order the source node
LLR vectors into even- and odd-indexed. The minimum of
each group and the global minimum of the node can be obtained
simultaneously by using compare-and-select (CAS) trees in the
comp unit.

The RSU operations are done in two CCs, with the previously
described procedures implemented in the sequence extension unit
(SEU) in the first CC. In the second CC, we calculate the ML PMs according to (21) and sort the \( |S| \) PMs in the
sequence sorter unit (SSU) to get the \( L \) smallest ones. The sorter
architecture is further explained in Section IV-F.

D. Basic Node Unit (BNU) of the NPU

The BNU, shown in Fig. 10, is used for directly decoding
R0 nodes, REP nodes, and G-PC nodes when \( s = r \) or when
these are decoded in part SR-II of the SR-List decoding when
\( s > r \). For the low-rate nodes, i.e., R0 and REP, an adder tree

Fig. 8. Flexible multi-stage decoding for \( L = 4 \) with decoding tree and
operation schedule for a 3 stage SCU with 64 PEs in the first stage. LLRs
in red are not saved to memory. Node latencies are obtained from Table II with
1 CC added for updating the PSUM memory using the PSU.

Fig. 9. Repetition sequence unit (RSU) for \( |S|_{\text{max}} = 4 \) for part 1 of the SR-
List decoding in Section III-B.
calculates the PM increment for the all-zero or all-one PSUM vector from the R0 and REP nodes. For the high rate nodes, i.e., the G-PC nodes, we instantiate a CAS tree, a PM fork unit, and a sorter for path forking. Similar to [38], the full sorter for the modified LLRs from (17) can be substituted with identifying the minimum magnitude LLR value for the \( \lambda \)-th list in the \( \lambda \)-th CC. That is, the CAS tree only has to identify the \( j \)-th minimum magnitude LLR value for the \( l \)-th list in the \( j \)-th CC. Note that for SPC and TYPE-III nodes, when \( s = r \), the BNU requires an extra CC to perform Wagner decoding to ensure that the G-PC nodes satisfy the even-parity constraint, otherwise, the BNU directly executes the path forking as done for R1 nodes. To improve the maximum clock frequency of the BNU, we insert registers after the adder and the CAS trees, which increases the latency of the BNU by one CC for path forking the G-PC nodes. Furthermore, we subtract the minimum PM value from the final PMs as in [12] at the end of the BNU to decrease the latency of the BNU by one CC for path forking the G-PC nodes.

The NPU latency for the different node types is summarized in Table II. Note that when decoding the SR nodes, since the indices of the minimum magnitude LLRs in (13) and the ML PMs of (21) are calculated in the RSU, one CC for SR(R1) and one Wagner decoding CC for SR(SPC) and SR(TYPE-III) can be removed. A more detailed decoding latency analysis is given in Section V-C.

**TABLE II**

| Node   | Clock cycles (CCs) |
|--------|---------------------|
| R0     | 1                   |
| REP    | 2                   |
| R1     | \( \min(L - 1, N_x) + 1 \) |
| SPC    | \( 1 + \min(L - 1, N_x) + 1 \) |
| TYPE-III | \( 1 + \min(L - 1, N_x) + 1 \) |
| SR(R1) | \( 2 + \min(L - 1, K_\gamma) + 1 \) |
| SR(SPC) | \( 2 + \min(L - 1, K_\gamma) + 1 \) |
| SR(TYPE-III) | \( 2 + \min(L - 1, K_\gamma) + 1 \) |

1 is for the inserted registers in Fig. 10 for the BNU.
2 is the fixed latency of the two parts of the RSU module in Fig. 9.

The critical path of an SCL decoder is generally through the PM sorting unit [6], and several works have thus focused on optimizing the PM sorting, either using pruned sorting [33], [39], [40] or partial sorting [11], [18], [41], with a comparison of various implementations in [33]. Most of these current sorters focus on the 2L \( \rightarrow \) L sorting required for the result in (4), including the SOA pruned rank-order sorter in [33], which prunes the sorter by using the property that the result of comparing certain PMs is known ahead of time. However, this property of the PMs depends on how the path forking is done in hardware and requires additional routing and control. As the SR-I part requires sorting \( \lfloor S \rfloor \) PMs from (21), the sorting size for the SR-List decoder is much larger compared to previous works, and all of the \( \lfloor S \rfloor \) PMs have different PM penalties so there is no explicit order known ahead of time. Therefore, in our decoder, we consider more general cases and assume no special properties on the input PMs.

To avoid a large complexity by fully sorting all input PMs, we simplify the sorter by allowing the output PMs to be partially sorted, as SCL decoding only requires the \( L \) smallest PMs without these being sorted. Herein, a full sorter can be replaced by two small full sorters and one layer of comparators, which results in our proposed partial rank-order sorter almost halving the number of comparators compared to a full sorter.

Let \( X \rightarrow Y \) denote a sorter with \( X \) inputs and \( Y \) outputs. Our partial rank-order sorter, shown in Fig. 11 for 8-to-4, is composed of two \( \frac{X}{2} \) to \( Y \) full rank-order sorters which return at least \( Y \) fully sorted results in ascending and descending order.
respective number of SCUs. The maximum achievable operating frequency for each design is limited to the benefit of additional SCU stages given that the largest code length for the 5G scenarios is only $N = 1024$, there is a 45% smaller area than the full sorter for 64-to-8.

**G. Controller**

Finally, the controller, shown in the lower right corner of Fig. 6, is responsible for orchestrating the decoding process, by activating the different main components of the decoder, i.e., the SCU, NPU, PSU, and LLR memories. The controller is given a list of instructions which can be generated offline or using a hardware scheme similar to the one in [42]. These instructions provide the controller with information about the current operation, e.g., the size and type of a special node for the NPU to decode. When the controller activates a component like the NPU, the NPU sub-controller uses the high-level instructions for the actual node decoding to control the RSU, the BNU, and the path forking process. Note that once the $g$-function has been executed at the root node of the polar decoding tree, the channel LLR memory can be overwritten with the channel LLRs of a new codeword while the controller continues to decode the current codeword to reduce the time between decoding multiple codewords.

**V. IMPLEMENTATION RESULTS**

In this section, we present the synthesis results for our SR-List decoder for 5G NR polar codes. All synthesis results are based on a STM 28 nm FD-SOI technology in the slow-slow corner and timing constraints that are not achievable to better determine the maximum achievable operating frequency for each design. To simplify the design space, we fix the default number of SCU stages (SCU) to 2, the number of PEs at the first stage of the decoder for 5G NR polar codes. All synthesis results are based on 45% smaller area than the full sorter for 64-to-8.

| $|S|_{\text{max}}$ | $L = 2$ | $L = 4$ | $L = 8$ |
|------------------|-------|-------|-------|
| Ccs               | 155   | 140   | 155   |
| Latency (μs)     | 0.109 | 0.101 | 0.112 |
| Area (mm²)       | 0.099 | 0.109 | 0.115 |
| Freq. (MHz)      | 1418  | 1385  | 1312  |
| T/P (Gbps)       | 4.685 | 5.065 | 4.587 |
| Area Eff. (Gbps/mm²) | 46.68 | 33.99 | 29.31 |

While our decoder is compatible with all DL codes, the results shown in this table are for the worst-case latency for all DL codes.
A. Quantized FER Performance

In Fig. 12, we show the FER performance of the decoder using floating-point and fixed-point (sign-magnitude) for UL and DL codes with \( L \in \{4, 8\} \). Let \( Q_{q_i q_f} \) denote a fixed-point number with one sign-bit, \( q_i = q_f = 1 \) integer bits, and \( q_f \) fractional bits. We use \( Q_{6.2} \) for the LLRs and \( Q_{7.0} \) for the PMs as the quantized FER results in Fig. 12. Numerical results show that this representation has a negligible FER performance loss compared to floating point.

B. Design-Space Exploration

While we already fixed some parts of the decoder architecture by setting \#SCU = 2, \#PE = 64, \( N_{\text{max}} = 32 \) and restricting the G-PC node to \( N_p \in \{0, 1, 2\} \) in Section III-D, we still have to determine \( |S|_{\text{max}} \). To determine an optimal \( |S|_{\text{max}} \), we perform a design-space exploration with \( |S|_{\text{max}} \in \{2, 4, 8\} \) as defined in Section III-D for DL polar codes with \( L \in \{2, 4, 8\} \). The corresponding results are given in Table IV. Note that our decoder is compatible with all PDCCH codes and only the worst-case hardware latency and throughput results amongst all PDCCH codes are shown in Table IV.

For the area in Table IV, we note that increasing \( |S|_{\text{max}} \) from 4 to 8 results in an increase in the area of 23.9%, 28.0%, and 28.7% for list sizes 2, 4, and 8, respectively. However, while increasing \( |S|_{\text{max}} \) reduces the number of decoding cycles, once the reduction in the maximum clock frequency has been accounted for, the actual decoding latency in \( \mu s \) increases slightly when \( |S|_{\text{max}} > 4 \) for \( L = 2 \) and \( L = 4 \) and there is only a small 4% reduction in the latency for \( L = 8 \). While \( |S|_{\text{max}} = 2 \) has the best area efficiency for \( L \in \{2, 8\} \), \( |S|_{\text{max}} = 4 \) always has a higher throughput and the difference between \( |S|_{\text{max}} = 2 \) and \( |S|_{\text{max}} = 4 \) in terms of area efficiency is minimal. Therefore, for our SR-List decoder, we set \( |S|_{\text{max}} = 4 \) as this represents the best trade-off when decoding time has a slightly higher priority than area.

C. Decoding Latency Analysis

To evaluate the decoding latency reduction from the various algorithms presented in Section III, we calculate the number of cycles to decode each DL polar code with \( E = 432 \) and different \( A \). The corresponding worst-case latency for PDCCH with \( L = 8 \) is shown in Fig. 13.

Compared with the baseline Fast-SCL decoder [14], the proposed flexible multi-stage decoding reduces the worst-case latency from 334 CCs to 311 CCs. Using our proposed optimized SR-List decoding algorithm further reduces the worst-case latency to 240 CCs, showing that the SR node provides a significant reduction in decoding latency. Moreover, using empirical path forking and rate-matching adaptation, our final SR-List decoder with all optimizations reduces the worst-case latency to 173 CCs which is a 48.2% reduction in the worst-case latency compared to the Fast-SCL decoder.

D. Comparison With Previous Works

In Table V, we present our final implementations results for our proposed SR-List decoder for UL and compare them with the SOA architectures in [6], [12], [14], [15], [16], [21], [23]. For the comparison, we provide the results for the UL-(1024, 512) polar codes. As these previous works are implemented in different technologies, we scale their results to 28 nm. Note that we only consider the worst-case latency for [15], [16]. While they include some dynamic optimization strategies, we assume that a decoder in a 5G NR modem needs to provide reliable fixed latency guarantees. We note that in length-1024 UL polar codes, length-32 SR nodes with \( s > r \) are seldom found for middle and high-rate codes. Therefore, the RSU size is reduced to 16 in Table V to improve the hardware results.

For \( L = 2 \), our decoder has a throughput of 4.457 Gbps and an area efficiency of 29335 Mbps/mm², which is a 1.97× improvement in area efficiency compared to [23] with a small 4% reduction in throughput. For \( L = 4 \), compared with the SCL architecture with an early stopping criterion [15], our proposed SR-List decoder has a 12.3% improvement in throughput, but our area efficiency is 33.6% lower. For \( L = 8 \), the scaled throughput our decoder is 2.532 Gbps which is 3.1× higher than [12] and our area efficiency is 4165 Mbps/mm² which is a 10.6% improvement over [14].

VI. CONCLUSION

In this work, we presented a low-latency and low-complexity generalized SR node-based SCL decoder implementation optimized for 5G NR polar codes. Moreover, we applied several optimizations to significantly reduce the decoding latency and complexity of SR-List decoding to better meet the strict 5G requirements. Numerical results show that our final decoder using all optimizations has a 48.2% reduction in decoding latency compared to the SOA decoder. While some optimizations are specific to the SR node, many of our optimizations and analysis can be used with other types of polar decoders. Additionally, our

TABLE V

| Algorithm | Technology [nn] | List-size | CCs | Latency [μs] | Area [mm²] | Freq. [MHz] | 7TP [Gbps] |
|-----------|----------------|----------|-----|-------------|------------|------------|------------|
| Our work for UL¹ | SR-List | 2 | 235 | 355 | 395 | 1411 | 4.457 |
| | SCL | 4 | 2662 | 2408 | 618 | 721 | 3.619 |
| | SCU | 8 | 1325 | 1577 | 637 | 1194 | 2.532 |
| | Multi-bit | 16 | 790 | 1100 | 617 | 1426 | 1.608 |
| | Fast-SCL | 32 | 478 | 594 | 594 | 1273 | 1.273 |
| | Split-tree SCL | 64 | 2 | 2 | 2 | 2 | 2 |

¹Our works are implemented with \(|S|_{\text{max}} = 4 \) and compatible with all UL codewords, where latency is for UL-(1024, 512).

²Synthesis results are scaled to 28 nm with area \( \times s^3 \) and frequency \( \times 1/s^2 \).

³Only the worst-case latency of these dynamic implementations is considered in the scaled results.

In Table V, we present our final implementations results for our proposed SR-List decoder for UL and compare them with the SOA architectures in [6], [12], [14], [15], [16], [21], [23]. For the comparison, we provide the results for the UL-(1024, 512) polar codes. As these previous works are implemented in different technologies, we scale their results to 28 nm. Note that we only consider the worst-case latency for [15], [16]. While they include some dynamic optimization strategies, we assume that a decoder in a 5G NR modem needs to provide reliable fixed latency guarantees. We note that in length-1024 UL polar codes, length-32 SR nodes with \( s > r \) are seldom found for middle and high-rate codes. Therefore, the RSU size is reduced to 16 in Table V to improve the hardware results.

For \( L = 2 \), our decoder has a throughput of 4.457 Gbps and an area efficiency of 29335 Mbps/mm², which is a 1.97× improvement in area efficiency compared to [23] with a small 4% reduction in throughput. For \( L = 4 \), compared with the SCL architecture with an early stopping criterion [15], our proposed SR-List decoder has a 12.3% improvement in throughput, but our area efficiency is 33.6% lower. For \( L = 8 \), the scaled throughput our decoder is 2.532 Gbps which is 3.1× higher than [12] and our area efficiency is 4165 Mbps/mm² which is a 10.6% improvement over [14].
approach of simplifying the decoding process of the generalized SR node may also find use in other decoders for different types of generalized nodes. We implemented the decoder and synthesis results based on a STM 28 nm FD-SOI technology show that the proposed decoder for DL with maximum code length 512 and $L = 2$ yields a throughput of 5.065 Gbps and an area efficiency of 47.09 Gbps/mm². For length-1024 UL polar codes, the SR-List decoder with $L = 8$ achieves an area efficiency of 4.165 Gbps/mm² and a 2.532 Gbps throughput which exceeds the SOA decoders.

**ACKNOWLEDGMENT**

The authors would like to thank the anonymous reviewers for their helpful comments.

**DEDICATION**

Dedicated to Prof. Alexander Vardy (deceased March 2022), who contributed significantly to the development of polar codes and several other areas of coding theory.

**REFERENCES**

[1] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] 3rd Generation Partnership Project (3GPP) “Chairman’s Notes of Agenda Item 7.1.5 Channel coding and modulation,” 3GPP TSG RAN WG1 Meeting 87, Reno, USA, Rep. R1-1613710, Nov. 2016.

[3] 3rd Generation Partnership Project (3GPP) “5G NR: Multiplexing and Channel Coding,” 3GPP, Sophia Antipolis, France, Tech. Specification 38.212 version 15.2.0 Release 15, Jul. 2018.

[4] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[5] K. Niu and K. Chen, “CRC-aided decoding of polar codes,” *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, Oct. 2012.

[6] A. Ballatsoukas-Stimming, M. B. Parizi, and A. Burg, “LLR-based successive-cancellation list decoding of polar codes,” *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5165–5179, Oct. 2015.

[7] 3rd Generation Partnership Project (3GPP), “Final Report of 3GPP TSG RAN WG1 #AH1_NR v1.0.0,” 3GPP TSG RAN WG1 Meeting 88, Spokane, USA, Rep. R1-1701553, Feb. 2017.

[8] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation list decoders for polar codes with multibit decision,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[9] J. Lin, C. Xiong, and Z. Yan, “A high throughput list decoder architecture for polar codes,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 24, no. 6, pp. 2378–2391, Jun. 2016.

[10] C. Xiong, J. Lin, and Z. Yan, “Symbol-decision successive cancellation list decoder for polar codes,” *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 675–687, Feb. 2016.

[11] C. Xia et al., “A high-throughput architecture of list successive cancellation polar codes decoder with large list size,” *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 3859–3874, Jul. 2018.

[12] X. Liu et al., “A 5.16 Gbps decoder ASIC for polar code in 16 nm FinFET,” in *IEEE Trans. Signal Process.*, vol. 68, pp. 5954–5967, 2020.

[13] S. A. Hashemi, C. Condo, and W. J. Gross, “A fast polar code list decoder architecture based on sphere decoding,” *IEEE Trans. Commun.*, vol. 63, no. 12, pp. 2388–2390, Dec. 2016.

[14] S. A. Hashemi, C. Condo, and W. J. Gross, “Fast and flexible successive-cancellation list decoders for polar codes,” *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5756–5769, Nov. 2017.

[15] D. Kim and I.-C. Park, “A fast successive cancellation list decoder for polar codes with an early stopping criterion,” *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 4971–4979, Sep. 2018.

[16] H.-Y. Lee, Y.-H. Pan, and Y.-L. Ueng, “A node-reliability based CRC-aided polar code decoder,” in *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 3859–3874, Jul. 2018.

[17] X. Liu et al., “A 5.16 Gbps decoder ASIC for polar code in 16 nm FinFET,” in *Proc. IEEE Int. Symp. Wireless Commun. Syst.*, 2018, pp. 1–5.

[18] J. Lin, C. Xiong, and Z. Yan, “A high throughput list decoder architecture for polar codes,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[19] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation list decoders for polar codes with multibit decision,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[20] X. Liu et al., “A 5.16 Gbps decoder ASIC for polar code in 16 nm FinFET,” in *Proc. IEEE Int. Symp. Wireless Commun. Syst.*, 2018, pp. 1–5.

[21] S. A. Hashemi, C. Condo, and W. J. Gross, “A fast polar code list decoder architecture based on sphere decoding,” *IEEE Trans. Commun.*, vol. 63, no. 12, pp. 2388–2390, Dec. 2016.

[22] S. A. Hashemi, C. Condo, and W. J. Gross, “Fast and flexible successive-cancellation list decoders for polar codes,” *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5756–5769, Nov. 2017.

[23] D. Kim and I.-C. Park, “A fast successive cancellation list decoder for polar codes with an early stopping criterion,” *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 4971–4979, Sep. 2018.

[24] H.-Y. Lee, Y.-H. Pan, and Y.-L. Ueng, “A node-reliability based CRC-aided successive-cancellation list decoder architecture combined with post-processing,” *IEEE Trans. Signal Process.*, vol. 66, pp. 5954–5967, 2020.

[25] C. Kestel et al., “A 506 Gbit/s polar successive cancellation list decoder,” in *3GPP TSG RAN #AH1_NR v1.0.0*, 3GPP TSG RAN WG1 Meeting 88, Spokane, USA, Rep. R1-1701553, Feb. 2017.

[26] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation list decoders for polar codes with multibit decision,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[27] J. Lin, C. Xiong, and Z. Yan, “A high throughput list decoder architecture for polar codes,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[28] C. Xiong, J. Lin, and Z. Yan, “Symbol-decision successive cancellation list decoder for polar codes,” *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 675–687, Feb. 2016.

[29] C. Xia et al., “A high-throughput architecture of list successive cancellation polar codes decoder with large list size,” *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 3859–3874, Jul. 2018.

[30] H.-Y. Lee, Y.-H. Pan, and Y.-L. Ueng, “A node-reliability based CRC-aided successive-cancellation list decoder architecture combined with post-processing,” *IEEE Trans. Signal Process.*, vol. 66, pp. 5954–5967, 2020.

[31] C. Kestel et al., “A 506 Gbit/s polar successive cancellation list decoder with CRC,” in *Proc. IEEE Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.*, 2020, pp. 1–7.

[32] Y. Fan et al., “A low-latency list successive-cancellation decoding implementation for polar codes,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 303–317, Feb. 2016.

[33] C. Xia et al., “A 506 Gbit/s polar successive cancellation list decoder with CRC,” in *Proc. IEEE Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.*, 2020, pp. 1–7.

[34] C. Xiong, J. Lin, and Z. Yan, “A multimode area-efficient SCL polar decoder,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 24, no. 12, pp. 3499–3512, Dec. 2016.

[35] J. Lin and Z. Yan, “An efficient list decoder architecture for polar codes,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 23, no. 11, pp. 2508–2518, Nov. 2015.

[36] P. Giard et al., “PolarBear: A 28-nm FD-SOI ASIC for decoding of polar codes,” *IEEE J. Emerg. Top. Circuits Syst.*, vol. 7, no. 4, pp. 616–629, Dec. 2017.

[37] B. Yuan and K. K. Parhi, “LLR-based successive-cancellation list decoder for polar codes with multibit decision,” *IEEE Trans. Circuits Syst. II*, vol. 64, no. 1, pp. 21–25, Jan. 2017.
Yueqing Ren was born in 1996. He received the B.S. degree from the Nanjing University of Science and Technology, Nanjing, China, in 2018, and the M.E. degree from Southeast University, Nanjing, China, in 2021. He is currently pursuing a Ph.D. degree in electrical engineering under the supervision of Prof. Andreas Burg with the Telecommunications Circuits Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. His research interests include forward error correction (FEC) theory and practice, digital signal processing, and VLSI circuits for communications.

Andreas Tøftegaard Kristensen (Graduate Student Member, IEEE) was born in Hillerød, Denmark, in 1994. He received the B.Sc. degree in electrical engineering and the M.Sc. degree (Hons.) in computer science and engineering from the Technical University of Denmark, Denmark, in 2017 and 2019, respectively. He is currently working toward the Ph.D. degree in electrical engineering under the supervision of Prof. Andreas Burg with the Telecommunications Circuits Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. His research interests include vital-sign detection using Wi-Fi routers, machine learning for communications, full-duplex radio, and hardware for error-correcting codes.

Yifei Shen was born in 1997. He received the B.S. degree from the Chien-Shiung Wu College (Honors College), Nanjing, China, and the M.S. degree from the School of Information Science and Engineering, Southeast University, Nanjing, China, in 2016 and 2018, respectively. He is currently working toward the Ph.D. degree with Southeast University. From 2021 to 2022, he is a Visiting Student with the Telecommunication Circuits Laboratory, École Polytechnique Fédérale de Lausanne, Switzerland. His research interests include error-correction codes, VLSI design for digital signal processing, and synthetic biology. He was the recipient of the Best Student Paper Award at the 2016 IEEE International Conference on Digital Signal Processing and the 2020 IEEE Circuits and Systems Society Pre-Doctoral Scholarship Award.

Alexios Balatsoukas-Stimming (Member, IEEE) received the Diploma and M.Sc. degree in electronics and computer engineering from the Technical University of Crete, Chania, Greece, in 2010 and 2012, respectively, and the Ph.D. degree in computer and communications sciences from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, in 2016. He was a Marie Skłodowska-Curie Postdoctoral Fellow with the European Laboratory for Particle Physics, Meyrin, Switzerland, for one year. He was a Postdoctoral Researcher with the Telecommunications Circuits Laboratory, EPFL, from 2018 to 2019. He has also been a Visiting Postdoctoral Researcher with Cornell University, Ithaca, NY, USA, and the University of California at Irvine, Irvine, CA, USA. He is currently an Assistant Professor with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. His research interests include VLSI circuits for communications, error correction coding theory and practice, and applications of approximate computing and machine learning to signal processing for communications.
Chuan Zhang (Senior Member, IEEE) received the B.E. degree in microelectronics and the M.E. degree in very-large scale integration (VLSI) design from Nanjing University, Nanjing, China, in 2006 and 2009, respectively, and the M.S.E.E. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities (UMN), USA, in 2012. He is currently the Young Chair Professor of Southeast University, Nanjing, China. He is also with the LEADS, National Mobile Communications Research Laboratory, Frontiers Science Center for Mobile Information Communications and Security of MoE, Quantum Information Center of Southeast University, and the Purple Mountain Laboratories, Nanjing, China. His research interests include low-power high-speed VLSI design for digital signal processing and digital communications, bio-chemical computations and neuromorphic engineering, and quantum communications.

Dr. Zhang is an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS - II. He was an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING and IEEE OPEN JOURNAL OF CIRCUITS AND SYSTEMS, and a Corresponding Guest Editor for the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS twice. He is a Distinguished Lecturer and the Secretary of the Circuits and Systems for Communications TC of the IEEE Circuits and Systems Society. He is also a Member of the Applied Signal Processing Systems TC of the IEEE Signal Processing Society, and Circuits and Systems for Communications TC, VLSI Systems and Applications TC, and Digital Signal Processing TC of the IEEE Circuits and Systems Society. He was the recipient of the Best Contribution Award of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS) in 2018, the Best Paper Award in 2016, Best (Student) Paper Award of the IEEE International Conference on DSP in 2016, three Best (Student) Paper Awards of the IEEE International Conference on ASIC in 2015, 2017, and 2019, Best Paper Award Nomination of the IEEE Workshop on Signal Processing Systems in 2015, three Excellent Paper Awards and two Excellent Poster Presentation Awards of the International Collaboration Symposium on Information Production and Systems from 2016 to 2018, Outstanding Achievement Award of Intel Collaborative Research Institute in 2018, and the Merit (Student) Paper Award of the IEEE APCCAS in 2008, Three-Year University-Wide Graduate School Fellowship of UMN and the Doctoral Dissertation Fellowship of UMN.

Andreas Burg (Senior Member, IEEE) was born in Munich, Germany, in 1975. He received the Dipl.-Ing. degree from the Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, in 2000, and the Dr. sc. techn. degree from the Integrated Systems Laboratory of ETH Zurich, in 2006. In 1998, he was with Siemens Semiconductors, San Jose, CA. During his doctoral studies, he was with Bell Labs Wireless Research for a total of one year. From 2006 to 2007, he was a Postdoctoral Researcher with the Integrated Systems Laboratory and Communication Theory Group of the ETH Zurich. In 2007 he co-founded Celestrius, an ETH-spinoff in the field of MIMO wireless communication, where he was responsible for the ASIC development as the Director for VLSI. In January 2009, he joined ETH Zurich as SNF Assistant Professor and the Head of the Signal Processing Circuits and Systems Group, Integrated Systems Laboratory. In January 2011, he joined the Ecole Polytechnique Federale de Lausanne (EPFL) where he is leading the Telecommunications Circuits Laboratory. He was promoted to Associate Professor with Tenure in June 2018. In 2021, he co-founded RAAAM Technologies, a spinoff from EPFL and Bar-Ilan University, Ramat Gan, Israel, to commercialize the highest density embedded memories in any standard CMOS technology.

Mr. Burg has served on the TPC of various conferences on signal processing, communications, and VLSI. He was the TPC Co-Chair for VLSI-SoC 2012 and the TCP Co-Chair for ESSCIRC 2016 and SiPS 2017. He was the General Chair of ISLPED 2019 and he was the Editor of the IEEE TRANSACTION OF CIRCUITS AND SYSTEMS in 2013 and on the Editorial Board of the Springer Microelectronics Journal. He is currently the Editor of the Springer Journal on Signal Processing Systems, the MDPI Journal on Low Power Electronics and its Applications, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION, and the IEEE TRANSACTIONS ON SIGNAL PROCESSING. He is also the Chair of the EURASIP TAC-SPCN and a Member of the IEEE CAS-VSATC.