# Ultralow-Latency Successive Cancellation Polar Decoding Architecture Using Tree-Level Parallelism

Dongyun Kam, *Graduate Student Member, IEEE*, Hoyoung Yoo, *Member, IEEE*, and Youngjoo Lee, *Member, IEEE*

***Abstract*— Achieving the attractive error-correcting capability with a simple decoder structure, the polar code using successive cancellation (SC) decoding is now expected to be installed at the resource-limited IoT or embedded communications. However, the existing SC decoders normally suffer from the long processing latency caused by the serialized processing steps, limiting the practical applications of polar codes. In this article, to solve this latency problem, we present a new low-complexity merging operation that can increase the number of parallel factors for realizing the tree-level parallelism. We also modify the previous pruning method to further reduce the number of visited nodes at the parallel SC decoding scenario. In addition, a novel parallel partial-sum calculator (PSC) architecture is introduced to update partial-sum registers with multiple decoded bits by taking only one processing cycle. Implementation results show that the proposed 8-parallel SC polar decoder in 28-nm CMOS requires only 0.140 $\mu$s to decode a (1024, 512) codeword of 5G system, remarkably reducing the decoding latency when compared to the state-of-the-art designs.**

***Index Terms*— 5G communications, low-latency processing, parallel decoder, polar codes, successive-cancellation decoding.**

## I. Introduction

AFTER Arikan's remarkable work [1], the polar code has been popularly researched due to its attractive performance even for short-length codewords, and consequently, it has been considered in recent applications, such as 5G new radio (NR) systems [2] and IoT communications [3]. For decoding the transmitted polar codeword, the successive cancellation (SC) algorithm is normally utilized at first for providing the reasonable error-correcting capability with low computational complexity [1], [4]. However, it is well known that the conventional SC decoding suffers from a long processing delay caused by the serialized internal decoding orders [1], consequently increasing the overall decoding time of SC-based list (SCL) decoding methods enhancing the correcting performance further [5]–[11]. Therefore, it is still urgent to develop the practical low-latency processing scenario of SC decoding for realizing the cost-effective SC decoder itself at lightweight communication protocols [12] or the fast baseline architecture of SCL decoder at 5G NR solutions [13].

To solve this latency problem, a number of studies have recently reported simplifying a decoding tree, which conceptually represents the computing procedure of SC algorithm [14]–[19]. For example, the 2-b SC decoding in [14] reduces the number of the visited nodes by combining two leaves having the same parent. By considering patterns of consecutive leaf nodes, furthermore, the simplified SC (SSC) decoding algorithm in [15] remarkably reduces the decoding latency by pruning nodes having leaves of all-frozen bits (Rate-0) and all-information bits (Rate-1). Based on the SSC decoding, the work in [16] reveals that the node with mixed leaf patterns can also be pruned by applying the maximum likelihood (ML) decoding. However, the ML-based decoding cannot be used for eliminating the upper-level nodes due to the impractical amount of computational complexity. Instead of exploiting the complex ML function, the recent Fast-SSC algorithm in [17] introduces the dedicated cost-effective functions only for the frequently occurring patterns, i.e., the repetition (REP) and the single parity-check (SPC) patterns, which results in a significant reduction of the decoding latency.

However, the previous approaches are all based on the original decoding tree, which visits the processing nodes in order, and they are potentially limited by the serialized steps increasing the decoding latency. Therefore, the recent cutting-edge approaches have presented the concept of tree-level parallelism that breaks the original decoding tree into several small-sized sub-trees, theoretically supporting the parallel decoding operations [20]–[23]. As the straightforward parallel decoding requires the complex ML-like merging operations at the end of each sub-tree operation for calculating the hard-decision estimates in parallel, the previous works have developed the divide-and-conquer method for relaxing the computational costs of parallel SC decoder architecture [21], [23]. For the parallel decoding scenario, nevertheless, the previous architectures are still inefficient in terms of decoding latency as the parallel hard-decision estimates use multiple processing cycles for calculating partial-sum values that should be updated before activating the next decoding step

Manuscript received September 29, 2020; revised January 12, 2021 and February 24, 2021; accepted March 14, 2021. Date of publication April 8, 2021; date of current version June 4, 2021. This work was supported in part by the National Research Foundation (NRF) funded by the Ministry of Science and ICT (MSIT), Korean Government, under Grant 2019R1F1A1060767, in part by the MSIT through the Grand Information Technology Research Center support program supervised by the Institute for Information & Communication Technology Promotion (IITP) under Grant IITP-2020-2020-0-01612, and in part by the Samsung Electronics Company Ltd. *(Corresponding author: Youngjoo Lee.)*

Dongyun Kam and Youngjoo Lee are with the Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), Pohang 37673, South Korea (e-mail: youngjoo.lee@postech.ac.kr).

Hoyoung Yoo is with the Department of Electronics Engineering, Chungnam National University, Daejeon 34134, South Korea.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3068965.

Digital Object Identifier 10.1109/TVLSI.2021.3068965

[11], [23]–[25]. There are few works for allowing multibit updates for partial-sum values [26], [27], but those designs are suitable for the serialized SC decoding, requiring a huge amount of hardware complexity to support the parallel SC decoding with tree-level parallelism.

Based on our previous work [28], this article introduces several optimization schemes that can realize the ultralow-latency parallel SC polar decoder. For the generalized parallel decoder architecture, we first define the single-cycle merging functions for major leaf-level patterns of parallel sub-trees. As each function provides the dedicated processing path for one leaf-level pattern, it is possible to shorten the critical path when compared to the previous works handling a number of patterns with a single merging function [23]. For the minor leaf-level patterns, which cannot be supported at the prior approaches, the proposed method newly utilizes the recursive sub-parallel operations by allowing additional processing cycles. In addition to the simplified merging function, we modify the previous low-latency optimizations schemes for the single decoding tree to be suitable for the proposed parallel SC decoder design. More precisely, we limit the freedom of pruning patterns in the previous works [17], generating the pruned but identical parallel sub-trees. The novel parallel partial-sum calculator (PSC) is also developed from the previous multibit PSC by sharing the common processing units as many as possible. Targeting the recent 5G NR specification [29], for the case study, an 8-parallel 1024-b polar decoder is implemented in 28-nm CMOS technology. Applying to the proposed tree-level parallelism supported by the dedicated decoder architecture, the prototype design operates at the speed of 950 MHz, and takes only 133 processing cycles for handling a 0.5-rate 1024b codeword, reducing the decoding latency by 32.8% when compared to the previous state-of-the-art solution.

The remainder of this article is organized as follows. We first describe the backgrounds of this work in Section II, and the proposed merging unit (MU) with tree-level parallelism is presented in Section III. The modified parallel sub-tree pruning and the parallel PSC architecture are newly introduced for further reducing the processing cycles in Sections VI and V, respectively. Applying the proposed ideas, the implementation results of the 8-parallel prototype decoder are summarized and compared to the previous works in Section VI. Finally, concluding remarks are made in Section VII.

## II. Backgrounds

### A. Polar Codes and SC Decoding

An $(N, K)$ polar code is defined as a linear block code of $N$ transferred bits protecting $K$ information bits. Before the encoding process, the message vector $\mathbf{u} = [u_0, u_1, \ldots, u_{N-1}]$ is generated by assigning the $K$ information bits to the $K$ most reliable channels over $N$ channels, which are constructed by using the channel polarization phenomenon. The remaining bits, i.e., frozen bits, are then set to the fixed values known to both the encoder and the decoder, which are generally selected to all zeros [30]. As described in [1], the encoding process of polar code is denoted as a multiplication of a $1 \times N$ message vector $\mathbf{u}$ and a $N \times N$ generate matrix $\mathbf{G}^{\otimes m}$, i.e., $\mathbf{x} = \mathbf{u}\mathbf{G}^{\otimes m}$,

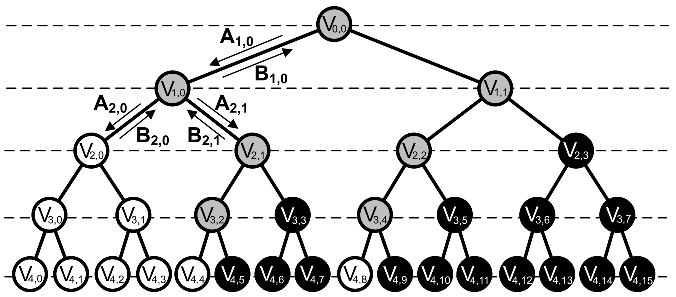

Fig. 1. Example of the SC decoding tree for a (16,10) polar code.

where $m = \log_2 N$, $\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and $\mathbf{G}^{\otimes m}$ is the $m$th Kronecker power of the polarizing matrix $\mathbf{G}$.

After transferring the codeword vector $\mathbf{x}$ through the noisy channel, the polar decoder observes the $1 \times N$ vector $\mathbf{y}$, and then starts to estimate the message bits $\hat{\mathbf{u}} = [\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{N-1}]$. For a polar code of length $N$, it is well known that the processing steps of the conventional SC decoding algorithm can be represented as a $\log_2 N$-depth binary decoding tree [15]. For the case of (16, 10) polar code, Fig. 1 illustrates the conceptual diagram of the SC decoding tree, where the black, white, and gray nodes represent the nodes having all-information, all-frozen, and mixed leaves, respectively. To represent the SC decoding process in detail, let $V_{i,j}$ be the $j$th node in each $i$th stage on the decoding tree, having $L_i = 2^{m-i}$ leaf nodes. A node $V_{i,j}$ receives a soft-value vector $\mathbf{A}_{i,j} = [\alpha_0^{i,j}, \alpha_1^{i,j}, \ldots, \alpha_{L_i-1}^{i,j}]$ generated by its parent node and returns an estimated hard-value vector $\mathbf{B}_{i,j} = [\beta_0^{i,j}, \beta_1^{i,j}, \ldots, \beta_{L_i-1}^{i,j}]$. Note that the soft-value vector $\mathbf{A}_{0,0}$ of the root node $V_{0,0}$ is initialized by using the received vector $\mathbf{y}$ as follows:

$$\alpha_l^{0,0} = \log\left(\frac{\Pr(y_l|x_l = 0)}{\Pr(y_l|x_l = 1)}\right), \quad 0 \le l \le N-1. \tag{1}$$

In the SC decoding algorithm, the node $V_{i,j}$ transmits the new soft-value vectors, i.e., $\mathbf{A}_{i+1,2j}$ and $\mathbf{A}_{i+1,2j+1}$ to left child and right child nodes, respectively. More precisely, the elements of two soft-value vectors are calculated as follows:

$$\begin{aligned} \alpha_l^{i+1,2j} &= F\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}\right) \\ \alpha_l^{i+1,2j+1} &= G\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}, \beta_l^{i+1,2j}\right) \end{aligned} \tag{2}$$

where $l$ denotes the index of each vector, satisfying $0 \le l \le L_{i+1} - 1$. With the hardware-friendly algorithm in [31], $F$ and $G$ are defined as

$$\begin{aligned} F(x, y) &= \mathrm{sgn}(x)\mathrm{sgn}(y)\min(|x|, |y|) \\ G(x, y, u) &= (-1)^u x + y \end{aligned} \tag{3}$$

where $\mathrm{sgn}(x)$ returns 1 if $x \ge 0$ and $-1$ otherwise. After receiving the hard-value vectors $\mathbf{B}_{i+1,2j}$ and $\mathbf{B}_{i+1,2j+1}$ from left and right child nodes, respectively, the node $V_{i,j}$ then generates the new hard-value vector $\mathbf{B}_{i,j}$, i.e., a partial-sum vector by performing the following calculations:

$$\begin{aligned} \beta_l^{i,j} &= \beta_l^{i+1,2j} \oplus \beta_l^{i+1,2j+1} \\ \beta_{l+L_{i+1}}^{i,j} &= \beta_l^{i+1,2j+1}. \end{aligned} \tag{4}$$

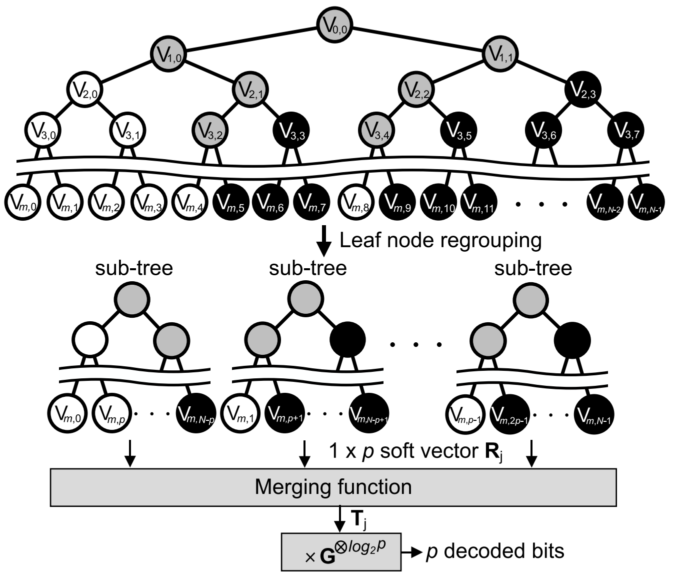

Fig. 2. Concept of $p$-parallel SC decoding [20].

At the $j$th leaf node $V_{m,j}$, note that the hard-value vector $\mathbf{B}_{m,j}$ is nothing but an estimated bit, i.e., $\mathbf{B}_{m,j} = [\beta_0^{m,j}] = [\hat{u}_j]$, which can be computed as

$$\hat{u}_j = \beta_0^{m,j} = \begin{cases} 0, & \text{if } j \in \mathcal{F} \text{ or } \alpha_0^{m,j} \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

where $\mathcal{F}$ stands for the set consisting of the indices of frozen-bit positions. The conventional SC decoding is then finished when the right-most leaf node $V_{m,N-1}$ computes the final hard-decision estimate, and the decoded output $\hat{\mathbf{u}}$ is finally estimated as $[\beta_0^{m,0}, \beta_0^{m,1}, \ldots, \beta_0^{m,N-1}]$.

As the $G$ operation of the node $V_{i,j}$ requires partial-sums generated by the previous estimated bits denoted as a vector form $\mathbf{B}_{i+1,2j}$, the conventional SC decoding generally suffers from the long processing delay due to the serialized decoding orders. To reduce the decoding latency, the works [14]–[19] present the pruning-based SC decoding algorithms, which prune the sub-trees having some patterns of consecutive leaf nodes. In particular, the Fast-SSC decoding in [17] uses the low-complexity pruning methods for four patterns of consecutive leaf nodes: Rate-0, Rate-1, REP, and SPC patterns. Note that the pruned tree for the four patterns provides the same error-correcting performance as the original tree and significantly reduces the number of visited nodes, leading to the low-latency decoding compared to the original SC algorithm. However, the pruning-based approaches are also based on the original SC decoding tree and still have latency limitations from the serialized processing orders of the decoding tree.

## *B. Parallel SC Decoding With Multiple Sub-Trees*

To solve the limitation of serialized processing steps, the parallel polar decoding approaches are introduced in [20], [22], [23]. Instead of handling a large-sized decoding tree, more precisely, the work in [20] introduces $p$ small-sized sub-trees by decomposing the original one, which can be processed in parallel. Fig. 2 depicts the concept of the $p$-parallel SC decoding process, where $p$ is typically a power of two. Note that each sub-tree contains $N/p$ leaf nodes from the original tree, constructing a binary tree architecture of $m' = m - \log_2 p$ levels. In other words, a sub-tree contains $L_i' = 2^{m'-i}$ leaf nodes. In this article, for the sake of simplicity, a node in the $k$th parallel tree is denoted as $V(k)_{i,j}$ where $0 \leq k \leq p-1$. Note that the leaf node $V(k)_{m',j}$ corresponds to the leaf node $V_{m,pj+k}$ in the original tree. Similar to the processing of the original SC tree, each node of parallel sub-trees receives an $1 \times L_i'$ soft-value vector $\mathbf{A}(k)_{i,j}$ from its parent node, and then returns the hard-value estimate $\mathbf{B}(k)_{i,j}$. To initialize multiple root nodes, i.e., $V(k)_{0,0}$, we define the first soft-value vector for each sub-tree, which is denoted as $\mathbf{A}(k)_{0,0}$, by selecting $N/p$ values from $\mathbf{A}_{0,0}$ in a round-robin fashion. More precisely, the $l$th element of $\mathbf{A}(k)_{0,0}$ becomes $\alpha(k)_l^{0,0} = \alpha_{pl+k}^{0,0}$. After the initializing process, all the sub-trees start the conventional SC decoding simultaneously. Whenever the decoding step reaches a leaf node of each sub-tree at the same time, the $p$ soft values of $V(k)_{m',j}$ are collected to construct a vector $\mathbf{R}_j = [r_0^j, r_1^j, \ldots, r_{p-1}^j] = [\alpha(0)_0^{m',j}, \alpha(1)_0^{m',j}, \ldots, \alpha(p-1)_0^{m',j}]$ for activating the merging function in Fig. 2. The merging function shown is applied to the soft-value vector $\mathbf{R}_j$ for estimating an $1 \times p$ hard-value vector, which is denoted as $\mathbf{T}_j = [t_0^j, t_1^j, \ldots, t_{p-1}^j]$ where $t_k^j = \beta(k)_0^{m',j}$. It is required to find the vector $\mathbf{T}_j$ that maximizes the following cost function:

$$C(\mathbf{R}_j, \mathbf{T}_j) = \sum_{k=0}^{p-1} \left(1 - 2t_k^j\right) r_k^j. \tag{6}$$

Similar to the conventional SC decoding, after estimating the hard-value vector, the parallel sequence of decoded information can be generated at a time by multiplying $\mathbf{G}^{\otimes \log_2 p}$, i.e., $[\hat{u}_{pj}, \hat{u}_{pj+1}, \ldots, \hat{u}_{pj+p-1}] = \mathbf{T}_j \mathbf{G}^{\otimes \log_2 p}$. Although the previous parallel SC decoding algorithm potentially reduces the number of processing cycles by utilizing the parallel sub-tree architecture, the direct implementation may cause even a slow decoder realization due to the long critical delay of merging function [20]. More precisely, for finding the maximum cost value of (6), we have to evaluate at most $2^p$ cases of $\mathbf{T}_j$ for the given $\mathbf{R}_j$, resulting in $p2^p$ multiplications, $(p-1)2^p$ additions, and $2^p - 1$ comparisons, requiring a tremendous amount of computing costs especially for the large parallel factor $p$. In order to mitigate the computational overheads, the prior work from [23] reduces the complexity of the merging function by using a divide-and-conquer method that considers few possible combinations of $\mathbf{T}_j$ for evaluating (6). However, the previous method still requires time-consuming sorting operations followed by a number of multiplications, making the decoder system slow or taking multiple cycles for operating the merging operations. In order to provide the ultralow-latency polar decoding, hence, it is necessary to develop a cost-effective way for realizing the merging function suitable for the massive-parallel processing.

TABLE I
RATIO OF FROZEN PATTERNS IN THE 5G POLAR CODES [29]

| $W_j$ | Code length $N$ | | | | | |
|---|---|---|---|---|---|---|
| | 32 | 64 | 128 | 256 | 512 | 1024 |
| 0×00 | 0 | 0.06 | 0.177 | 0.247 | 0.278 | 0.30 |
| 0×01 | 0 | 0.115 | 0.123 | 0.107 | 0.089 | 0.075 |
| 0×03 | 0 | 0.038 | 0.023 | 0.019 | 0.016 | 0.016 |
| 0×07 | 0.036 | 0.032 | 0.026 | 0.021 | 0.018 | 0.014 |
| 0×17 | 0.107 | 0.103 | 0.104 | 0.086 | 0.073 | 0.062 |
| 0×1F | 0.036 | 0.045 | 0.030 | 0.022 | 0.019 | 0.016 |
| 0×3F | 0.036 | 0.038 | 0.029 | 0.023 | 0.016 | 0.013 |
| 0×7F | 0.214 | 0.18 | 0.126 | 0.106 | 0.092 | 0.083 |
| 0×FF | 0.571 | 0.38 | 0.355 | 0.366 | 0.397 | 0.420 |
| Others | 0 | 0.006 | 0.005 | 0.002 | 0.001 | 0.0006 |

TABLE II
RATIO OF FROZEN PATTERNS IN THE POLAR CODES FROM [30]

| $W_j$ | Code length $N$ | | | | | |
|---|---|---|---|---|---|---|
| | 32 | 64 | 128 | 256 | 512 | 1024 |
| 0×00 | 0.25 | 0.125 | 0.25 | 0.25 | 0.312 | 0.336 |
| 0×01 | 0 | 0.25 | 0.063 | 0.156 | 0.094 | 0.078 |
| 0×03 | 0 | 0.125 | 0.063 | 0.031 | 0.109 | 0.016 |
| 0×07 | 0 | 0 | 0 | 0 | 0 | 0.016 |
| 0×17 | 0.5 | 0 | 0.25 | 0.125 | 0.104 | 0.102 |
| 0×1F | 0 | 0 | 0 | 0 | 0.016 | 0.016 |
| 0×3F | 0 | 0.125 | 0.063 | 0.031 | 0.031 | 0.016 |
| 0×7F | 0 | 0.25 | 0.063 | 0.126 | 0.109 | 0.109 |
| 0×FF | 0.25 | 0.125 | 0.25 | 0.25 | 0.297 | 0.313 |
| Others | 0 | 0 | 0 | 0 | 0 | 0 |

## III. PROPOSED LOW-COMPLEXITY MU

### A. Analysis of Frozen Patterns

To reduce the complexity of merging operation in Fig. 2, similar to the concept of previous work [23], we investigate the major patterns of frozen-bit positions for the given polar code structure. For the sake of simplicity, let $W_j$ denote the pattern value whose binary notation shows the frozen patterns of $p$ parallel leaf nodes each of which is from the different sub-tree at the same time. More precisely, $W_j = \sum_{k=0}^{p-1} w_k^j 2^{p-k-1}$, where $w_k^j$ is 0 or 1 if the leaf node $V(k)_{m',j}$ is the frozen or information node, respectively. As the frozen position provides the deterministic information, it is possible to construct the dedicated functions for a different pattern value $W_j$. However, similar to solve (6) directly, it is also impractical for preparing the dedicated functions for all the possible $W_j$. By observing the frequent $W_j$ values for the given polar code structure, as described in the work from [23], it is possible to narrow down the size of the problem significantly, providing the practical processing scenario for exploiting the tree-level parallelism.

Targeting the recent 5G polar codes in [29], for example, Table I depicts the ratio of different $W_j$ values for designing the 8-parallel SC decoder. Note that only nine patterns, i.e., $0 \times 00$, $0 \times 01$, $0 \times 03$, $0 \times 07$, $0 \times 17$, $0 \times 1F$, $0 \times 3F$, $0 \times 7F$, and $0 \times FF$, comprise about more than 99% of 256 possible frozen patterns by analyzing all the code rates defined in the 5G communication [29]. For the case of other polar code structures constructed in 2.5-dB AWGN channel [18], [30], similarly, Table II reveals that the selected nine $W_j$ patterns from Table I also covers most of the practical cases. If we change the parallel factor, it is still possible to find the major frozen patterns by investigating all the possible cases defined by the target system. When the 4-parallel decoding is considered for decoding 1024-b 5G polar codes, for example, only four patterns ($0 \times 0$, $0 \times 1$, $0 \times 7$ and $0 \times F$) account for 98% of existing frozen patterns. Targeting the same 5G codes, similarly, we can observe that 16 dominant patterns cover more than 99% of total cases for realizing the 16-parallel decoder. Therefore, it is clear that our approach can be applied to support any parallel factor without loss of generality, finding the major operations to be accelerated.

For the parallel factor of 8, which is used for realizing the prototype decoder, it is interesting that we found the same major patterns as the previous work from [23], utilizing the ML-like operations to manage the selected patterns simultaneously. Although the previous method partially relaxes the merging complexity with early evictions of less important candidates, however, the direct evaluation of (6) for the survived patterns is still time-consuming, causing the multicycle realization [23]. Moreover, the previous design cannot accept the minor patterns, which occurred in the real world depending on the construction ways of polar codes as exemplified in Table I. In the proposed work, on the other hand, we develop a novel method to handle the major patterns, leading to the fast realization while even supporting any kind of minor patterns.

### B. Low-Complexity Merging Function

Inspired by the previous work for the serialized decoding case [17], which also provides the dedicated functions for reducing the complexity of ML-like operations only for the pre-defined cases, we newly define nine dedicated functions denoted as $M_{W_j}(\cdot)$ as summarized in Table III. Adopting the dedicated functions, in other words, the hard-value vector is directly calculated without evaluating the complex equations of (6) when the current frozen pattern belongs to the selected major patterns, i.e., $\mathbf{T}_j = M_{W_j}(\mathbf{R}_j)$. To simplify the expression of these dedicated functions $M_{W_j}$, two primitive functions denoted as $\mathbf{y}_i = \text{REP}_i(\mathbf{z}_i)$ and $\mathbf{y}_i = \text{SPC}_i(\mathbf{z}_i)$ are introduced, where $\mathbf{y}_i = [y_0, y_1, \ldots, y_{i-1}]$ and $\mathbf{z}_i = [z_0, z_1, \ldots, z_{i-1}]$ represent $1 \times i$ hard-value and soft-value vectors, respectively. Note that the function $\text{REP}_i(\cdot)$ in the table represents the repetition pattern of frozen positions and simply returns the all-zero or all-one vector by examining $y_k = H(\sum_{l=0}^{i-1} z_l)$, where $H(x)$ is 1 for $x < 0$ and 0 otherwise. On the other hand, the function $\text{SPC}_i(\cdot)$ is the same as the pruning operation of the single-parity-check node defined in [17], and each element of the output vector $\mathbf{y}_i$ can be calculated as follows:

$$y_k = \begin{cases} H(z_k) \oplus P(\mathbf{z}_i), & \text{if } |z_k| = \min(|z_0|, \ldots, |z_{i-1}|) \\ H(z_k), & \text{otherwise} \end{cases} \tag{7}$$

TABLE III
DEFINITIONS OF THE DEDICATED MERGING FUNCTIONS $M_{W_j}$

| $W_j$ | Definition |
|---|---|
| 0x00 | $\mathbf{T}_j = \mathbf{0}$ |
| 0x01 | $\mathbf{T}_j = REP_8(\mathbf{R}_j)$ |
| 0x03 | $[t_0^j, t_2^j, t_4^j, t_6^j] = REP_4([r_0^j, r_2^j, r_4^j, r_6^j])$ <br> $[t_1^j, t_3^j, t_5^j, t_7^j] = REP_4([r_1^j, r_3^j, r_5^j, r_7^j])$ |
| 0x07 | $[t_0^j, t_1^j, t_2^j, t_3^j] = [t_4^j, t_5^j, t_6^j, t_7^j]$ <br> $= SPC_4([r_0^j + r_4^j, r_1^j + r_5^j, r_2^j + r_6^j, r_3^j + r_7^j])$ |
| 0x17 | $[c_0, c_1, c_2, c_3] = REP_4([F(r_0^j, r_4^j), F(r_1^j, r_5^j),$ <br> $F(r_2^j, r_6^j), F(r_3^j, r_7^j)])$ <br> $[t_4^j, t_5^j, t_6^j, t_7^j] = SPC_4([G(r_0^j, r_4^j, c_0), G(r_1^j, r_5^j, c_1),$ <br> $G(r_2^j, r_6^j, c_2), G(r_3^j, r_7^j, c_3)])$ <br> $[t_0^j, t_1^j, t_2^j, t_3^j] = [c_0 \oplus t_4^j, c_1 \oplus t_5^j, c_2 \oplus t_6^j, c_3 \oplus t_7^j]$ |
| 0x1F | $[c_0, c_1, c_2, c_3] = REP_4([F(r_0^j, r_4^j), F(r_1^j, r_5^j),$ <br> $F(r_2^j, r_6^j), F(r_3^j, r_7^j)])$ <br> $[t_4^j, t_5^j, t_6^j, t_7^j] = [H(G(r_0^j, r_4^j, c_0)), H(G(r_1^j, r_5^j, c_1)),$ <br> $H(G(r_2^j, r_6^j, c_2)), H(G(r_3^j, r_7^j, c_3))])$ <br> $[t_0^j, t_1^j, t_2^j, t_3^j] = [c_0 \oplus t_4^j, c_1 \oplus t_5^j, c_2 \oplus t_6^j, c_3 \oplus t_7^j]$ |
| 0x3F | $[t_0^j, t_2^j, t_4^j, t_6^j] = SPC_4([r_0^j, r_2^j, r_4^j, r_6^j])$ <br> $[t_1^j, t_3^j, t_5^j, t_7^j] = SPC_4([r_1^j, r_3^j, r_5^j, r_7^j])$ |
| 0x7F | $\mathbf{T}_j = SPC_8(\mathbf{R}_j)$ |
| 0xFF | $t_k^j = H(r_k^j)$ |

where $P(\mathbf{z}_i) = \bigoplus_{l=0}^{i-1} H(z_l)$. Based on these primitive functions defined above, note that we optimize each dedicated function by considering the correlation of the soft input values in $\mathbf{R}_j$ as detailed in Table III.

In order to derive each dedicated function, we first simplify the cost function (6) by considering the internal constraints of the hard-estimate vector $\mathbf{T}_j$, which is defined by the given frozen patterns $W_j$. Then, it is possible to find the closed-form to compute $\mathbf{T}_j$ that maximizes the reformulated cost function. When we observe the frozen pattern of $W_j = 0 \times 03$, for example, an $1 \times 8$ binary vector of eight corresponding decoded bits, denoted as $\hat{\mathbf{u}}_j = [\hat{u}_{8j+0}, \hat{u}_{8j+1}, \ldots, \hat{u}_{8j+7}]$, should include zero bits depending on the frozen pattern, i.e., $\hat{u}_{8j+0} = \hat{u}_{8j+1} = \hat{u}_{8j+2} = \hat{u}_{8j+3} = \hat{u}_{8j+4} = \hat{u}_{8j+5} = 0$. As the decoded bits can be derived by multiplying the hard-estimate vector and the corresponding generate matrix, i.e., $\hat{\mathbf{u}}_j = \mathbf{T}_j \mathbf{G}^{\otimes 3}$, we can directly conclude that $t_0^j = t_2^j = t_4^j = t_6^j = \hat{u}_{8j+6} \oplus \hat{u}_{8j+7}$ and $t_1^j = t_3^j = t_5^j = t_7^j = \hat{u}_{8j+7}$. If this prior knowledge is applied to (6), the cost function $C(\mathbf{R}_j, \mathbf{T}_j)$ can be simplified as $(1-2t_0^j)r_0^j + (1-2t_1^j)r_1^j + (1-2t_0^j)r_2^j + (1-2t_1^j)r_3^j + (1-2t_0^j)r_4^j + (1-2t_1^j)r_5^j + (1-2t_0^j)r_6^j + (1-2t_1^j)r_7^j = (1-2t_0^j)(r_0^j + r_2^j + r_4^j + r_6^j) + (1-2t_1^j)(r_1^j + r_3^j + r_5^j + r_7^j)$. To maximize this simplified cost function, the estimated bits $t_0^j$ and $t_1^j$ should be determined by the sign values of $(r_0^j + r_2^j + r_4^j + r_6^j)$ and $(r_1^j + r_3^j + r_5^j + r_7^j)$, respectively. By using the function $\mathrm{REP}_i(\cdot)$, this procedure can be finally represented as $[t_0^j, t_2^j, t_4^j, t_6^j] = \mathrm{REP}_4([r_0^j, r_2^j, r_4^j, r_6^j])$ and $[t_1^j, t_3^j, t_5^j, t_7^j] = \mathrm{REP}_4([r_1^j, r_3^j, r_5^j, r_7^j])$. As depicted in Table III, it is possible to similarly define other dedicated functions for all the major patterns $W_j$.

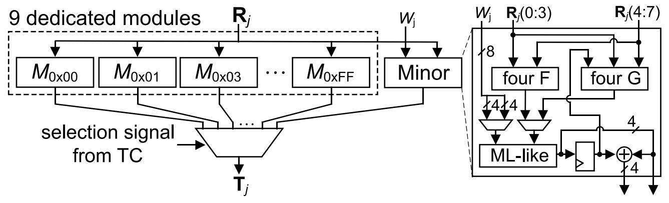

Fig. 3. Proposed merging unit for 8-parallel SC decoding.

As we only realize the nine dedicated functions for major patterns without solving the ML problem directly, the hardware costs of merging operation can be remarkably reduced, even allowing the single-cycle operation. If a minor pattern is detected during the parallel SC decoding, it is categorized into two 4-b frozen patterns. We simply utilize $F(\cdot)$ and $G(\cdot)$ functions in (3) to generate 4-parallel soft messages corresponding to the categorized patterns and then calculate 4-parallel hard-value estimates by utilizing 4-parallel ML-like operations. In other words, we perform a divide-and-conquer approach to handle the 8-b minor patterns, allowing one more processing cycle than the major cases. However, the additional overheads caused by these minor patterns can be negligible if we consider the major patterns covering more than 99% in practice as shown in Tables I and II. Therefore, the proposed method focusing on the major patterns is a practical and effective way for realizing the massive-parallel SC decoding.

## C. Proposed MU Architecture

Fig. 3 illustrates the detailed architecture of the proposed MU that accelerates the major leaf-node patterns, where $p$ is set to 8. In contrast that the previous work [23] still implements the ML-like function associated with the reduced number of candidates, we develop in this work nine processing paths, each of which corresponds to the dedicated function depicted in Table III, i.e., directly calculating the hard-estimate $\mathbf{T}_j$ from the soft-vector $\mathbf{R}_j$ with simple procedures. Therefore, the critical delay of each processing function, denoted as $M_{W_j}$ Fig. 3, is remarkably reduced when compared to that of the prior design [23]. The output of the proposed MU is then selected based on the current leaf-node pattern $W_j$. Due to the simple processing paths of dedicated functions, it is possible to perform the proposed MU in only one cycle whereas the previous MU architecture necessitates multiple cycles caused by long critical delay [23].

When the minor patterns are detected during the parallel decoding, however, it is inevitable to perform the ML-like processing for merging the results from $p$ sub-trees [20]. In this case, we introduce a divide-and-conquer strategy to reduce the size of ML-like processing. More precisely, the proposed MU starts with $p/2$ $F(x, y)$ operations in (3) for handling the minor patterns, and an $p/2$-input ML-like function is followed to determine the first $p/2$ bits at a time as shown in Fig. 3. Then, the latter part is dealt with $p/2$ $G(x, y, u)$ operations followed by the same $p/2$-input ML-like function. Fig. 4 shows the detailed architecture to solve the 4-input ML

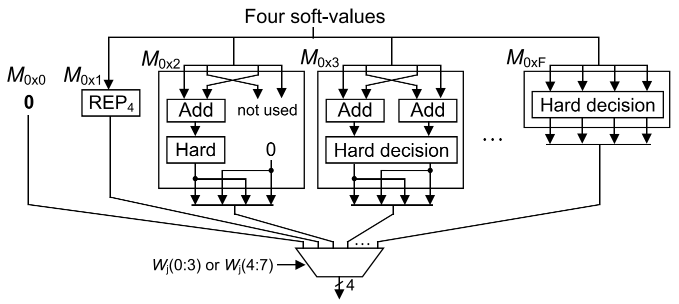

Fig. 4. Detailed architecture of ML-like function for minor patterns.

problem that supports all the minor patterns existing in the proposed 8-parallel processing. Even though the maximum number of candidates is now reduced from 256 to 16, handling the ML problem in a naïve way is still complex due to the serialized evaluation and sorting sequences [16]. Instead of the straightforward realization, we present in this work an ML-like function utilizing 16 dedicated functions each of which is dedicated to a single 4-b minor pattern. Similar to the dedicated functions for 8-b major patterns shown in Table III, it is possible to derive a closed-form equation to obtain the optimal solution for the fixed minor pattern. Due to the reduced number of bits for denoting the given frozen-bit pattern, note that the required logic-level complexity for realizing these functions is naturally smaller than that for major cases, which is acceptable at the practical parallel decoder. Note that the ML-like function shown in Fig. 4 accepts the current frozen pattern to select one of the 4-b estimates from 16 dedicated modules, always providing the same results as the serialized baseline SC decoding [4]. Inserting pipelined registers between $F$ and $G$ operations, in order not to increase the critical delay, we allow two processing cycles to manage the detected minor patterns. Under the two-cycle processing, as shown in Fig. 3, the hardware part for the $p/2$-input ML-like function can be shared in a time-interleaved manner to minimize the additional overheads, increasing the overall decoder complexity by less than 1%. By applying dedicated processing paths for both the major and minor cases, as a result, the proposed MU successfully supports all the existing frozen patterns in the given polar codewords.

Table IV shows the implementation results of MU designs for 8-parallel SC decoding in a 28-nm CMOS technology, where the previous architecture from [23] is modified to support the single cycle processing for a fair comparison. Due to the dedicated processing paths, as we expected, the proposed MU reduces the critical delay by more than 30%. Considering the operating frequency of the typical polar decoder architecture, which will be discussed in the following sections, it is inevitable to insert pipeline registers at the prior MU design, taking additional processing cycles even for the major leaf-node patterns. Moreover, note that the previous design for parallel-SC decoding cannot support minor patterns, as it only focuses on the major patterns that are the only leaf-node patterns of a certain polar code [23]. In the practical applications, however, it is possible to observe few minor patterns as revealed in Table I, and the prior work should include these patterns, increasing the processing delay further.

TABLE IV
IMPLEMENTATION RESULTS OF MERGING UNITS

| MU architecture | Proposed | [23] |
|---|---|---|
| Parallel factor | 8 | 8 |
| Critical path delay | 0.54 ns | 0.8 ns |
| Gate count[a] | 13,091 | 9,655 |

[a] Implemented at each critical path delay.

## IV. Pruning Scheme for Parallel Decoding

### A. Pruning Parallel Sub-Trees

In order to further reduce the processing latency of parallel SC decoding, in this work, the concept of the previous pruning method in [17] is modified and applied to reduce the number of activated nodes at each sub-tree. As $p$ leaf nodes should be visited at the same time, it is important to make the same pruning shapes to each parallel sub-tree, which is the basic concept of the proposed parallel pruning scheme. Fig. 5 depicts how the proposed parallel pruning is applied to the 4-parallel (16, 9) SC polar decoding example, which is based on a set of leaf nodes having the same positions in each sub-tree. For each set including $h = kp$ nodes, where $k$ is a natural number, we examine whether the frozen pattern of the constructed set is matched to the pre-defined pruning patterns.

For developing the proposed parallel pruning method, we selectively adopt the pruning patterns of the previous pruning approach specialized for the serialized SC decoding tree [17], [28]. More precisely, a new soft-value vector, which is exemplified in Fig. 5, is constructed by combining four soft-value vectors: $\mathbf{A}(0)_{1,1}$, $\mathbf{A}(1)_{1,1}$, $\mathbf{A}(2)_{1,1}$ and $\mathbf{A}(3)_{1,1}$. Note that this combined soft-value vector is then identical to $\mathbf{A}_{1,1}$ of the original SC tree. Therefore, the set for the parallel pruning shown in Fig. 5 becomes consecutive leaves at the original tree, and the previous pruning operations [17] can be applied to eliminate this set without loss of generality. Considering the complexity of the parallel pruning unit (PU), which will be described in the following section, we only accept four pruning patterns from [17], i.e., Rate-0, Rate-1, REP, and SPC patterns. Note that we always choose the size of the pruning set to be a multiple of $p$, so that the pruned sub-trees still exploit the proposed tree-level parallelism associated with the identical pruned structure. Combined with the massive-parallel processing associated with the simplified merging function, as a result, the proposed parallel pruning scheme offers the ultralow-latency SC decoding algorithm by minimizing the number of visited nodes even in the parallel decoding procedure.

### B. PU Architecture

To support the proposed parallel pruning algorithm, we also develop the parallel PU design whose internal architecture is detailed in Fig. 6. The proposed PU supports four pruning

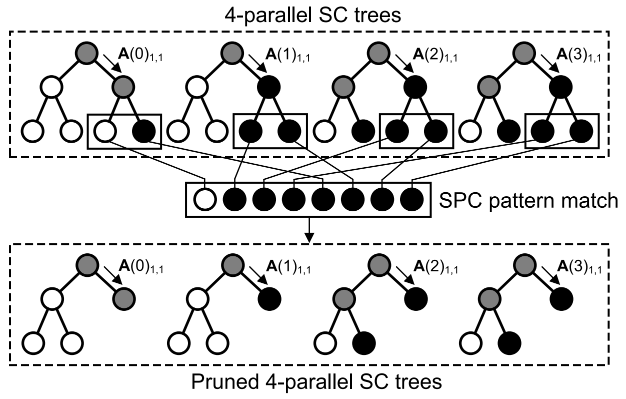

Fig. 5. Example of pruning sub-trees for a 4-parallel SC polar decoding.

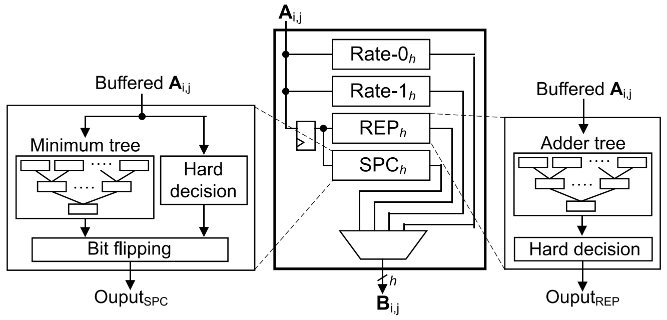

Fig. 6. Proposed pruning unit using tree architectures.

patterns of Rate-0, Rate-1, REP, and SPC cases, directly decoding more than $p$ bits at a time. Depending on the polar code structure, we pre-define the processing sequence reflecting the result of parallel pruning, which is managed by the top-level controller. When the pruning operation is activated, the proposed PU collects the current $h$ soft-values from $p$ parallel sub-trees, generating $\mathbf{A}_{i,j}$. The maximum number of pruned nodes is selected by considering the tradeoff between the hardware complexity and the number of removed nodes with the parallel pruning scheme.

Similar to the proposed MU design, as depicted in Fig. 6, we design dedicated processing paths for handling each pruning pattern individually, relaxing the delay overheads caused by additional PU logic. For the case of Rate-0 and Rate-1 pruning patterns, more precisely, we can simply set the output estimate vector as all zeros and sign bits, respectively, requiring no additional hardware as reported in [17]. When we observe REP and SPC patterns, on the other hand, it is required to perform $\text{REP}_h(\cdot)$ and $\text{SPC}_h(\cdot)$ operations described in Section III. To support these cases efficiently, as depicted in Fig. 6, the $\text{REP}_h(\cdot)$ and $\text{SPC}_h(\cdot)$ operations are realized by using tree architectures of additions and comparisons, respectively. These trees are designed to support the maximum number of pruned nodes $h$, and the dummy inputs are introduced when the input pattern has fewer nodes. As the large-size tree-like processing in general requires a long processing delay [17], as shown in Fig. 6, we allow an additional processing cycle by inserting pipelined registers in front of the tree architectures for SPC and REP nodes. Implemented in a 28-nm CMOS process, the critical delay of the parallel PU design becomes 0.6 ns, which is similar to that of the proposed MU shown in Table IV. It is reasonable that we can allow much more parallel inputs to PU architecture with simple pruning patterns compared to the MU design with complex leaf-node patterns, some of whose merging functions take long processing delays due to the complex data-level dependences as depicted in Table III.

## V. Proposed Parallel PSC

### A. Parallel Multibit Partial-Sum Update

To perform the $G$ function shown in (3), the SC-based polar decoder requires partial-sums computed by the prior decoded bits. In the conventional serialized SC polar decoding, the partial-sums can be directly generated by updating a decoded bit to the partial-sum registers of each stage, which stores the combined values from the previous decoded bits [4], [24], [25]. As the proposed decoding method estimates multiple bits in parallel, similar to the previous pruning-based SC decoders [17], the naive implementation of (4) uses the recursive partial-sum computation with additional processing cycles, increasing the overall decoding latency [17]. To relax the latency overheads for recursively updating partial-sums, for decoding a polar code of $N = 2^m$ bits, the generate matrix $\mathbf{G}^{\otimes m}$ can be partially reconstructed on demand by utilizing the additional matrix generation (MG) unit to support the multibit updates of the pruning-based serialized SC decoding [27], [32]. Note that the $N/2$ registers are additionally utilized to temporally store an $1 \times N/2$ binary vector, denoted as $\mathbf{S}(x) = [s_0(x), s_1(x), \ldots, s_{N/2-1}(x)]$, for the single-cycle parallel partial-sum updates (PSUs), where $x$ represents the index of the most recently updated bit ($0 \le x < N$). As reported in [27], the pruning-based SC decoding generates $t$ estimated bits, and then the current partial-sum register $\mathbf{S}(x)$ is updated to $\mathbf{S}(x+t)$ as follows:

$$\mathbf{S}(x+t) = \mathbf{S}(x) \oplus [\mathbf{D}(x+t)\cdot\mathbf{IN}, \mathbf{D}(x+t)\cdot\mathbf{IN}] \quad (8)$$

where $\mathbf{D}(x) = [d_0(x), d_1(x), \ldots, d_{N/4-1}(x)]$ is the $(x\%N/4)$-th row of $\mathbf{G}^{\otimes \log_2 N/4}$. The $1 \times N/4$ vector $\mathbf{IN}$ is constructed by placing $t$ estimated bits in parallel. In general, $N/4$ is divisible by $t$, and therefore we need to repeatedly allocate the estimated bits to fill the $N/4$ positions of $\mathbf{IN}$ if $t \neq N/4$. Fig. 7 illustrates the previous PSC preforming (8), which is composed of an MG module, a PSU module, and a crossbar network [27]. Instead of storing all the patterns of $\mathbf{G}^{\otimes \log_2 N/4}$, it is possible to generate the next row $\mathbf{D}(x+t)$ on demand from the current one $\mathbf{D}(x)$ as follows:

$$\mathbf{D}(x+t) = \mathbf{D}(x) \oplus [\mathbf{0}, d_0(x), d_1(x), \ldots, d_{N/4-1-t}(x)] \quad (9)$$

where the row-wise zero-vector $\mathbf{0}$ contains $t$ elements. As the value of $t$ can vary depending on the size of pruned patterns, as a result, we need to introduce a number of multiplexers at the MG module as shown in Fig. 7. Note that the input size of multiplexers is gradually increased up to $\lceil \log_2 2t_{\max} \rceil$, where $t_{\max}$ is the maximally allowable bits to be decoded in parallel.

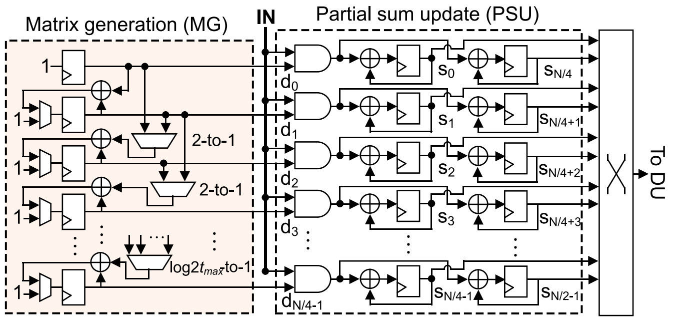

Fig. 7. Previous partial-sum calculator for $N$-bit polar codes [27].

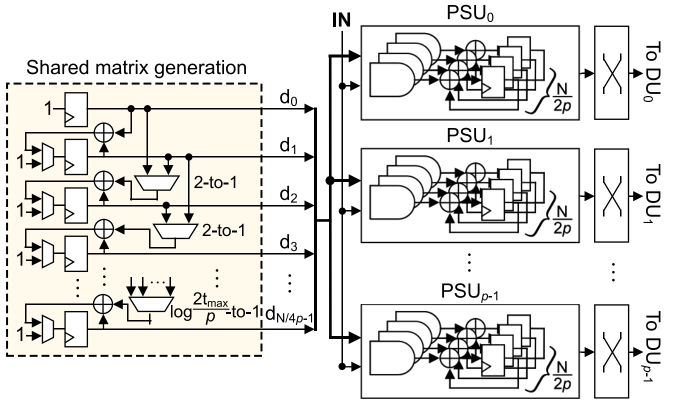

Fig. 8. Proposed low-cost parallel partial-sum calculator.

If we directly apply the prior PSC design for supporting the tree-level parallelism, it is required to utilize $p$ identical PSC modules each of which is allocated to a single sub-tree. To mitigate the complexity overheads, as depicted in Fig. 8, we introduce the novel parallel SC architecture that can share the MG unit providing a common row-wise vector $\mathbf{D}$ to $p$ PSU modules. This is mainly possible as the proposed parallel sub-trees have an identical shape from the parallel pruning method described in Section IV. As the size of the sub-tree is smaller than that of the original decoding tree, moreover, it is possible to realize the MG unit for taking care of $\mathbf{G}^{\otimes \log_2 N/4p}$, accordingly reducing the hardware complexity. More precisely, we utilize only $N/4p$ flip-flops for MG unit, and the maximum size of multiplexer for generating $\mathbf{D}(x+t)$ is reduced from $\lceil \log_2 2t_{\max} \rceil$ to $\lceil \log_2 2t_{\max}/p \rceil$ as illustrated in Fig. 8. Note that each PSU module handles $1 \times N/2p$ partial-sum vector, thus the total complexity of $p$ PSU modules becomes identical to that of one PSU design for the original decoding tree in Fig. 7. Therefore, the proposed design sharing MG module effectively supports a single-cycle parallel partial-sum computation for the parallel SC decoding, even reducing the hardware costs.

## B. High-Speed Partial-Sum Crossbar

When the typical line decoder architecture is used for realizing the SC processing [4], it is necessary to properly connect partial-sums to processing elements (PEs) by using the crossbar as depicted in Fig. 7, which normally utilizes a wide-input multiplexer per each PE [24]. In order to realize the single-cycle PSC without degrading the operating frequency of the polar decoder, it is important to reduce the critical path that starts from the crossbar to the corresponding PEs. Therefore, it is required to carefully analyze the worst case, i.e., the largest size of a multiplexer of crossbar design. Among total $q$ PEs, as PE$_0$ works at every stage in line architecture, it is well known that the conventional crossbar forces to PE$_0$ having the largest multiplexer [24]. Fig. 9(a) shows a simple example of the previous method applied to the 8-PE line decoder for 32-b polar codes, which connects five partial-sum registers to PE$_0$ [24]. It is noticeable that the selection signal of the 5-to-1 multiplexer is determined by the current decoding stage, and there are severe imbalances in terms of the accessing counts on the connected partial-sum registers. For example, as shown in the figure, s$_8$ is accessed for all the decoding stages, whereas s$_{10}$ and s$_{14}$ are only selected once at the last stage (stage 5). Therefore, it is possible to reduce the number of connected registers by additionally introducing $2^{\log_2\ N-t}$ dummy registers dedicated only to PEs for processing the $t$th stage. As exemplified in Fig. 9(b), for this case study, the proposed method adds one more register s$'_0$ for handling the last stage, which reduces the size of the input multiplexer for PE$_0$ from 5 to 4. For the case of longer polar codes, in general, we can reduce the crossbar overheads for each PE by adding the dedicated register from the last stage until it reduces the number of inputs.

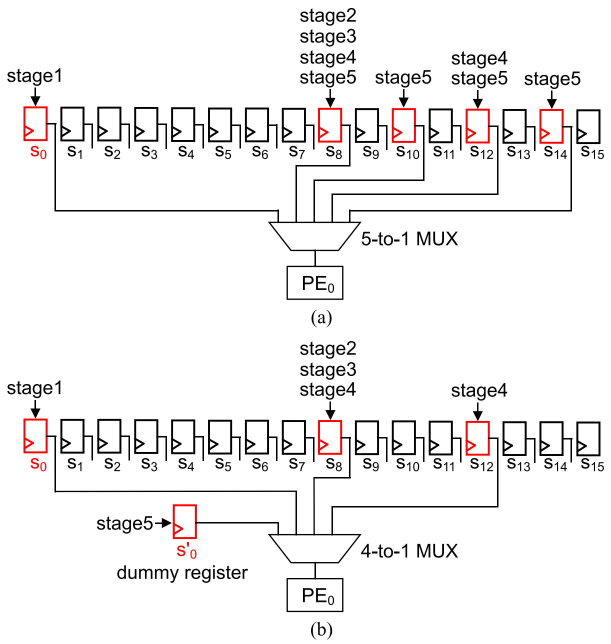

Fig. 9. (a) Conventional crossbar design for PE$_0$. (b) Proposed architecture reducing the multiplexer overheads.

For supporting the SC decoding of 1024-b polar codes, Table V compares different PSC architectures using the generate matrix reconstruction, which are equally implemented at the speed of 950 MHz in 28-nm CMOS technology. Compared to the previous architecture [27], the proposed MG module and

TABLE V
IMPLEMENTATION RESULTS OF VARIOUS PSC ARCHITECTURES

| PSC architecture | Proposed | [27] | [24] |
|---|---|---|---|
| Multiple-bit update | Available | Available | Not available |
| Largest MUX size in MG | 4-to-1 | 7-to-1 | - |
| Largest MUX size in crossbar | 6-to-1 | 129-to-1 | 129-to-1 |
| Number of MUXs | 417 | 2369 | 704 |
| Number of REGs | 600 | 768 | 768 |
| Gate count[a] | 10,762 | 22,711 | 16,854 |

[a] Implemented at the speed of 950MHz.

crossbar design drastically reduce the number of registers as well as the number of multiplexers by targeting the 8-parallel SC decoding trees. Note that the proposed crossbar successfully relaxes the maximum size of multiplexer from 129-to-1 to 6-to-1 with additional seven dummy registers for the last three decoding stages, reducing the overall critical delay by 10%. As a result, the proposed architecture significantly optimizes both the hardware overhead and the delay, leading to the cost-effective realization while supporting the single-cycle PSU even for the multibit decoding scenario.

## VI. SIMULATION AND IMPLEMENTATION RESULTS

### *A. Algorithm-Level Performance*

To confirm the algorithm-level performances of the proposed 8-parallel SC polar decoding, we simulated the error-correcting performances of the proposed decoding algorithm and compared it to the conventional SC polar decoding case. More precisely, we evaluated bit-error-rate (BER) and frame-error rate (FER), which are defined to the ratios of error cases to the transmitted data in terms of bit and codeword levels, respectively, where the channel condition is denoted as $E_b/N_0$ for representing the ratio between the energy per bit and the noise power spectral density. Fig. 10 illustrates the BER and FER performances for decoding (1024, 512) 5G polar codes, where each codeword internally includes a CRC-24 check [29]. As the proposed parallel SC decoding fully considers the frozen patterns even for the minor cases, it is obvious that the error-correcting performances from the parallel processing are identical to those of the baseline serialized SC decoding. Considering the practical polar decoder implementation, we additionally simulated BER and FER performances using 5 b fixed-point numbers [12], which are enough to represent the internal values of the proposed parallel decoding process allowing the negligible performance loss as shown in Fig. 10.

To show the impacts of the proposed parallel-SC algorithm, as illustrated in Fig. 11, we then evaluated how much the different optimizations affect the decoding latency in terms of the processing cycles. For the fair estimation, we used the line decoder architecture to evaluate different decoding schemes, which consists of 512 PEs for 1024-b 5G polar codes [4]. Note that the proposed tree-level parallelism associated with the single-cycle merging functions definitely leads to the ultralow-latency SC decoding for any code rate of 5G specification by

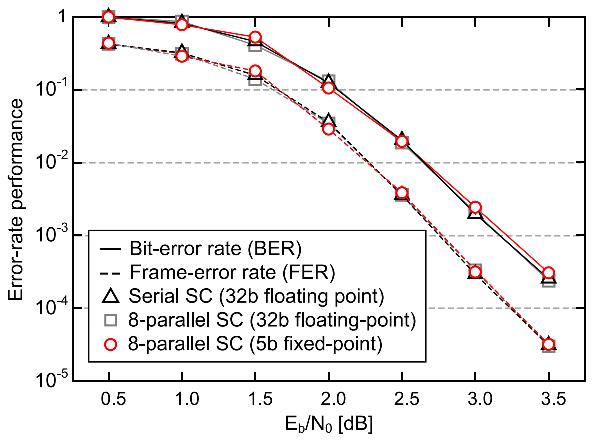

Fig. 10. BER and FER performances of different SC decoding algorithms for (1024, 512) 5G polar codes.

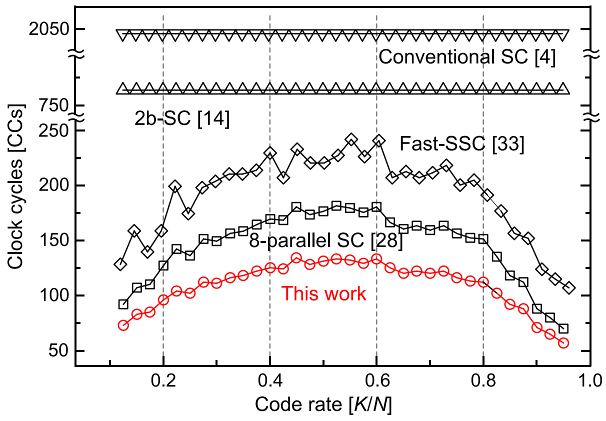

Fig. 11. Latency evaluations of different SC decoding algorithms at the line decoder architecture, including 512 PEs for 1024-b 5G polar codes.

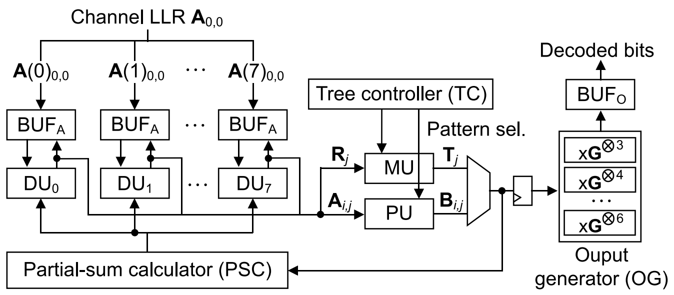

Fig. 12. Block diagram of 8-parallel polar decoder architecture.

utilizing the parallel processing of small-sized sub-trees when compared to the serialized algorithms [4], [14] and the state-of-the-art pruning-based algorithm [33]. The proposed PSC design further reduces decoding cycles by taking only one clock cycle for updating the partial-sum registers in parallel even compared to our latest work [28]. As a result, the fully optimized parallel SC decoder necessitates only 131 clock cycles to decode a 0.5-rate 1024-b 5G code, which is 15.4 and 1.32 times faster than the conventional SC decoding [4] and our previous parallel algorithm [28], respectively.

### *B. Prototype Decoder*

To evaluate the hardware-level improvements, we designed and realized a prototype 8-parallel polar decoder in 28-nm CMOS technology. As illustrated in Fig. 12, the prototype

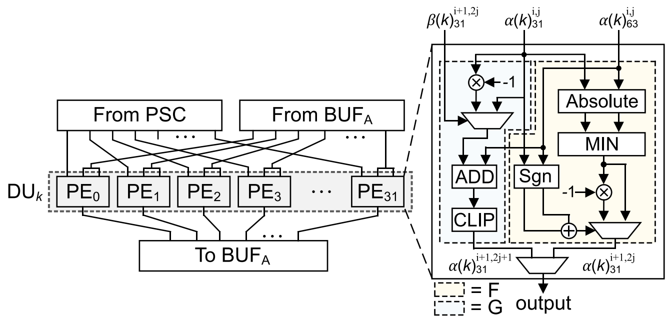

Fig. 13. Detailed architecture of the proposed decoding unit (DU).

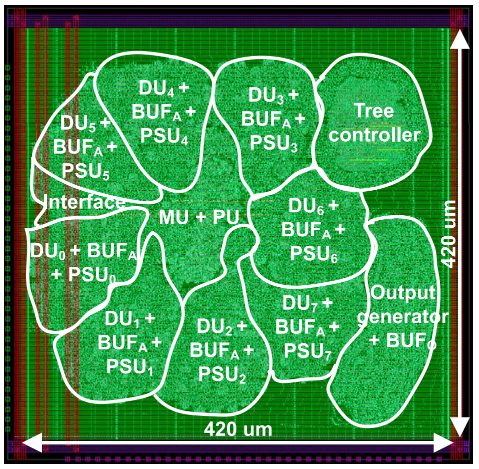

Fig. 14. Layout of the proposed 8-parallel SC polar decoder.

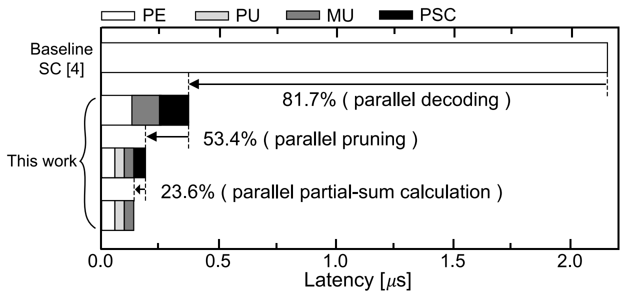

Fig. 15. Latency reductions with the proposed optimization schemes.

decoder basically includes eight pairs of a decoding unit (DU) connected to an internal buffer ($\mathrm{BUF_A}$). Managed by the top-level tree controller (TC), we utilized the proposed MU, PU, and PSC designs for supporting the proposed tree-level parallelism effectively, which are precisely described in the previous sections. Note that the final decoded codeword is computed at the output generator (OG) followed by the output buffer ($\mathrm{BUF_O}$).

During the parallel SC decoding steps, the $k$-th DU accesses the current soft-vector $\mathbf{A}(k)_{i,j}$ and partial-sums from the corresponding $\mathrm{BUF_A}$ and PSC, respectively, and generates $\mathbf{A}(k)_{i+1,2j}$ and $\mathbf{A}(k)_{i+1,2j+1}$. The generated soft-vectors are then stored in the buffer for the case of intermediate node processing. When the current node is the leaf of the pruned parallel tree, on the other hand, the results are also transferred to MU or PU for calculating the hard-estimate vector that is transferred to PSC and OG modules. More specifically, the output decoded bits are computed by multiplying the hard-estimate vector $\mathbf{T}_j$ or $\mathbf{B}_{i,j}$ by the generate matrix with the proper size when the current leaf nodes in DUs are the bottom nodes of parallel trees or intermediate nodes resulted from the pruning operation, respectively. The selection of the hard-estimate vector from MU or PU is managed by a TC module that contains the control sequence of each code structure by considering the current $W_j$ as well as the parallel pruning results. Note that we use the pipelined registers before calculating the output decoded bits, cutting the critical delay for fast decoder realization as shown in Fig. 12. Targeting (1024, 512) codewords, the prototype decoder successfully supports eight decoding sub-trees in parallel, each of which is mapped to a line-based DU architecture with 32 PEs as depicted in Fig. 13 [4]. In order to optimize the processing latency, in addition, the decoder introduces the parallel pruning method up to 64 leaf nodes if the collected patterns from parallel sub-trees belong to the pre-defined formats, i.e., Rate-0, Rate-1, REP, and SPC cases. By reducing the critical delay caused by some multiplexers having an excessive number of PE inputs, the decoder can operate at the speed of 950 MHz in 28-nm CMOS technology, occupying the silicon area of 0.18 $\mathrm{mm^2}$ as shown in Fig. 14.

From the conventional serial SC decoding in [4], Fig. 15 illustrates how the proposed hardware-level optimizations gradually reduce the latency of processing operations in (1024, 512) 5G polar codes. Note that the conventional architecture only utilizes the PEs that serially generate the decoded bit one by one, requiring a long decoding periods. Adopting the proposed tree-level parallelism we can reduce the required latency from PEs by a factor of $p$, however, there are some overheads caused by MU and PSC operations. The parallel pruning then allows reducing the number of activated nodes, accordingly saving the decoding latency of each processing module. The decoding latency is optimized by using the proposed parallel PSC design, as depicted in Fig. 15, finally reducing the overall decoding time by 93.4% compared to the baseline decoding scheme [4].

To select acceptable configurations of the proposed parallel decoding for a 1024-b polar code, we have also analyzed the tradeoff between the decoding latency and the hardware complexity according to the number of DUs and PEs per DU as shown in Fig. 16. The decoding latency is gradually reduced by increasing the number of DUs, i.e., increasing the parallel factor, where each DU equally contains 32 PEs. Similarly, for the fixed 8-parallel processing, it is obvious that the decoding latency is reduced by adopting more PEs per DU design as also reported in [4]. It is natural that we necessitate more hardware resources to lower the processing latency, and therefore it would be required to select the proper hardware-level configurations by considering both the latency and the complexity. Note that the 8-parallel decoder having 32 PEs per DU successfully presents the attractive point, which is used for the prototype design, reducing a sufficient amount of processing delay with the acceptable complexity overheads.

Table VI summarizes the implementation results of different SC polar decoders targeting a 1024-b polar code. For a fair comparison, the processing latency is normalized to support

TABLE VI
IMPLEMENTATION RESULTS OF SC POLAR DECODERS

| | **This work** | [28] | [23] | [4] | [14] | [17] | [18] | [32] | [33] | [13] |
|---|---|---|---|---|---|---|---|---|---|---|
| Decoding scheme | Parallel SC | Parallel SC | Parallel SC | SC | 2b-SC | Fast-SSC | Fast-SSC | Fast-SSC | Fast-SSC | Fast-SSC |
| Technology (nm) | 28 | 65 | 90 | 65 | 45 | 65 | 65 | 65 | 65 | 65 |
| Number of PEs | 256 | 256 | 256 | 64 | 1022 | 64 | 64 | 64 | 64 | 64 |
| Parallel factor | 8 | 8 | 8 | - | - | - | - | - | - | - |
| Frequency (MHz) | 950 | 650 | 409 | 500 | 750 | 450 | 450 | 420 | 430 | N/A |
| Area (mm$^2$) | 0.09 | 0.30 | 2.31 | 0.30 | N/A | 0.60 | 0.44 | 0.57 | 0.64 | 0.38 |
| Gate count | 197.5K | 208.3K | N/A | N/A | 338.5K | N/A | N/A | N/A | N/A | N/A |
| Processing cycles[a] | 133 | 174 | 304 | 2080 | 767 | 270 | 252 | 214 | 198 | N/A |
| Decoding latency[a] ($\mu$s) | 0.140 | 0.267 | 0.74 | 4.16 | 1.022 | 0.6 | 0.56 | 0.51 | 0.46 | 0.66 |
| Decoding throughput (Mbps) | 7341 | 3835 | 2887 | 246 | 1002 | 1719 | 1829 | 2000 | 2213 | 1557 |
| Area efficiency[b] (Gbps/mm$^2$) | 15.1 | 12.7 | 2.4 | 0.82 | N/A | 2.87 | 4.16 | 3.51 | 3.46 | 4.10 |
| Energy efficiency (pJ/b) | 5.2 | 26.9 | N/A | N/A | N/A | 188.41 | 124.69 | 173.07 | 169.89 | 102.65 |

[a] For decoding a (1024, 512) 5G polar code [29].
[b] Scaled area results to 65nm CMOS technology .

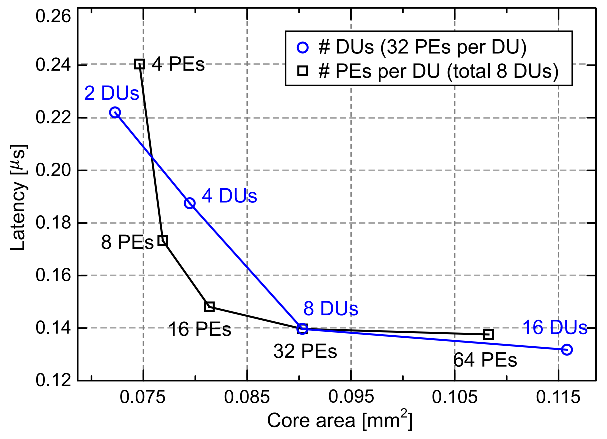

Fig. 16. Tradeoff between the decoding latency and the hardware costs.

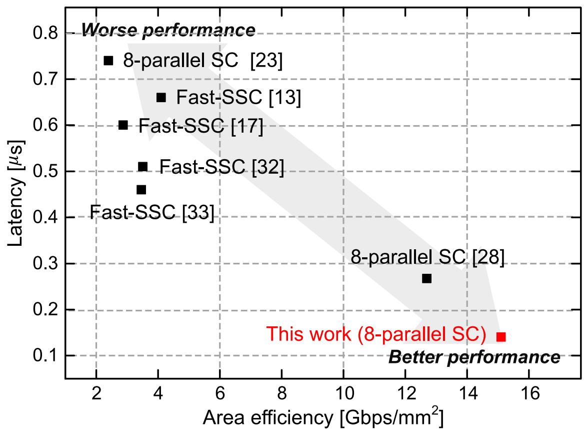

Fig. 17. Evaluation chart of different SC polar decoder designs.

the code construction method used for the 5G system [29]. By adopting the proposed parallel decoder architecture, our prototype design significantly reduces the number of decoding cycles. More precisely, the proposed work takes only 133 processing cycles to decode a 1024-b codeword, which shortens the decoding latency by 93.6%, 32.8%, and 56.8% when compared to the conventional serialized SC decoding [4], the recent pruning-based serial decoding [33] and the previous 8-parallel decoding method with the serialized PSC architecture [28], respectively. Considering the operating frequency, as a result, the proposed design requires only 0.140 $\mu$s for decoding a 1024-b codeword, achieving the shortest processing latency among the existing designs as depicted in Table VI. The reduced latency of the proposed parallel architecture directly leads to the high-throughput design, achieving the decoding throughput of more than 7.3 Gbps, which is 1.91 times improvement even compared to the previous work having the same parallel factor [28]. Fig. 17 illustrates the evaluation chart showing the efficiencies of different SC polar decoders by comparing the area efficiency and the decoding latency. Note that the proposed design achieves the ultralow-latency decoding operation by accelerating the major leave patterns at the parallel SC algorithm. At the same time, the prototype design only occupies 0.09 mm$^2$ due to the reduced sub-tree sizes associated with the proposed hardware-level optimization methods, clearly outperforming the other related works in terms of area efficiency as depicted in Fig. 17. Therefore, the proposed optimization methods successfully offer practical ways for utilizing the tree-level parallelism, achieving the ultralow-latency SC polar decoding.

## VII. CONCLUSION

In this article, we have proposed novel optimization schemes for realizing the ultralow-latency SC polar decoder using tree-level parallelism. The proposed single-cycle MU significantly reduces the processing delay by focusing on the major leaf-node patterns while supporting the rarely occurred minor patterns with recursive paths. Based on the previous pruning philosophy for the serialized decoding, the proposed parallel pruning generates the identical parallel sub-trees, minimizing the number of visited nodes at the parallel SC decoding scenario. The PSC architecture is finally introduced to support the single-cycle parallel update of partial-sum registers without increasing the hardware complexity. By breaking prior bottlenecks to realize the massive-parallel architecture, we successfully implement an 8-parallel SC polar decoder in 28-nm CMOS technology, which overcomes the previous state-of-the-art designs in terms of the decoding latency as well as the hardware efficiency, leading to the ultralow-latency error-correction solution especially for the lightweight wireless communication systems.

## References

[1] E. Arikan, "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] *Chairmans Notes of Agenda Item 7.1. 5 Channel Coding and Modulation*, document TSG RAN WG1 Meeting no. 87, 3GPP, Oct. 2016.

[3] A. Sharma and M. Salim, "Polar code: The channel code contender for 5G scenarios," in *Proc. IEEE Int. Conf. Comput. Commun. Electron. (Comptelix)*, Jul. 2017, pp. 676–682.

[4] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 289–299, Jan. 2013.

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

[6] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 12, pp. 2368–2380, Dec. 2016.

[7] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 8, pp. 609–613, Aug. 2014.

[8] A. Balatsoukas-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.

[9] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[10] J. Lin, C. Xiong, and Z. Yan, "A high throughput list decoder architecture for polar codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2378–2391, Jun. 2016.

[11] J. Lin and Z. Yan, "An efficient list decoder architecture for polar codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2508–2518, Nov. 2015.

[12] O. Dizdar and E. Arikan, "A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 3, pp. 436–447, Mar. 2016.

[13] F. Ercan, T. Tonnellier, and W. J. Gross, "Energy-efficient hardware architectures for fast polar decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 322–335, Jan. 2020.

[14] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.

[15] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.

[16] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725–728, Apr. 2013.

[17] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.

[18] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast low-complexity decoders for low-rate polar codes," *J. Signal Process. Syst.*, vol. 90, no. 5, pp. 675–685, May 2018.

[19] M. Hanif and M. Ardakani, "Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes," *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2360–2363, Nov. 2017.

[20] B. Li, H. Shen, and D. Tse, "Parallel decoders of polar codes," Sep. 2013, *arXiv:1309.1026*. [Online]. Available: http://arxiv.org/abs/1309.1026

[21] C. Xiong, J. Lin, and Z. Yan, "Symbol-based successive cancellation list decoder for polar codes," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2014, pp. 198–203.

[22] B. Li, H. Shen, D. Tse, and W. Tong, "Low-latency polar codes via hybrid decoding," in *Proc. 8th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Aug. 2014, pp. 223–227.

[23] C. Xiong, J. Lin, and Z. Yan, "A multimode area-efficient SCL polar decoder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 12, pp. 3499–3512, Dec. 2016.

[24] Y. Fan and C.-Y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3165–3179, Jun. 2014.

[25] M. Mousavi, Y. Fan, C.-Y. Tsui, J. Jin, B. Li, and H. Shen, "Efficient partial-sum network architectures for list successive-cancellation decoding of polar codes," *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 3848–3858, Jul. 2018.

[26] T. Che and G. Choi, "An efficient partial sums generator for constituent code based successive cancellation decoding of polar codes," 2016, *arXiv:1611.09452*. [Online]. Available: http://arxiv.org/abs/1611.09452

[27] J. Han, R. Liu, and R. Wang, "Simplified multi-bit SC list decoding for polar codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 996–1000.

[28] D. Kam and Y. Lee, "Ultra-low-latency parallel SC polar decoding architecture for 5G wireless communications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[29] *Multiplexing and Channel Coding*, document 38.212, V.15.3.0, 3GPP, Sep. 2018.

[30] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.

[31] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2011, pp. 1665–1668.

[32] F. Ercan, C. Condo, and W. J. Gross, "Reduced-memory high-throughput fast-SSC polar code decoder architecture," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2017, pp. 1–6.

[33] F. Ercan, T. Tonnellier, C. Condo, and W. J. Gross, "Operation merging for hardware implementations of fast polar decoders," *J. Signal Process. Syst.*, vol. 91, no. 9, pp. 995–1007, Nov. 2018.

**Dongyun Kam** (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2018 and 2020, respectively, where he is currently working toward the Ph.D. degree.

His current research interests include low-latency polar decoding, next-generation communication algorithms, and architectures for digital systems.

**Hoyoung Yoo** (Member, IEEE) received the B.S. degree in electrical and electronics engineering from Yonsei University, Seoul, South Korea, in 2010, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2012 and 2016, respectively.

Since 2016, he has been with the Department of Electronics Engineering, Chungnam National University, Daejeon, where he is currently an Associate Professor. His current research interests include VLSI for 5G systems and reverse engineering for FPGA chips.

**Youngjoo Lee** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2008, 2010, and 2014, respectively.

Since 2017, he has been with the Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea, where he is currently an Associate Professor. Prior to joining POSTECH, he was with Interuniversity Microelectronic Center (IMEC), Leuven, Belgium, from 2014 to 2015, where he researched reconfigurable SoC platforms for software-defined radio systems. From 2015 to 2016, he was with the faculty of the Department of Electronic Engineering, Kwangwoon University, Seoul, South Korea. His current research interests include the algorithms and architectures for embedded processors, intelligent mobile multimedia systems, advanced error-correction codes, and next-generation communication systems.