IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 October 2022, accepted 9 November 2022, date of publication 14 November 2022, date of current version 17 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3221742

RESEARCH ARTICLE

Low-Latency Multi-Kernel Polar Decoders

HOSSEIN REZAEI, (Graduate Student Member, IEEE), NANDANA RAJATHEVA, (Senior Member, IEEE), AND MATTI LATVA-AHO, (Senior Member, IEEE)
Centre for Wireless Communications, University of Oulu, 90570 Oulu, Finland
Corresponding author: Hossein Rezaei (hossein.rezaei@oulu.fi)

This work was supported by the Academy of Finland through 6G Flagship Program under Grant 346208.

ABSTRACT Polar codes have been receiving increased attention for application in beyond 5G networks. They offer low-complexity decoding algorithm and can achieve symmetric channel capacity. However, the majority of research works have focused on the codes constructed by the binary kernel (2×2 polarization matrix) which bounds the code length to an integer power of 2. Multi-kernel polar codes have been proposed as a method that allows the construction of polar codes with sizes different from powers of 2 by mixing multiple kernels of different dimensions. A hardware implementation based on the successive cancellation (SC) algorithm found in the literature shows that it suffers from a long decoding latency. In this paper, we design and implement a multi-kernel decoder based on the fast-simplified SC (fast-SSC) algorithm to decrease the decoding latency. It can decode any code constructed by binary and ternary (3×3) kernels featuring flexible code length, code rate, and kernel sequence. FPGA implementation results reveal that a polar code of length $N = 1536$, rate $\mathcal{R} = 1/2$ with Processing Element (P_e) value of $P_e = 240$, gains 84.6% lower latency compared to the original algorithm. Also, the architecture supports polar codes constructed by purely-binary and purely-ternary kernels. A polar code of length $N = 1024$, rate $\mathcal{R} = 1/2$, and $P_e = 120$ achieves an information throughput of 432 Mbps.

INDEX TERMS FPGA, hardware implementation, low lateny, polar code, successive cancellation, URLLC.

I. INTRODUCTION

Polar codes, proposed by Arikan [1], can achieve the symmetric channel capacity using the channel polarization phenomenon when the code length approaches infinity. Thanks to their low-complexity implementation, it has gained considerable attention under successive cancellation (SC) decoding algorithm. Moreover, the 3GPP standardization organization has considered polar codes as a coding scheme in the fifth generation (5G) of mobile communication standards. The reliability of polar codes under the state-of-the-art cyclic redundancy check (CRC) aided successive cancellation list (SCL) decoding [2], [3], makes them an ideal choice for ultra-reliable low-latency communication (URLLC) systems using beyond 5G network.

However, polar codes suffer from high decoding latency originated by the serial nature of the SC algorithm. Various researchers have tried to reduce the latency of the SC decoding. There are different algorithms such as simplified-SC (SSC) [4], maximum-likelihood (ML) nodes [5] and fast-SSC [6], which considerably decrease SC decoding latency. New node patterns are proposed in [7] and [8] to further reduce the decoding latency. Polar codes presented in [9] lower the decoding latency at the cost of losing some error-correction performance. The works in [10] and [11] implemented sequence repetition fast-SSC (SRFSC) algorithm to decrease the latency of polar codes. The authors in [12] proposed pipelined combinational SC that effectively decreases the latency of polar codes at the cost of significant increase in hardware complexity. Finally, memory footprint optimization and operation merging are capable of lowering the latency of fast-SSC hardware architecture by consuming less memory in the implementation phase [13], [14].

Another drawback tied to polar codes is that the codewords are limited to those constructed by Kronecker product expansion of binary (2×2) kernel which results in bounding

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

the code length to powers of 2. However, practical applications demand various block lengths with different rates. To increase the achievable code length, puncturing [15] and shortening [16] methods have been proposed. However, these methods cost additional optimization steps and decoding complexity inconsistent with their block lengths.

Multi-kernel polar codes have been proposed to increase the length and rate flexibility of polar codes [17]. They can employ larger kernels in their code construction along with binary kernel. Specifically, a ternary (3 × 3) kernel offers desirable flexibility in constructing a multi-kernel code supporting code lengths that are powers of 2, powers of 3, or a product of both with a reasonable decoding complexity overhead. In [18], the first architecture for a multi-kernel successive cancellation polar decoder from binary and ternary kernels is proposed. It supports any code length and code rate up to the maximum supported code length. However, it uses the SC algorithm which suffers from a very large decoding latency. Also, it lacks the support for applications demanding long block lengths since the maximum supported code length is 4096.

In [19], we have implemented an algorithm optimized for short packet communications to decode short polar codes constructed by pure binary kernel. The contribution of this paper is threefold. First, we optimize the algorithm in [19] to further decrease the latency of polar codes with long block lengths. Second, we extend the algorithm to support multi-kernel polar codes constructed from binary-ternary mixed kernels. We also introduce some new patterns to prune multi-kernel polar tree. The algorithm we give, is rate-flexible and decodes any code constructed by purely-binary, purely-ternary, and binary-ternary mixed kernels. Finally, based on the proposed architecture for the conventional fast-SSC in [6], a hardware architecture will be presented. The FPGA implementation results is compared to the state-of-the-art schemes in terms of latency, throughput, and implementation cost.

The remainder of the paper is organized as follows. In Section II, the preliminaries of polar codes with variants of the SC algorithm and code construction by binary-ternary mixed kernels are given. Section III outlines the proposed algorithm. In Section IV, the proposed hardware architecture is detailed. The FPGA implementation of the proposed architecture and performance analysis are summarized in section V. Finally, section VI concludes this work.

II. POLAR CODES

A. CODE CONSTRUCTION

A polar code of length N that carries K information bits is denoted as $\mathcal{P}(N, k)$. The encoder usually sets the remaining N-k bits to a determined value (mainly zero). The code rate can be computed as $R \triangleq k/N$. Arikan proposed channel polarization [1] as a code construction method under SC decoding to reach the symmetric channel capacity ($I(W)$) of the binary-input discrete memoryless channel (B-DMC) W.

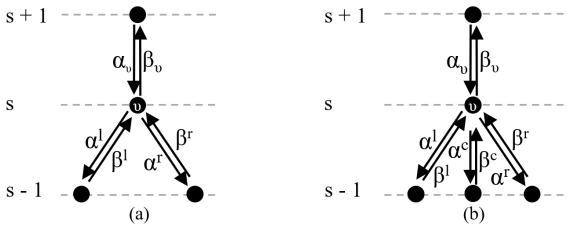

FIGURE 1. (a) Binary and (b) ternary node message passing.

As the code length increases, the reliability of each individual channel W_i^N ($1 \leq i \leq N$) approaches to either one (perfectly reliable ($I(W_i^N) \rightarrow 1$)) or zero (perfectly unreliable ($I(W_i^N) \rightarrow 0$)). Determining the optimal location of information and frozen bits may differ depending on the channel type and method of code construction. In this work, for polar code construction, we have used the method proposed in [1] using a systematic encoding scheme.

The authors in [20] have proposed a generalized construction approach for polar codes. Along with the binary kernel, larger kernels have also been explored in this work. This construction method outperforms the puncturing [15] and shortening [16] methods. This method offers error-correction performance gains ranging from 0.1 dB to 1.1 dB at frame error rate (FER) of almost 10^{-3} with reference to puncturing [15] and shortening [16] methods.

The encoding process can be represented through linear transformation $x = uG$, where u is a N-bit input vector to the encoder, G is the generator matrix and x is the encoder output.

The polarization matrices for binary and ternary (3×3) [20] kernels are proposed as

$$T_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \qquad T_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

In multi-kernel codes of length $N = n_0 \times n_1 \times \ldots \times n_s$ with n_is not necessarily distinct prime numbers, G is constructed as a series of Kronecker product between kernels of different sizes in form of $G \triangleq T_{n_0} \otimes T_{n_1} \otimes \ldots \otimes T_{n_s}$ where T_{n_i}s are squared matrices.

B. SUCCESSIVE CANCELLATION DECODING

To decode a codeword under SC algorithm, the decoder needs to traverse the polar binary tree which is composed of $n + 1$ levels with $n = \log_2 N$. Let $\lambda \in [0, n]$ be the level of a given node in the polar binary tree. The leaves and the root are located at level 0 and n, respectively, and $2^{n-\lambda}$ leaves exists under each processing node. The log-likelihood ratios (LLRs), defined as $\alpha_n = \{\alpha_0, \alpha_1, \ldots \alpha_{N-1}\}$, enter from the root and they need to visit all leaves to get decoded. The LLRs need three functions to traverse the tree. For a given node v (Fig. 1 (a)), α_{v_l} is the function required to travel to the left branch and it can be estimated as

$$\alpha_{v_l}[i] = sgn(\alpha_v[i].\alpha_v[i + 2^{(\lambda-1)}])min(|\alpha_v[i]|, |\alpha_v[i + 2^{(\lambda-1)}]|) \quad (1)$$

where $i \in [0 : 2^{(\lambda-1)} - 1]$. The node ν can compute the LLR vector and transfers them to the right branch when the hard decision bits (β_{ν_l}) are received from the left branch.

$$\alpha_{\nu_r}[i] = \alpha_\nu[2i](1 - 2\beta_{\nu_l}[i]) + \alpha_\nu[2i+1] \text{ for } i \in [0 : 2^{(\lambda-1)} - 1]. \quad (2)$$

where α_{ν_r} is the LLR of the right branch. The codeword β_ν can be computed at node ν when the hard decision bits of the right branch are ready.

$$\begin{cases} \beta_\nu[i] = \beta_{\nu_l}[i] \oplus \beta_{\nu_r}[i], \\ \beta_\nu[i + 2^{(\lambda-1)}] = \beta_{\nu_r}[i]. \end{cases} \quad (3)$$

for $i \in [0, 2^{\lambda-1}-1]$. Defining $\mathcal{A}$ and $\mathcal{A}^c$ as sets of information and frozen bits, respectively, the hard decisions (β_ν) in a leaf node can be estimated as

$$\beta_\nu = \begin{cases} h(\alpha_\nu); & \text{if } \nu \in \mathcal{A}, \\ 0; & \text{if } \nu \in \mathcal{A}^c \end{cases} \quad (4)$$

where $h(x)$ is a binary quantizer computed as

$$h(x) = \begin{cases} 0; & \text{if } x \geq 0, \\ 1; & \text{otherwise.} \end{cases} \quad (5)$$

C. SSC AND FAST-SSC DECODING

The SSC [4] and fast-SSC [6] decoders are proposed to address the latency issue associated with SC decoding. Two *Rate*-0 (*R*0) and *Rate*-1 (*R*1) nodes are proposed in the SSC algorithm to eliminate the need for traversing their child nodes. *R*0 is the parent node to a set of all frozen bits. For a *R*0 node located at level λ, it can be decoded by returning a vector of 2^λ zeros. *R*1, on the other hand, is the parent node to a set of information bits and a given *R*1 node located at level λ can get decoded by taking a hard decision on input LLRs. In other words, it can be decoded as

$$\beta_\nu[i] = h(\alpha_\nu[i]) \text{ for } i \in [0, 2^\lambda - 1]. \quad (6)$$

In a fixed-point representation, the *R*1 node can simply get decoded by returning the most significant bit of the soft information.

In fast-SSC algorithm, two new node types are introduced to further decrease the decoding latency. In what follows, these two nodes called repetition (*REP*) and single-parity check (*SPC*) nodes will be described.

- *REP* Nodes: This family contains only one information bit on the rightmost position and the rest of nodes are frozen. For a *REP* node located at level λ, the information bit repeats 2^λ times over the outputs and it can be calculated by threshold detection as

$$\beta_\nu[i] = h(\sum_{i=0}^{2^\lambda-1} \alpha_\nu[i]) \text{ for } i \in [0, 2^\lambda - 1]. \quad (7)$$

- *SPC* Nodes: This family contains only one frozen bit located at the leftmost position. To decode a *SPC* node placed at level λ, the hard decisions need to be computed as $h(\alpha_\nu)$. Now, the parity bit can be computed as

$$\text{parity} = \bigoplus_{i=0}^{2^\lambda-1} h(\alpha_\nu[i]). \quad (8)$$

After calculating the parity bit, it needs to estimate the bit index of the least reliable bit as

$$j = \underset{i}{argmin} \, |\alpha_\nu[i]|. \quad (9)$$

The final step is to calculate the output of the SPC node as

$$\beta_\nu[i] = \begin{cases} h(\alpha_\nu) \oplus \text{parity} & \text{when } i = j \\ h(\alpha_\nu) & \text{otherwise} \end{cases} \quad (10)$$

for $i \in [0, 2^\lambda - 1]$.

In addition to the previously introduced nodes, there are some simplified node mergers that can further reduce the latency. The most practical node mergers are as follows.

- *REPSPC* Merge: This node presented in [6] and is parent to a *REP* and *SPC* node located on the left and right branches, respectively.
- Generalised Repetition (*G-REP*) Merge: This family [8] is a *Rate-R* node where $0 < R < 1$ presented as a scheme to integrate multiple nodes located at multiple levels. Considering t and l_0 as the depth and the lowest level of constituent nodes, respectively, a *G-REP* node located at level L contains a *Rate-C* ($0 < C \leq 1$) node on the rightmost branch at level $l_0 = L\text{-}t$. The rest of child nodes are *R*0.
- Generalized Parity Check (*G-PC*) Merge: Similar to *G-REP*, this family [8] also is a *Rate-R* ($0 < R < 1$) node and it integrates multiple nodes located at multiple levels. A *G-PC* node located at level L contains only one *R*0 node located on the leftmost branch at level $l_0 = L\text{-}t$. The rest of child nodes are *R*1.

It should be mentioned that in the cases of *REPSPC*, *G-REP* and *G-PC* nodes, the decoded bits need to be propagated backward to the root node.

D. CODE CONSTRUCTION BY BINARY-TERNARY MIXED KERNELS

The method for constructing the generator matrix of multi-kernel codes is explained in section II-A. The message passing criterion for a ternary node ν is illustrated in Fig. 1 (b). To pass the massages from a ternary kernel, some new functions need to be defined. Defining (1) as f^b, (2) as g^b, and (3) as C^b, for a ternary node located at level λ in a pure-ternary polar code, the decoding functions for $i \in [0, 3^{\lambda\text{-}1}\text{-}1]$ are:

$$\alpha_{\nu_l}[i] = sgn(\alpha_\nu[i].\alpha_\nu[i + 2^{(\lambda-1)}].\alpha_\nu[i + 2^\lambda]) \; min(|\alpha_\nu[i]|, |\alpha_\nu[i + 2^{(\lambda-1)}]|, |\alpha_\nu[i + 2^\lambda]|). \quad (11)$$

$$\alpha_{\nu_c}[i] = (1\text{-}2\beta_i^l)\alpha_i + f^b(\alpha_{i+2^{(\lambda-1)}} + \alpha_{i+2^\lambda}). \quad (12)$$

$$\alpha_{v_r}[i] = (1\text{-}2\beta_i^l)\alpha_{i+2^{(\lambda-1)}} + (1\text{-}2\beta_i^l \oplus \beta_i^c)\alpha_{i+2^{\lambda}}. \quad (13)$$

$$[\beta_i, \beta_{i+2^{(\lambda-1)}}, \beta_{i+2^{\lambda}}] = [\beta_i^l \oplus \beta_i^c, \beta_i^l \oplus \beta_i^r, \beta_i^l \oplus \beta_i^c \oplus \beta_i^r]. \quad (14)$$

Hereby, we define (11) as f^T, (12) as g_1^T, (13) as g_2^T, and (14) as C^T.

As discussed in [20] and [21], the Kronecker product is not commutative, therefore different ordering of kernels result in different transformation matrices. In other words, the location of information and frozen bits is changed using different kernel orders which directly affects the error-correction performance. Currently there is no theoretical way to identify the order of kernel multiplication. Therefore, simulation with different kernel orders is needed to find the sequence with the best error-correction performance. The method in [21] is used to obtain the kernel orders. As a guideline, we consider LDPC WiMAX code lengths [22] which shows that using only a few non-binary kernels, multi-kernel polar codes can achieve the most desired block lengths. Although the latency value of various multi-kernel codes will be reported in Section V, throughout this work we only provide an in-depth analysis of codes with one ternary kernel which results in kernel sequences below.

- $\mathcal{P}(48, 24)$ with $G = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2$
- $\mathcal{P}(96, 48)$ with $G = T_2 \otimes T_2 \otimes T_2 \otimes T_3 \otimes T_2 \otimes T_2$
- $\mathcal{P}(192, 96)$ with $G = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2$
- $\mathcal{P}(384, 192)$ with $G = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2$
- $\mathcal{P}(768, 384)$ with $G = T_2 \otimes T_2 \otimes T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2$
- $\mathcal{P}(1536, 768)$ with $G = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_2$

E. MULTI-KERNEL VERSUS PUNCTURING AND SHORTENING METHODS

Fig. 2 illustrates the error-correction performance of multi-kernel [20], puncturing [15] and shortening [16] methods over an additive white Gaussian noise (AWGN) channel using SC and SCL with a list size of $L = 8$. Two different block lengths of $N = 48$ and $N = 72$ ($G = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_3$) with rate $\mathcal{R} = 1/2$ are selected in the multi-kernel scenario. For the punctured and shortend polar codes, the mother polar code of length $N' = 64$ (for the case of $N = 48$) and $N' = 128$ (for the case of $N = 72$) are used. Obviously, multi-kernel decoding considerably outperforms punctured and shortened methods.

In terms of complexity, multi-kernel decoding offers lower decoding complexity with respect to puncturing and shortening methods since smaller Tanner graphs are used in their code construction. The punctured and shortend polar codes are constructed from a mother polar code of length $N' = 2^{\lceil log_2 N \rceil}$ and the mother code determines the code's complexity. A metric that can be used to evaluate the complexity is the overall number of the LLRs need to be calculated in decoding process of different schemes. With s being the number of stages in the code's Tanner graph (identical to the number of kernels used in the code construction), $N \times s$ and $N' log_2 N'$ LLRs need to be computed to decode an entire codeword in cases of multi-kernel and puncturing/shortening methods, respectively. Therefore, for a polar code of length $N = 48$ (72), 240 (360) LLRs needs to be calculated in case of multi-kernel codes. On the other hand, 384 (896) LLRs computation is needed for punctured and shortened polar codes. Obviously, 37.5% (59.8%) lower LLRs need to be calculated using multi-kernel codes which shows a substantial reduction in complexity.

F. FAST MULTI-KERNEL DECODING

Fast-SSC decoding of multi-kernel polar codes is investigated in [23]. It is proved in [23] that $R0$, $R1$ and SPC ($R = \frac{N-1}{N}$) nodes for ternary kernel can be computed using the same method as for binary kernel. The mixed repetition node, however, has a different decoding rule and it is categorized into three groups for the ternary kernel. In this paper, we refer to this group as REP^T. More detail on decoding steps of each node is available in [23].

III. MULTI-KERNEL DECODING ALGORITHM

In this section, the algorithm that supports multi-code decoding of polar codes will be presented. The proposed algorithm supports purely binary, purely ternary and binary-ternary mixed decoding of polar codes. In the case of mixed kernel polar codes, any order of the kernels can be considered and there is no need for the decoder to have any prior knowledge of the code structure. The goal of the algorithm is to decrease the decoding latency of the polar codes. Therefore, the prevailing patterns in short to long block lengths are identified and corresponding specialized decoding algorithm is presented. These patterns are given in five groups where they eliminate the need for partial sequential decoding. The hardware architecture and FPGA implementation of the algorithm will be detailed in the following section.

In this section, the depth is calculated as $t = L\text{-}l_0$ where L and l_0 are the location of the parent node and the lowest level of the leaves, respectively. The five groups of high-level node mergers for multi-kernel decoder are as follows.

- Group A Patterns: The $R0SPC$ node is first identified in [9] where it merges two $R0$ and SPC nodes located at the same level. To generalize this idea, this group integrates nodes from multiple levels of the binary tree where t $R0$ nodes are located on the left branches of a *Rate-R* ($0 < R < 1$) node. We categorize the subtrees into three different *Rate-R* patterns. Two $R0^t SPC$ and $R0^{t\text{-}1} REPSPC$ are proposed in [24] and another member is introduced as $R0^t R1$ in [19] shown as "Group A" in Fig. 3. Table 1 tabulates the count of appearances of the proposed node mergers in the Tanner graph of polar codes with various block lengths after pruning the tree. A decoder for this group will be provided in the following section.

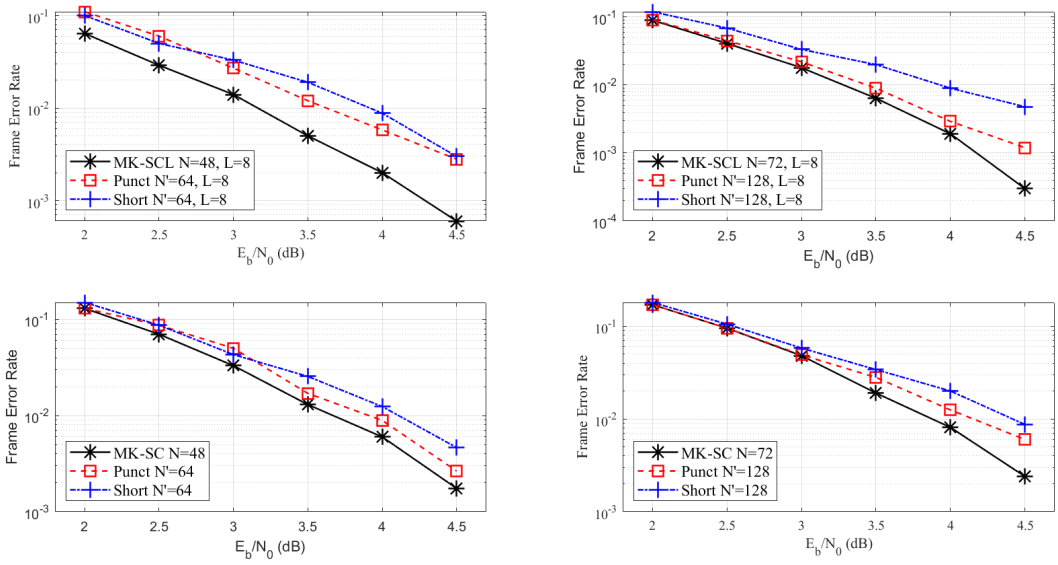

FIGURE 2. The error correction performance of multi-kernel polar codes with respect to puncturing and shortening methods.

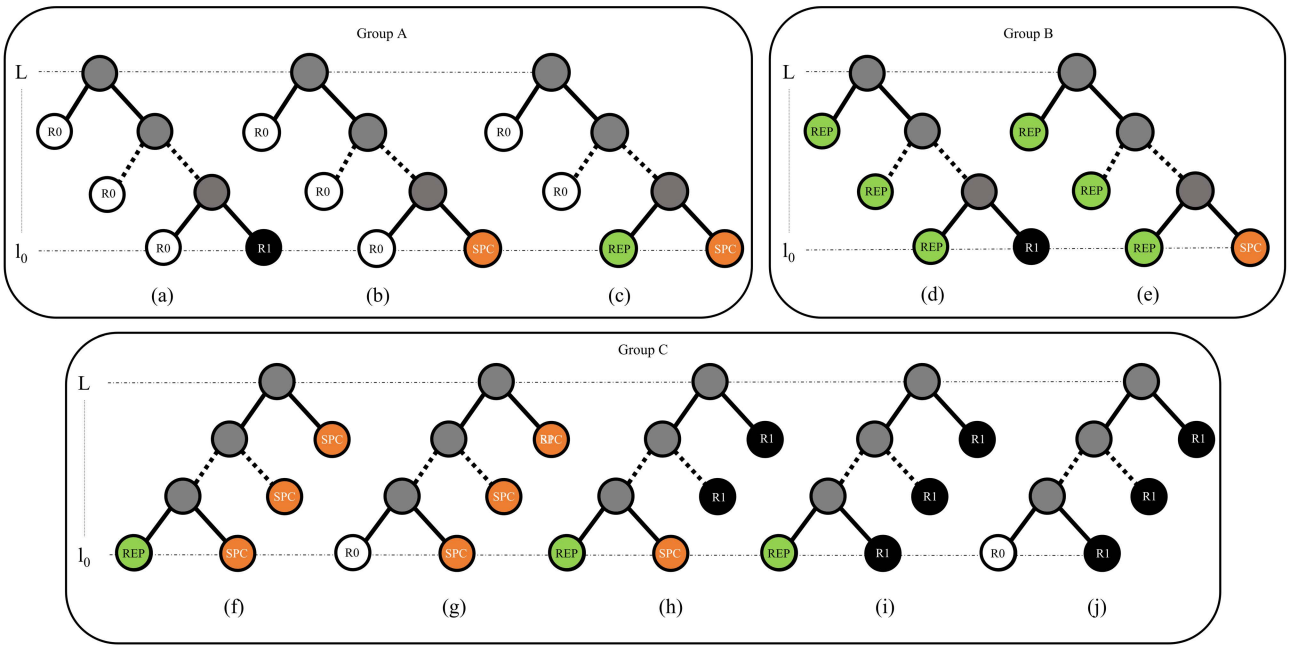

FIGURE 3. Tree illustration of the most frequent node patterns.

- Group B Patterns: The *REPR*1 and *REPSPC* nodes are primarily proposed in [8] and [6], respectively, where they merge a *REP* node by either *R*1 or *SPC* nodes. We generalize these nodes using t *REP* nodes from multiple levels. Thus, this group categorizes them into REP^tR1 and REP^tSPC, respectively, displayed as "Group B" in Fig. 3. A general decoding algorithm for this group is available in [24]. The REP^tSPC pattern can be decoded faster by the following algorithm. It is assumed that the information bit integrated at a *REP* node at level l is q_l. The first step for decoding is to calculate the information bit at each *REP* node at level l in parallel as

$$q_l = h\Big(\sum_{i=0}^{2^l-1}\sum_{k=0}^{2^t-1} \alpha_{2^t i+k} \boxplus \alpha_{2^t i+k+2^{t-1}}\Big). \tag{15}$$

After decoding t *REP* nodes in parallel, the decoded information bits need to be encoded again before proceeding to the *SPC* node. The first group of information bits to be encoded is a concatenation of t *REP* nodes with a different number of leaves from 2^{t-1} to 1 where

TABLE 1. Count of the proposed node mergers of groups A-C in polar codes of different length with rate $R = \frac{1}{2}$.

Block Length	k	Group A	Group B	Group C
512	256	3	2	5
1024	512	3	5	8
2048	1024	7	6	13
4096	2048	14	8	22
8192	4096	17	18	35
16384	8192	36	22	58
32768	16384	58	39	97

TABLE 2. Details of new functions supported by the proposed decoder.

Function	Description
$C^{b^t}/C_0^{b^t}$	Up to t consecutive C^b/C_0^b operations.
f^{b^t}	Up to t consecutive f^b operations.
$g^b f^b$	g^b instruction followed by f^b.
C^b/C_0^b-g^b	C^b/C_0^b operation followed by g^b.
$g_0^{b^t}$	Up to t consecutive g_0^b operations.
$f^b g_0^b$	f^b instruction followed by g_0^b.

the single leaf *REP* node corresponds to q_L. The order of nodes starts from the lowest level to the highest level node and the last bit is set as 0. For example the sequence to be encoded for $t = 3$ is

$$q = \{0, 0, 0, q_{l_0}, 0, q_{l_0+1}, q_{l_0+2}, 0\}. \tag{16}$$

This stream can be encoded by a polar code generator matrix of size $2^{t\text{-}1}$ as

$$a = qG. \tag{17}$$

Now the encoded *SPC* bits of the *SPC* node at l_0 level can be directly calculated as

$$\beta_i = h(\sum_{k=0}^{2^t-1}(1 - 2a_k)\alpha_{2^t i+k}) \text{ for } i \in \{0, 2^{l_0} - 1\}. \tag{18}$$

Since the output of the merged node is located at level L, the encoded a bits are added to each β_i as $a \oplus \beta_i$ for $i \in \{0, 2^{l_0} - 1\}$.

REP^t-$R1$ can also get decoded using the same procedure as $REP^t SPC$ by substituting the *SPC* node with *R1* node.

- Group C Patterns: This group also integrates nodes from multiple levels of the binary tree and generalizes four different patterns (*REPSPC*, *REPR1*, *R0SPC* and *R0R1* [6]). Three mergers of this this group, $REPSPC^t$, $REPSPCR1^{t\text{-}1}$ and $REPR1^t$, are presented in [24]. In [19], we added two $R0SPC^t$ and $R0R1^t$ to this group as new members. This family is shown under "Group C" in Fig. 3. The decoding procedure of $REPSPC^t$ merger can be made faster using the following algorithm. First, the *REP* node can get decoded by

$$q_L = h(\sum_{i=0}^{2^{l_0}-1}\sum_{j=0}^{2^t-1}\boxplus\alpha_{2^t i+j}). \tag{19}$$

Now we can directly calculate partial sum bits in parallel at level L as below.

$$\beta_{2^t i+k} = h(\alpha_{2^t i+k} + \sum_{j=0/k}^{2^t-1}\boxplus\alpha_{2^t i+j}) \tag{20}$$

for $i \in \{0, 2^{l_0} - 1\}$ and $k \in \{0, 2^t - 1\}$. As the final step, it only needs to perform a parity check for each i as

$$\sum_{k=0}^{2^{l_0}-1}\beta_{2^t i+k} + q_L = 0. \tag{21}$$

Like the processing of the *SPC* node, the partial sum bit with the least reliable LLR must be flipped in case the parity check is not fulfilled.

The $R0SPC^t$, $REPSPCR1^{t\text{-}1}$, $REPR1^t$ and $R0R1^t$ can be decoded using the same algorithm as $REPSPC^t$. The only difference is that in the cases of $REPR1^t$ and $R0R1^t$, there is no need for the final parity check step. It should be noted that $REPSPC^t$, $R0SPC^t$ and $REPSPCR1^{t\text{-}1}$ mergers degrade the error correcting capability of the decoder by a small margin. The effect of this algorithm on error-correction performance will be investigated in Section V-B.

- Group D Patterns: By introducing groups A-C, there is an opportunity to merge other functions such as f^b, g^b, and C^b. These types of functions have no effect on the overall critical path since they introduce significantly lower delays compared to leaf nodes. This group is summarized in Table 2.

  Let β_{v_l} and β_{v_r} be the codeword estimates coming from the left and right branches of node v, respectively. The C^b/C_0^b operations combine β_{v_l} and β_{v_r} using (3) to estimate the codeword of node v. In case of C_0^b operation, β_{v_l} is a vector of zeros. The C^b/C_0^b operations constitute a large portion of instructions in SC-based decoders. For instance, it counts for 26% of overall instructions in a $\mathcal{P}(1024, 512)$ under our proposed algorithm. The simulation results reveal that 92% of combine operations are consecutive. We generalize the consecutive combine operation as $C^{b^t}/C_0^{b^t}$ where it merges t consecutive combine operations. Similar consecutive node processing is also possible for f^b operation using (1). Based on our simulations, f^b counts for 23% of overall instructions under our proposed algorithm in a $\mathcal{P}(1024, 512)$ in which 71% of f^b operations are consecutive. We generalized this operation as f^{b^t}.

  There are three functions proposed in [13] that are added to this group. $g^b f^b$ function which calculates g^b followed by f^b, C^b/C_0^b-g^b that calculates a C^b/C_0^b operation followed by a g^b operation, and $f^b g_0^b$ which calculates f^b followed by g_0^b. Finally, we generalized the last member of this group as $g_0^{b^t}$ which calculate t consecutive g_0^b operations using (2) with β_{v_l} equals zero.
- Group E Patterns: As mentioned in Section II-D, ternary fast-SSC algorithm is investigated in [23]. However, they have considered some constraints on each group

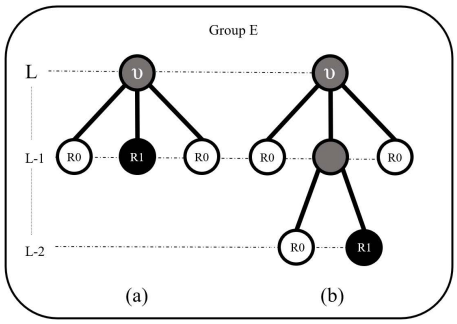

FIGURE 4. Tree representation of the most frequent node patterns in mixed kernel polar codes.

of REP^T that limits the appearance of repetition nodes in the polar tree. We implement a generalized form of repetition nodes that can be computed instead of stored which is the case in [23].

Here, we also introduce two node mergers called $R0R1R0$ (Fig. 4 (a)) and $R0^2R1R0$ (Fig. 4 (b)), that frequently appear in the polar codes including ternary kernels in their kernel sequence. The decoding procedure of $R0R1R0$ can be made faster by the following algorithm. First, the $R1$ node will be decoded by

$$\beta_{vc}[i] = h(\alpha_v[i] + min(\alpha_v[2^{L-1}+i], \alpha_v[2^L+i])) \quad \text{for } i \in \{0, 2^{L-1}-1\}. \tag{22}$$

Now partial sum bits can be calculated in parallel at level L as given below.

$$\begin{cases} \beta_v[0:2^{L-1}-1] = \beta_{vc}, \\ \beta_v[2^{L-1}:2^L-1] = 0, \\ \beta_v[2^L:2^L+2^{L-1}-1] = \beta_{vc}. \end{cases} \tag{23}$$

The $R0^2R1R0$ node can also be decoded faster using the algorithm below. The $R1$ node can be decoded as

$$\begin{aligned} \beta_{R_1}[i] = h(&\alpha_v[i] + \alpha_v[2^{L-2}+i] \\ &+min(\alpha_v[2^{L-1}+i], \alpha_v[2^L+i]) \\ &+min(\alpha_v[2^{L-1}+2^{L-2}+i], \\ &\times \alpha_v[2^L+2^{L-2}+i])) \\ &\text{for } i \in \{0, 2^{L-2}-1\}. \end{aligned} \tag{24}$$

The partial sum bits at level L can be calculated as

$$\begin{cases} \beta_v[0:2^{L-1}-1] = \beta_{R_1} + \beta_{R_1}, \\ \beta_v[2^{L-1}:2^L-1] = 0, \\ \beta_v[2^L:2^L+2^{L-1}-1] = \beta_{R_1} + \beta_{R_1}. \end{cases} \tag{25}$$

IV. HARDWARE IMPLEMENTATION

This section summarizes the hardware implementation aspects of the proposed algorithm. The overall architecture is designed based on the conventional fast-SSC architecture for polar codes presented in [6]. The datapath architecture and efficient hardware architectures for each node merger are detailed in this section.

To find the location of the node mergers in the polar tree, the decoder first needs to calculate the position of the information and frozen bits. Then the location of *REP* and *SPC* nodes and on top of those the location of multi-level node mergers need to be calculated. We have developed a software program to calculate the location of these nodes and it generated an output used to configure the implemented decoder. In the following, the overall architecture, memory requirement and the functional blocks will be presented.

TABLE 3. Details of functions supported by the proposed decoder.

Function	Description
f^b, f^{b^2}, f^{b^3}	calculates α_l using (1).
f^T	calculates α_l in ternary kernels using (11).
$g^b, g_0^b, g_0^{b^2}$	calculates α_r using (2).
$f^b g_0^b$	f^b followed by g_0^b using (1) and (2).
$g^b f^b$	g^b followed by f^b using (2) and (1).
g_1^T	calculates α_c in ternary kernels using (12).
g_2^T	calculates α_r in ternary kernels using (13).
C^b, C^{b^2}, C^{b^3}	combines β_l and β_r using (3).
$C_0^b, C_0^{b^2}, C_0^{b^3}$	combines β_l and β_r using (3).
$C^b g^b, C_0^b g^b$	calculates α_r using (3) followed by (2).
C^T	combines β_v in ternary kernel nodes by (14).
$R1$	calculates β_v by taking a hard decision.
REP^b, REP^T	calculates β_v for repetition nodes.
SPC^b, SPC^T	calculates β_v for SPC nodes.
$REPSPC$	calculates β_v for $REPSPC$.
$R0^t R1$	calculates β_v for $R0^t R1$.
$R0^t SPC$	calculates β_v for $R0^t SPC$.
$R0^{t-1} REPSPC$	calculates β_v for $R0^{t-1} REPSPC$.
REP^t-$R1/SPC$	calculates β_v for all nodes of Group B.
$REP/R0$-$R1^t/SPC^T$	calculates β_v for all nodes of Group C.
$R0R1R0$	calculates β_v using (22) and (23).
$R0^2R1R0$	calculates β_v using (24) and (25).

A. DECODER ARCHITECTURE

The overall architecture for conventional fast-SSC is detailed in [6]. Table 3 outlines the operations supported by the proposed datapath. The calculation of the function assignment list is offline and a new list of functions can be transferred to the decoder upon requirement. Each instruction word is 6 bits long. The instructions directly point to the functions, and size of the functions can be calculated by the depth where the node is located at.

Different memory units are used for channel LLR values, intrinsic LLR values α, the partial sum β, decoding instructions, and final codeword. First, the instructions are loaded into the instruction RAM to be read and fetched by the controller (instruction decoder). The controller then triggers the channel loader and the processing unit (ALU) to store the channel LLRs into the channel RAM and perform the correct function, respectively. The ALU, where the functions listed in Table 3 are performed, can read/write data from/to the α-RAM and β-RAM. The data stored into β-RAM is the estimated codeword and is accessible from outside the decoder.

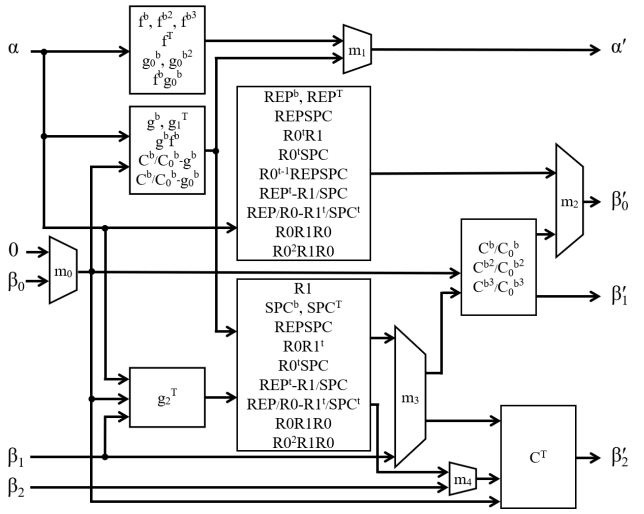

FIGURE 5. Proposed datapath architecture supporting all proposed nodes.

1) DATAPATH ARCHITECTURE

The core part of decoder is the datapath, depicted in Fig. 5, where all functions presented in Table 3 are implemented. Resource-sharing plus multiplexing is used to decrease the complexity of the circuit. For instance, all members of each Group B and C in Fig. 3 are implemented by a single specified decoder. The datapath includes four inputs α, β_0, β_1 and β_2 which generate four corresponding outputs α', β'_0, β'_1 and β'_2. The m_0 multiplexer selects either a vector of zeros or the decoded stream coming from the left branch. A multiplexer (m_1) chooses among the output of the functions that generate soft outputs as the output of the current stage (α'). The m_2 multiplexer picks the correct function that generates β'_0. Eventually, m_3 and m_4 select the correct inputs to the combine blocks which is responsible to combine β_l and β_r in binary nodes and β_l, β_c and β_r in ternary nodes. The original critical path goes through g_2^T-*SPC*-*Combine* path.

We define the execution of an instruction as an step which may consume one or several clock cycles. It directly depends on the allocation of the physical processing elements ($P_e = 2P$) to execute the task. The choice of appropriate P_e for the decoder is critical since too small and too large P_e values cause high decoding latency and inefficient resource utilization, respectively.

The utilization rate for a semi-parallel decoder (α_{sp}) is shown in [25] to be calculated as

$$\alpha_{sp} = \frac{log_2 N}{4P_e + log_2(\frac{N}{4P_e})}. \tag{26}$$

Using (26) in [25], it is shown that even for a small value of P_e, the maximum throughput can be achieved for long block lengths. Our simulations show that $P_e = 120$ and $P_e = 240$ are acceptable choices for a decoder with maximum code length of $N = 32768$.

2) MEMORY

In the proposed decoder architecture, the required input/output buffers including those buffers consumed for storing internal values are taken into account. To attain the highest achievable throughput, the decoder needs to load the next α values from the channel while decoding the one previously stored in the memory. To this end, two separate memories are used to store the channel and internal α values. The memory that stores β values also benefits from the same architecture. In what follows, more details will be presented regarding these memories and data routing.

- Channel α Values: The stored values in the channel RAM are transferred to the decoder in groups of 32 LLRs. Thus, for the largest code supported by the decoder in this paper ($N = 32768$), 1024 clock cycles are required to transfer a frame. In order to prevent the throughput loss by the channel α memory, it is vital to transmit a new frame to the α memory while the decoder is decoding another frame. Therefore, α memory is divided into two separate banks each P_e LLRs wide. This way, the decoder's input bus width can be selected to $32 \times Q_c$ bits where Q_c is the number of channel quantization bits. The decoder needs $P_e = 2P$ channel LLRs at a given clock cycle to operate; thus, it needs to access a bus with $2P \times Q_c$ bits wide. To keep the width of read and write buses equal, multiple RAM banks ($P_e/32$) are used, each of which has the depth of $\lceil \frac{N_{max}}{P_e} \rceil$ and width of $32 \times Q_c$ bits. It should be noted that data is written only to one bank at a time; however, it is read from all banks at the same time.
- Internal α Values: The f and g functions and their variants generate α values. Each function accepts two (in case of binary nodes) or three (in case of ternary nodes) α values to operate and generates an α value at a given clock cycle. After calculating β values at a given stage, the corresponding α values are no longer needed and they can be overwritten to save RAM. The parallelism level is limited to $2P$ α inputs. Two separate ports are used for read and write operations. Similar to [18], four banks with two different widths are used depending on the stage being binary or ternary.
- Internal β Values: This memory is composed of three dual-port RAMs with $2P$ width that stores internal β values. Each RAM has the duty of storing the decoded stream of left, middle or right children in ternary nodes. In case of binary nodes, the internal left and right β values are stored in the left and middle banks, respectively.
- Estimated Codeword: This memory is used to transfer the estimated codeword to the output environment, and it is separated from β-RAM to support full speed decoding. In order to keep the codeword bus narrower than $2P$, the estimated codeword is stored into this memory when $2P$ bits are generated. This way, the decoder is able to start new decoding right after the previous one is decoded.

- α Router: This router is proposed in [18] and it is used to choose the part of the memorized word that needs to be overwritten during a write operation.
- β Router: This router reads/writes data from/to the internal β-RAM. Reading operation involves P or $P^{b/t}$ bits per bank in the binary or ternary cases, respectively, while each word contains $2P = 3P^{b/t}$ bits. When writing, the input data is either selected from the combine block or the hard decision coming from the leaves.

B. IMPLEMENTED FUNCTIONS

In this section, a specified architecture for crucial functional blocks used in hardware implementation of the multi-kernel polar codes will be described.

1) $R1$ FUNCTION

Up to P $R1$ functions can get decoded at the same time by taking a hard decision on LLRs (returning the sign of LLRs in two's complement format) with no latency overhead.

2) SPC FUNCTION

The SPC block is the most complex block in this group. The core part of SPC is a compare-select (CS) block which is responsible for finding the index of the least reliable input bit. It is shown in [6] that we can decode SPC block of length $N_{v_{SPC}} \leq 8$ in only one clock cycle. However, for SPC blocks with $N_{v_{SPC}} > 8$ pipeline stages with optimized depth is required. The maximum length of constituent nodes for SPC blocks embedded in all node mergers is selected to $N_{v_{SPC}} = 8$ in order to calculate the result in the same clock cycle the inputs are fed. However, in other branches where the SPC nodes appear in the tree, the maximum length is selected to the maximum possible ($N_{v_{SPC}} = P$), and optimized pipeline stages are inserted in order to increase the performance. We use the notation of SPC^b and SPC^T in cases of binary and ternary kernels.

3) REP FUNCTION

In the binary case, the REP^b block with length $N_{v_{REP^b}}$ can be decoded by accumulating all input LLRs and concatenating the sign of this summation $N_{v_{REP^b}}$ times. In this paper, we assumed that the maximum length of constituent nodes for REP^b block is $N_{v_{REP^b}} = P$. In the ternary case, the repetition node with rate $R = \frac{1}{N}$ can be decoded by taking a hard decision on the accumulation of all the LLRs whose indices appear in the repetition pattern of the parent node. The patterns proposed in [23] are used as REP^T group. The repetition blocks are implemented using purely combinational logic, and they can provide their output in the same clock cycle as inputs arrive.

4) $REPSPC$ FUNCTION

This block implements the $REPSPC$ [6] function and its architecture is depicted in Fig. 6. The length of REP and SPC blocks is limited to $N_{v_{REP}} = N_{v_{SPC}} = 8$. Therefore, the overall length of this block is $N_v = 16$. A REP and two SPC blocks are needed in this architecture. First, an f^b function calculates the vector of α_{REP} in order to feed it to the REP block. Then, two g^b blocks calculate the α_{SPC_0} and α_{SPC_1}, one assuming that the output of the REP block is all zero and the other all ones. These values will be fed into SPC_0 and SPC_1 blocks.

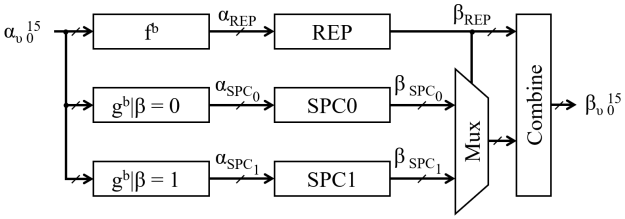

FIGURE 6. Architecture of *REPSPC* decoder [6].

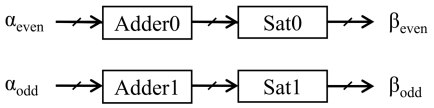

FIGURE 7. Architecture of $R0^3R1$ decoder.

A multiplexer selects the correct output out of two possible SPC outputs, i.e β_{SPC_0} and β_{SPC_1} in Fig. 6. The output of the REP block (β_{REP}) selects the correct SPC output. Finally, a combine block calculates the overall output (β_v) using β_{REP} and either β_{SPC_0} or β_{SPC_1}. It is worth noting that this block is also implemented as a purely combinational block, and it generates an output in the same clock cycle as the inputs are fed.

5) $R0^tR1$ FUNCTION

This block implements $R0^tR1$ function. The depth and overall length of this function are selected as $t = 3$ and $N_v = 16$, respectively. Thus, the $R1$ node is composed of two child nodes ($\{\beta_1\beta_0\}$) at the rightmost position resulting in repeating these two bits $N_v/2$ times at level L. The bit positions at level L will be $\{\beta_1, \beta_0, \beta_1, \beta_0, \ldots, \beta_1, \beta_0, \beta_1, \beta_0\}$. Assuming that the input LLR indexes stand as $\{\alpha_0, \alpha_1, \ldots \alpha_{15}\}$, β_0 and β_1 can be calculated by returning the sign of separate summation of all odd and even indexed input LLRs. The block diagram of $R0^3R1$ is illustrated in Fig. 7. Two adders accumulate all odd and even indexed LLRs separately, and a sign detector blocks follow the adders to generate the decoded output. It should be noted that there is no need for a saturation check after addition in this function since only the sign of the addition is needed. As it can be seen from the figure, this block is implemented using purely combinational logic resulting in the capability of providing an output in one clock cycle.

6) $R0^tSPC$ AND $R0^{t-1}REPSPC$ FUNCTIONS

These blocks decode $R0^tSPC$ and $R0^{t-1}REPSPC$ functions. In $R0^tSPC$, the depth is selected as $t = 2$ limiting the

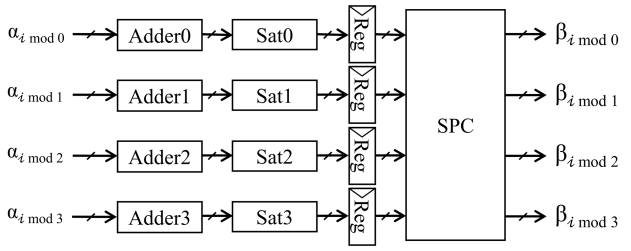

FIGURE 8. Architecture of $R0^2SPC$ decoder.

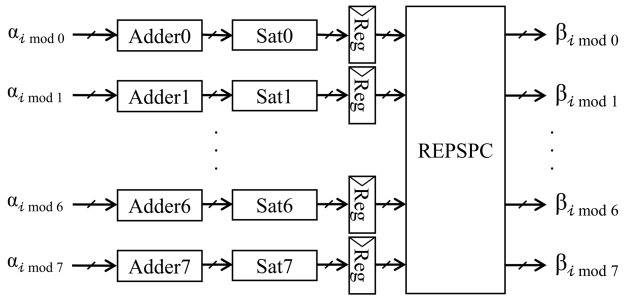

FIGURE 9. Architecture of $R0^2REPSPC$ decoder.

overall length to $N_v = 16$. The rightmost *SPC* node is replicated N_v/N_{SPC} times at level L where $N_{SPC} = 4$. The core part of this decoder is a *SPC* block where it can be fed by accumulation of the LLR indices modulo-4. Fig. 8 illustrates the architecture of $R0^2SPC$. It requires $N_{SPC} = 4$ adders to perform the summation of LLR indices modulo-4 followed by saturation check blocks to carry out saturation check afterwards. A pipeline stage is needed to avoid long critical path. The registers store the output of the saturation blocks and they feed the *SPC* block to calculate the final output. The output codeword can be computed by repeating the output codeword of the *SPC* block four times. The pipeline stage adds an extra step to the overall latency.

The $R0^{t\text{-}1}REPSPC$ can also be decoded by following the same steps as $R0^tSPC$. The depth of this block is selected as $t = 3$ limiting the overall length to $N_v = 32$. The rightmost node *REPSPC* is replicated $N_v/N_{REPSPC} = 4$ times at level L where $N_{REPSPC} = 8$. The key part of this block is a *REPSPC* block where it can be fed by accumulation of LLR indices modulo-8. Fig. 9 depicts the architecture of $R0^2REPSPC$ where it demands $N_{REPSPC} = 8$ adders to perform the summation of the LLR indices modulo-8 followed by saturation check blocks. A pipeline stage is also employed to avoid long datapath latency by storing the output of the saturation blocks. Finally, the register outputs are fed into *REPSPC* block to generate the final output. The output codeword can be calculated by repeating the output codeword of the *REPSPC* block four times. Since a pipeline stage is used, this block adds an additional step to the overall latency.

7) REP^t-R1/SPC FUNCTION

This block decodes node mergers represented as "Group B" in Fig. 3. A generic algorithm for decoding this block is presented in the previous section. However, our simulations reveal that it demands a noticeable amount of hardware resources. In what follows, we use a more reliable and resource-efficient architecture. By deploying resource sharing, all patterns of "Group B" can be decoded by the same block.

Assume that a REP^tR1 or a REP^tSPC node is located at level L with depth $t = 2$. The architecture of the decoder is illustrated in Fig. 10 where it needs three *REP* and four *SPC*/*Sign* blocks to be implemented. The length of REP_{L-1} block located at level $L-1$ is limited to $N_{v_{REP_{L-1}}} = 8$, and that of REP_{L-2} and *SPC*/*Sign* blocks located at level $L-2$ is selected to $N_v = 4$. This limits the overall length of this block to $N_v = 16$.

To decode this block, first f^b block calculates the vector of LLRs ($\alpha_{REP_{L-1}}$) to feed it to the REP_{L-1} block. Second, two g^b blocks compute the α_{v_0} and α_{v_1}, one assuming the output of the REP_{L-1} block is all zeros and the other all ones. Now, for each upper and lower parts of the circuit, an f^b block calculates the vector of $\alpha_{REP_{L-2}}$ in order to feed it to the REP^0_{L-2} and REP^1_{L-2} blocks. In the upper half of the architecture, two g^b blocks calculate the $\alpha_{v_{00}}$ and $\alpha_{v_{01}}$, one assuming that the output of the REP^0_{L-2} block is all zeros and the other all ones. $\alpha_{v_{10}}$ and $\alpha_{v_{11}}$ will be generated by following exactly the same procedure in the lower half. A control flag will select the type of the leaf node in *SPC*/*Sign* block. This block will decode *SPC* or *R*1 blocks when the control flag is 0 or 1, respectively.

A multiplexer chooses the correct output out of two possible inputs for *SPC*/*Sign* blocks for each upper and lower parts of the circuit. The output of the REP_{L-2} block ($\beta'_{REP_{L-2}}$) selects the correct *SPC*/*Sign* output of each part. Now, a combine block can calculate the overall output of each part, i.e. β_{v_0} and β_{v_1}. β_{v_0} can be calculated using $\beta'_{REP^0_{L-2}}$ as the control signal and either $\beta_{v_{00}}$ or $\beta_{v_{01}}$ as the data. Similarly, β_{v_1} can be calculated using $\beta'_{REP^1_{L-2}}$ as the control signal and either $\beta_{v_{10}}$ or $\beta_{v_{11}}$ as the data. Finally, another multiplexer chooses the final output out of two possible inputs, i.e. β_{v_0} and β_{v_1}. The output of the REP_{L-1} block ($\beta_{REP_{L-1}}$) selects the correct output. The final block combines $\beta_{REP_{L-1}}$ with either β_{v_0} or β_{v_1} to calculate the overall output (β_v). This block adds no extra decoding step to the overall latency.

The proposed architecture is scalable with the capability of extending to nodes with higher depths at the cost of resource consumption. It also does not affect the overall error correction performance of the decoder.

8) $REP/R0 - R1^t/SPC^T$ FUNCTION

This block implements a decoder for all patterns of "Group C" in Fig. 3. The advantage of this decoder is that it is able to decode five different node patterns of

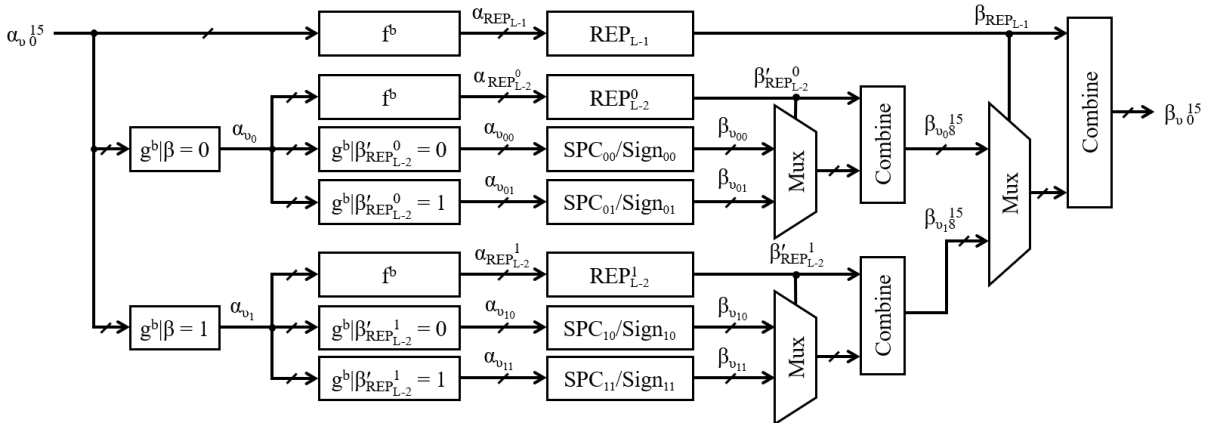

FIGURE 10. Architecture of the *REP*^{2}-*R*1/*SPC* decoder.

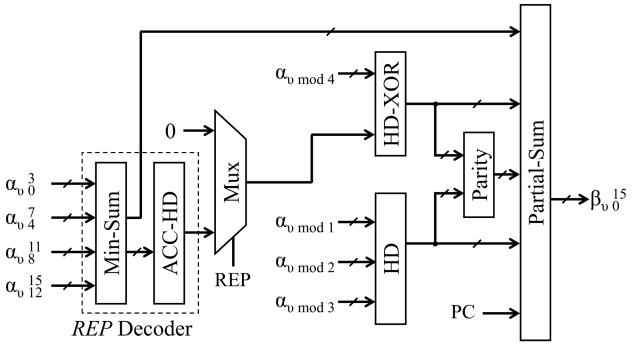

FIGURE 11. Architecture of *REP*/*R*0 – *R*1^2/*SPC*^{2} decoder.

"Group C" using some control signals. The depth of this block is selected as $t = 2$ which limits the block length to $N_v = 16$.

The proposed algorithm in [19] is directly implemented to decode this function. The architecture of the decoder is illustrated in Fig. 11. Depending on the leftmost node, the multiplexer chooses either a vector of zeros or the output coming from the *REP* decoder which implements equation (19). The *REP* decoder is implemented by a Min-Sum block followed by an accumulation-hard decision block (ACC-HD in Fig. 11). The REP flag determines if the leftmost node is a *R*0 or a *REP* node. The HD-XOR block computes the hard decision of $\alpha_{v \mod 4}$ and XORs them by a vector generated by replication of the multiplexer's output four times. The HD block computes the hard decision of the rest of the soft inputs.

Now, four parity check bits can be computed over the bit indices modulo-4. The partial sum bit with least LLR value will be flipped in case the parity check is not passed. The Min-Sum block calculates the indices and transfers them to Partial-Sum block. The Partial-Sum block computes the final output using the LLR indices, hard decisions and parity check (PC) flag which determines if a parity check is required. This function adds no additional steps to the overall latency.

V. FPGA IMPLEMENTATION AND PERFORMANCE ANALYSIS

A. VERIFICATION METHODOLOGY

All polar codes of this section are constructed to be optimal for $E_b/N_o = 2.5$ dB similar to [13]. VHDL coding in Xilinx Vivado 2019.1 environment is used to validate the constructed codes and Logic synthesis, technology mapping, and place and route are conducted targeting a Xilinx FPGA. A software program generates random codewords using binary phase-shift keying (BPSK) modulation over an AWGN channel. As mentioned previously, to protect the decoder from stalling, a new frame is transmitted to the decoder while it decodes another one. The test setup is designed carefully to avoid slowing the decoder down by the interface.

B. EFFECT OF QUANTIZATION ON PERFORMANCE

In order to implement the algorithm on hardware, we need to quantize the LLRs. The quantization scheme is selected as $Q(Q_i, Q_c, Q_f)$ where Q_i, Q_c, and Q_f are the number of LLR quantization bits for internal, channel and fraction bit sizes, respectively. Fig. 12 (a) illustrates that using (5, 4) quantization paradigm for a polar code of $\mathcal{P}(1024, 512)$ with $\mathcal{R} = 1/2$, the performance is very close to that of the floating-point scheme.

We consider two lossy and lossless implementations. As mentioned in section III, the only group that affects error-correction performance is Group C which will be excluded in lossless implementation. Fig. 12 (b) illustrates the error-correction performance of fast-SSC versus lossy and lossless versions of the algorithm for the same polar code as Fig. 12 (a). It can be observed that compared to fast-SSC, the lossless algorithm adds no error-correction loss to the performance. The lossy algorithm, on the other hand, has some error-correction performance overhead, as shown in the figure. Depending on the application, the lossless or lossy versions can be employed.

The error-correction performance of binary-ternary mixed polar codes is simulated through a binary-input AWGN

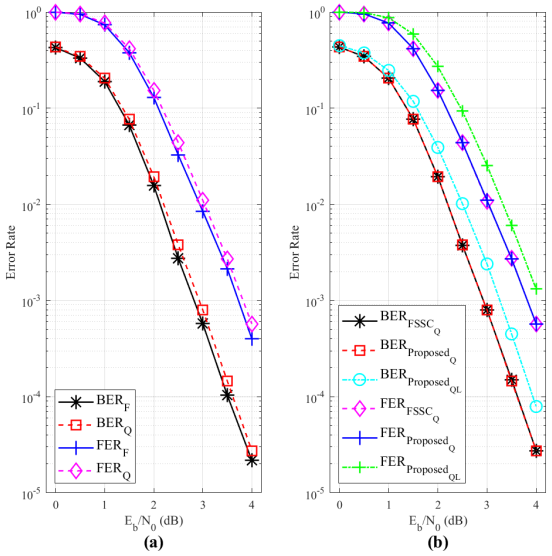

FIGURE 12. Error-correction performance of (a) lossless floating-point versus quantized and, (b) Lossy versus lossless nodes for a polar code of length $N = 1024$ and rate $R = \frac{1}{2}$ under the proposed algorithm.

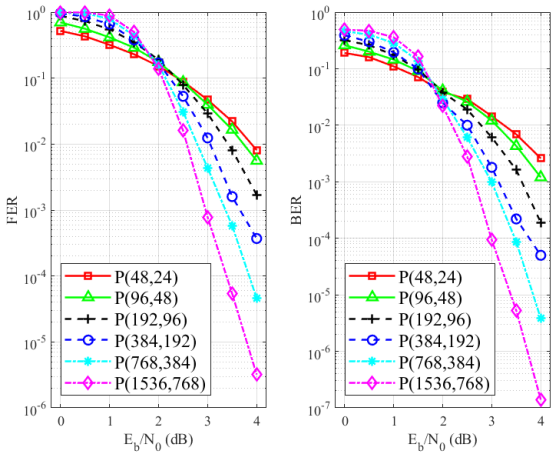

FIGURE 13. Error-correction performance of binary-ternary mixed polar codes.

channel using BPSK modulation. Fig. 13 depicts the error-correction performance of multi-kernel codes under the proposed algorithm using fixed-point format.

C. EFFECT OF BLOCKLENGTH AND CODE RATE ON THE LATENCY

In this section, the latency behavior using all functions described in section III for both binary and mixed-kernel cases will be presented. Table 4 tabulates the effect of code length on decoding latency of fast-SSC versus the proposed algorithm in the binary case. The code rate is considered as $\mathcal{R} = 1/2$ for all block lengths under two different P_e values of size 120 and 240. It can be seen that comparing [19] to fast-SSC, the latency improvement lowers with block length growth. For instance, with $P_e = 128$ the latency improvement is 37.4% and 13.8% for block lengths of 512 and 32768, respectively. This is due to the fact that [19] targeted short packet polar codes. For higher code lengths, a larger part of the latency stems from propagating the LLRs to the leaves, meaning that f, g and C functions play a dominant role in the overall latency as the block length increases. Since the goal of this paper is to optimize the algorithm for long block lengths, it reduces considerable decoding steps with the block length growth. For instance, with reference to [19] with P_e value of 128, the proposed algorithm offers 8.3% and 37.8% latency decrement for block lengths of 512 and 32768, respectively. The effect of code rate in the binary case is summarized in Table 5. Obviously, the latency is independent from the code rate since it only relies on the location of frozen and information bits. From the table, it can be observed that under the proposed algorithm with $P_e = 120$, the latency of the code rate $\mathcal{R} = 6/8$ is less than that of the code rate $\mathcal{R} = 2/8$ where it is comparable to the latency of code rate $\mathcal{R} = 4/8$. Under two different P_e values, the minimum latency belongs to code rates $\mathcal{R} = 1/8$ and $\mathcal{R} = 7/8$ which mainly stems from frequent appearance of $R0$ and $R1$ nodes. It can be seen that the proposed algorithm has the lowest latency with respect to fast-SSC and [19] in all different rates. With regard to fast-SSC, it achieves up to 52.3% and 50.3% latency decrement for P_e values of 120 and 240, respectively.

The latency values for various mixed-kernel polar codes under the algorithm presented in [18] versus the proposed algorithm is summarized in Table 6. To have a fair comparison, the P_e values are considered identical to that of [18] for each code length. Obviously, the block lengths with only one binary kernel in their kernel sequences achieve the highest performance improvement since the algorithm extensively prunes the section of tree composed of consecutive binary kernels. It is shown that up to 84.6% latency improvement is achieved.

By considering a sufficiently large processor, each node of the polar tree can be considered as a single operation. Therefore, the complexity of the decoder can be referred to as the number of nodes that appear in the polar tree. Comparing the nodes appearing in [23] to that of the proposed algorithm (Table 7) for two code lengths of $N = 96$ and $N = 768$, the complexity is decreased up to 60.5%. This improvement is directly affected by the position of the frozen and information nodes appearing in the polar tree.

D. COST AND PERFORMANCE ANALYSIS

The FPGA utilization and performance evaluation of fast-SSC [6], [9], and [13], combinational SC [12] and the proposed algorithms in [19] and this paper, for a polar code of length $N = 1024$ and rate $\mathcal{R} = 1/2$ is tabulated in Table 8. We consider latency as the number of clock cycles (CCs) required for decoding a code stream and returning the corresponding codeword.

With reference to the fastest variant of fast-SSC, the proposed algorithm gains up to 66.8% higher information

TABLE 4. Latency comparison of fast-SSC versus the proposed algorithm for polar codes of different lengths with rate $R = \frac{1}{2}$.

Block Length	k	$P_e = 128$ L (CCs) (FSSC)	$P_e = 128$ L (CCs) (UF-SSC [19])	$P_e = 120$ L (CCs) (this work)	$P_e = 256$ L (CCs) (FSSC)	$P_e = 256$ L (CCs) (UF-SSC [19])	$P_e = 240$ L (CCs) (this work)
512	256	115	72	66	107	72	66
1024	512	214	123	102	191	100	95
2048	1024	392	274	212	332	212	190
4096	2048	749	565	415	597	410	346
8192	4096	1451	1130	773	1086	759	608
16384	8192	2832	2349	1543	2002	1502	1140
32768	16384	5612	4838	3007	3750	2935	2106

TABLE 5. Latency comparison of fast-SSC versus the proposed algorithm for polar codes of $N = 1024$ and different code rates.

Block Length	k	Rate	$P_e = 128$ L (CCs) (FSSC)	$P_e = 128$ L (CCs) (UF-SSC [19])	$P_e = 120$ L (CCs) (this work)	$P_e = 256$ L (CCs) (FSSC)	$P_e = 256$ L (CCs) (UF-SSC [19])	$P_e = 240$ L (CCs) (this work)
1024	128	1/8	181	107	92	162	90	87
1024	256	2/8	227	127	102	205	103	95
1024	384	3/8	228	140	115	202	117	109
1024	512	4/8	214	123	102	191	100	95
1024	640	5/8	232	145	127	208	121	117
1024	768	6/8	232	136	103	210	114	95
1024	896	7/8	208	122	85	182	95	74

TABLE 6. Latency comparison of conventional multi-kernel multi-code decoder versus the proposed algorithm for polar codes of different length with rate $R = \frac{1}{2}$.

Block Length	k	P_e	L (CCs) [18]	L (CCs) this work	Improv. (%)
48 {3,2,2,2,2}	24	36	137	32	76.6
96 {2,2,2,3,2,2}	48	36	272	65	76.1
192 {3,2,2,2,2,2,2}	96	36	587	151	74.3
324 {2,2,3,3,3,3}	162	120	652	271	58.4
384 {3,2,2,2,2,2,2,2}	192	120	1156	228	80.3
576 {2,2,2,2,2,2,3,3}	288	120	1234	453	56
768 {2,2,3,2,2,2,2,2,2}	384	120	2326	441	81
1536 {3,2,2,2,2,2,2,2,2,2}	768	240	4663	719	84.6
1728 {2,2,2,2,2,2,3,3,3}	864	240	3548	953	73.1
2916 {2,3,3,2,3,3,3,3}	1458	240	5953	1571	73.6

TABLE 7. Comparing the number of leaf nodes appear in polar tree under fast-SSC [23] versus the proposed decoder.

Block Length	R	# nodes [23]	# nodes this work	Improvement (%)
96	0.25	37	15	59.5
96	0.5	43	17	60.5
96	0.75	37	18	51.4
768	0.25	196	155	20.9
768	0.5	223	127	43
768	0.75	172	144	16.3

throughput. Taking advantage of larger P_e values, [6] and [9] offer considerably higher operating frequency respecting [13]. The latter work however, offers considerably lower latency. Comparing to [6] and [9], the maximum achieved clock frequency is slightly decreased under our proposed algorithm. This generally originates from additional routing and logic selection that increases the latency of the critical path.

In terms of resource consumption, [6], [9] and the proposed multi-kernel algorithm consume almost identical number of LUTs which mainly stems from larger P_e values in [6] and [9]. Our scheme saves 22.5% LUTs regarding [13] since the latter design employs a more complex instruction set and also needs more functions to implement the decoder. A moderate number of registers is used in all mentioned designs so far, where the difference comes from register duplication to address the target clock frequency.

Although designing multi-kernel architecture increases the consumed RAM, our scheme saves 6% and 8.2% RAM comparing to [6] and [9], respectively. This is due to the fact that pruning a through level of a polar tree can be interpreted as consuming $N \times Q_i$ lower bits of RAM. Also, different quantization scheme is used in our implementation. It worth noting that P_e value has no direct effect on the the amount of occupied memory. Our implementation consumes 1.2× higher RAM comparing to [13]. This is because the main goal of [13] is memory optimization.

A combinational SC decoder is implemented in [12] which offers 1.87× throughput achievement and 2.02× lower RAM consumption comparing to our multi-kernel decoder. However, area overhead is significant since 8× higher LUTs and 4.04× higher registers are employed

TABLE 8. FPGA utilization and performance comparison of polar codes of $\mathcal{P}(1024, 512)$ and rate $R = \frac{1}{2}$.

Work	Algorithm	P_e	FPGA family	Q (Q_i, Q_c, Q_f)	LUTs	Reg.	RAM (kbits)	f (MHz)	L (CCs)	T/P (Mbps)
[6]	fast-SSC	512	Altera	(6, 5, 1)	23,020	1,024	42.8	103	220	240
[9]	fast-SSC	512	Altera	(6, 5, 1)	23,353	5,814	43.8	103	204	259
[13]	fast-SSC	128	Altera	(6, 4, 1)	29,828	2,332	18.3	80.68	170	243
[12]	Combinational SC	-	Xilinx	(5, 5, 0)	190,127	22,928	13.3	-	-	1,240
[19]	UF-SSC	128	Xilinx	(7,6, 0)	18,982	3,384	37.7	94.36	123	393
this work	Multi-Kernel fast-SSC	120	Xilinx	(5,4, 0)	23,126	4,548	40.2	86.14	102	432

with reference to the proposed algorithm. Finally, comparing the proposed multi-kernel algorithm to [19], 19.6% LUTs and 25.6% registers are consumed which is mainly due to adding new functions, additional routing and logic selection. Also, 6.2% extra RAM is used due to overhead caused by multi-kernel architecture. The critical path is also increased by 8.7%. However, due to significantly decreasing the latency, the information throughput is increased by 9.9%.

VI. CONCLUSION AND FUTURE WORK

In this work, an efficient algorithm for decoding multi-kernel polar codes is presented. The proposed algorithm supports any code length with any rate constructed by binary-ternary mixed kernels with code length $N \leq N_{max}$. It offers fast decoding for a wide variety of code patterns in polar codes. An in-depth analysis along with a hardware architecture and FPGA implementation of the algorithm is provided. Decoding a polar code of length $N = 1024$ and rate $\mathcal{R} = 1/2$ with the maximum clock frequency and $P_e = 120$, an information throughput of 432 Mbps is obtained. The proposed algorithm improved the decoding latency by 52.3% in reference to the fast-SSC algorithm.

Future work foresees a hardware implementation of a CRC-concatenated SCL algorithm. The proposed node mergers under the SCL algorithm can substantially improve the reliability of polar codes respecting the original algorithm.

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. 2019.

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

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

[4] 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.

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

[6] 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.

[7] 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.

[8] C. Condo, V. Bioglio, and I. Land, "Generalized fast decoding of polar codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[9] 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.

[10] H. Zheng, S. A. Hashemi, A. Balatsoukas-Stimming, Z. Cao, T. Koonen, J. M. Cioffi, and A. Goldsmith, "Threshold-based fast successive-cancellation decoding of polar codes," *IEEE Trans. Commun.*, vol. 69, no. 6, pp. 3541–3555, Jun. 2021.

[11] H. Zheng, A. Balatsoukas-Stimming, Z. Cao, and T. Koonen, "Implementation of a high-throughput fast-SSC polar decoder with sequence repetition node," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2020, pp. 1–6.

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

[14] 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. 2019.

[15] K. Niu, K. Chen, and J.-R. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3423–3427.

[16] R. Wang and R. Liu, "A novel puncturing scheme for polar codes," *IEEE Commun. Lett.*, vol. 18, no. 12, pp. 2081–2084, Dec. 2014.

[17] M. Benammar, V. Bioglio, F. Gabry, and I. Land, "Multi-kernel polar codes: Proof of polarization and error exponents," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2017, pp. 101–105.

[18] G. Coppolino, C. Condo, G. Masera, and W. J. Gross, "A multi-kernel multi-code polar decoder architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4413–4422, Dec. 2018.

[19] H. Rezaei, V. Ranasinghe, N. Rajatheva, M. Latva-aho, G. Park, and O.-S. Park, "Implementation of ultra-fast polar decoders," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2022, pp. 235–241. [Online]. Available: https://ieeexplore.ieee.org/document/9814456/

[20] F. Gabry, V. Bioglio, I. Land, and J. C. Belfiore, "Multi-kernel construction of polar codes," in *Proc. IEEE Int. Conf. Commun. Workshops, (ICC Workshops)*, May 2017, pp. 761–765.

[21] V. Bioglio, F. Gabry, I. Land, and J.-C. Belfiore, "Minimum-distance based construction of multi-kernel polar codes," in *IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.

[22] K.-W. Shin and H.-J. Kim, "A multi-mode LDPC decoder for IEEE 802.16e mobile Wimax," *J. Semiconductor Technol. Sci.*, vol. 12, no. 1, pp. 24–33, Mar. 2012.

[23] A. Cavatassi, T. Tonnellier, and W. J. Gross, "Fast decoding of multi-kernel polar codes," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–6.

[24] H. Gamage, V. Ranasinghe, N. Rajatheva, and M. Latva-aho, "Low latency decoder for short blocklength polar codes," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2020, pp. 305–310.

[25] 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.

HOSSEIN REZAEI (Graduate Student Member, IEEE) received the M.Sc. degree in digital electronics from the Iran University of Science and Technology, Tehran, Iran, in 2016. He is currently pursuing the Ph.D. degree with the University of Oulu, Oulu, Finland. His current research interests include channel coding algorithm design with a focus on polar codes, digital predistortion (DPD) design, and implementation of communication systems on embedded platform.

NANDANA RAJATHEVA (Senior Member, IEEE) received the B.Sc. degree (Hons.) in electronics and telecommunication engineering from the University of Moratuwa, Sri Lanka, in 1987, and the M.Sc. and Ph.D. degrees from the University of Manitoba, Winnipeg, MB, Canada, in 1991 and 1995, respectively. He is currently a Professor with the Centre for Wireless Communications, University of Oulu, Finland. During his graduate studies, he was a Canadian Commonwealth Scholar in Manitoba. From 1995 to 2010, he was a Professor/an Associate Professor with the University of Moratuwa and the Asian Institute of Technology, Thailand. He is currently leading the AI-driven air interface design task in Hexa-X EU Project. He has coauthored more than 200 refereed papers published in journals and in conference proceedings. His research interests include physical layer in beyond 5G, machine learning for PHY and MAC, integrated sensing and communications, and channel coding.

MATTI LATVA-AHO (Senior Member, IEEE) received the M.Sc., Lic.Tech., and Dr.Tech. (Hons.) degrees in electrical engineering from the University of Oulu, Finland, in 1992, 1996, and 1998, respectively. From 1992 to 1993, he was a Research Engineer at Nokia Mobile Phones, Oulu, Finland, after that he joined the Centre for Wireless Communications (CWC), University of Oulu. He was the Director of CWC, from 1998 to 2006, and the Head of the Department for Communication Engineering, until August 2014. Currently, he serves as an Academy of Finland Professor and is the Director for National 6G Flagship Program. He is also a Global Fellow with Tokyo University. His research interests include mobile broadband communication systems and currently his group focuses on 6G systems research. He has published over 500 journals or conference papers in the field of wireless communications. In 2015, he received the Nokia Foundation Award for his achievements in mobile communications research.