IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 January 2024, accepted 30 January 2024, date of publication 8 February 2024, date of current version 16 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3364384

RESEARCH ARTICLE

Efficient Hardware Realization of SC Polar Decoders Using Compound Pipelined Processing Units and Auxiliary Registers

YASIR ALI[1,2], YUANQING XIA[3,1], (Fellow, IEEE), TAYYAB MANZOOR[4], SHAHZAD ALI[1], MOHAMED ABOUHAWWASH[5], S. S. ASKAR[6], AMIT KRISHAN KUMAR[7,8], (Member, IEEE), AND RUIFENG MA[1]

[1]School of Automation, Beijing Institute of Technology, Beijing 100081, China
[2]Department of Telecommunication and Teleinformatics, Joint Doctoral School, Silesian University of Technology, 44-100 Gliwice, Poland
[3]Zhongyuan University of Technology, Zhengzhou 450007, China
[4]School of Automation and Electrical Engineering, Zhongyuan University of Technology, Zhengzhou 450007, China
[5]Department of Mathematics, Faculty of Science, Mansoura University, Mansoura 35516, Egypt
[6]Department of Statistics and Operations Research, College of Science, King Saud University, Riyadh 11451, Saudi Arabia
[7]Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam
[8]Faculty of Electrical-Electronic Engineering, Duy Tan University, Da Nang 550000, Vietnam

Corresponding author: Yuanqing Xia (xia_yuanqing@bit.edu.cn)

This project is funded by King Saud University, Riyadh, Saudi Arabia. Researchers Supporting Project number (RSP2024R167).

ABSTRACT Polar codes have garnered substantial research attention due to their impressive performance characteristics and have found applications in recent technologies, including 5G New Radio (NR) systems, Internet of Things (IoT) communications, and cyber-physical systems that utilize sensor and actuator networks. However, the existing SC decoders suffer from lengthy processing latencies due to their sequential processing steps, thereby restricting the practical applicability of polar codes. To address this latency issue, this paper introduces a Compound Pipeline Processing Unit (CPPU) and its simplified counterpart, a crucial step in realizing tree-level compound pipelining. In contrast to sequential circuitry, the previously described combinational architecture lacks internal storage elements, with the clock period defined by the longest path delay. This strategy conserves hardware resources by avoiding memory usage, but it inevitably decelerates the decoder's performance. Notably, implementation results underline the efficiency of the proposed CPPU-based SC polar decoder using a fully unrolled encoder and decoder on the targeted platform of a Virtex UltraScale - XCVU190 Field Programmable Gate Array (FPGA), using a parametric approach in the Very High-Speed Integrated Circuit Hardware Description Language (VHDL). The assessment of error-correction performance involves examining various combinations of integral and fractional bits in LLR quantized representations. This approach achieves a throughput of about 2672 Mbps, accompanied by a substantial reduction of 17% in Lookup Table (LUT) usage. Furthermore, the decoder's speed is enhanced by approximately 17.34% for a code length of 128 bits and LLR quantization of 5 bits.

INDEX TERMS Polar codes implementations, compound-logic pipeline processing, latency reduction, simplified non-statistical LLR metric.

I. INTRODUCTION

After Arikan's pioneering work [1], significant research efforts have been focused on exploring polar codes, driven by their remarkable performance. Consequently, this has resulted in their integration into modern applications like

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

5G New Radio (NR) systems [2], Internet of Things (IoT) communications [2], and smart sensor networks in cyber-physical systems [3], [4].

For decoding transmitted polar codewords, the Successive Cancellation (SC) algorithm is commonly the initial choice due to its efficient error correction and computational simplicity [5], [6]. However, the traditional SC decoding method is known to experience prolonged processing delays stemming from the serial nature of internal decoding orders [1]. This leads to increased overall decoding latency when employed in SC decoding techniques that aim to enhance error correction further [3], [7]. Hence, it becomes essential to create viable low-latency SC decoding strategies to achieve economical SC decoders suitable for lightweight communication protocols [8]. These strategies can also serve as rapid foundational architectures for SC decoders within 5G NR solutions [9]. However, all the prior methodologies have derived from the basic decoding tree structure, following a sequential node processing approach. These methods tend to be hindered by the serialized steps inherent in such an approach, which ultimately leads to increased decoding latency. Recognizing this limitation, recent cutting-edge advancements have put forth the concept of tree-level parallelism by leveraging the pipelining operation. This involves breaking down the primary decoding tree into smaller sub-trees, theoretically enabling simultaneous parallel decoding operations [10].

This paper introduces a range of optimization techniques designed to achieve a low-latency combinational pipelined processing SC polar decoder. In the context of a generalized pipelined decoder architecture, we define augmentation of the mother code, which we call compound-logic pipelined codes producing leaf-level patterns of merged sub-trees. Each of these functions offers a dedicated processing path for a specific sub-tree pattern, thereby shortening the critical path in comparison to prior approaches, which often use a single merging function to handle numerous patterns [10]. Significantly, concise segmented polar codes have been put to practical use as the effective error correction code for the improved mobile broadband control channel within 5G networks [11], [12]. This paper's primary aim is to delve into the design and implementation of a latency-efficient polar code encoder and SC decoder system on an FPGA platform. This implementation employs both fully unrolled and combinational architecture, enabling the processing of an entire codeword in a single clock cycle, which offers significant advantages in terms of achieving high throughput. In alignment with the recent 5G NR specification [8], we conduct a comprehensive case study involving a pipelined polar decoder realized on a Virtex UltraScale - XCVU190 FPGA platform. Through rigorous testing, our decoder demonstrates an impressive approximately 17.34% increase in throughput compared to the reference combinational implementations for $N = 128$ [13], [14], [15]. At the core of our decoder lies our proposed Compound-Logic Pipelined Processing Unit (CPPU) and its simplified counterpart, a foundational component that reduces resource demands for the widely used 5-bit Log Likelihood Ratio (LLR) quantization. Simultaneously, it mitigates decoder latency by pre-computing results, setting it apart from reference designs [13], [14], [15]. This innovation contributes to improved performance and efficiency, making our decoder implementation a valuable asset in modern communication systems. The primary contribution of this research endeavor can be summarized as follows:

- To introduce a Compound Pipeline Processing Unit (CPPU) integrated with auxiliary registers, which plays a crucial role in achieving tree-level compound pipelining for SC decoders. By incorporating this novel architecture, the paper addresses the issue of lengthy processing latencies associated with sequential SC decoders, thereby improving the practical applicability of polar codes in various applications.
- Instead of using individual units for each LLR input, the paper introduces a modification that simplifies the relationships within the decoder and improves its efficiency compared to the basic decoder, enhancing the overall decoding process.
- The paper contributes to the hardware implementation of polar codes by demonstrating the efficiency of the proposed CPPU-based SC polar decoder. The decoder is implemented on a Virtex UltraScale - XCVU190 FPGA using a parametric approach in VHDL. This hardware implementation achieves a substantial throughput of approximately 2672 Mbps, while also reducing the Lookup Tables (LUTs) usage by 17%. By utilizing the proposed S-CPPU-based architecture, the paper achieves a significant speed improvement of approximately 17.34% for a code length of 128 bits and LLR quantization of 5 bits.

The paper's subsequent sections are organized in this manner: Section II covers polar code encoding and SC decoding algorithms, emphasizing hardware-friendly simplifications. Section III discusses the hardware implementation background and overview of the related work done. Section IV discusses the proposed CPPU architecture, our parametric design, and encoder implementation. It also introduces decoder components for $N = 8$ and LLR representation considerations. Section V details the Simplified CPPU (S-CPPU), and explains our basic $N = 4$ decoder implementation with simplified relationships. Section VI presents hardware implementations on FPGA to go through the synthesis outcomes, comparing our designs with references. It includes throughput and frequency analysis, BER performance based on quantization representation, and a simplified non-statistical metric. Section VII concludes the paper with a discussion of the results.

II. PRELIMINARIES

A. SC POLAR CODE

A polar code with a configuration of (N, K) is characterized as a linear block code consisting of N transmitted bits,

out of which K are information bits. Prior to the encoding process, the message vector $\mathbf{u} = [u_0, u_1, \ldots, u_{N-1}]$ is formed by allocating the K information bits to the K most reliable channels among the N channels, as determined by the channel polarization process. The remaining bits, known as frozen bits, are then assigned predetermined values recognized by both the encoder and decoder, often set to all zeros [11]. Following the approach detailed in [1], the encoding procedure of a polar code is mathematically represented as the product of a $1 \times N$ message vector $\mathbf{u}$ and an $N \times N$ generator matrix $\mathbf{G}^{\otimes n}$, denoted as $\mathbf{x} = \mathbf{u}\mathbf{G}^{\otimes n}$. Here, $n = \log_2 N$, $\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and $\mathbf{G}^{\otimes n}$ represents the n-th Kronecker power of the polarizing matrix $\mathbf{G}$.

Following the passage of the codeword vector $\mathbf{x}$ through a channel affected by noise, the polar decoder receives the $1 \times N$ vector $\mathbf{y}$ and then proceeds to compute the message bits $\hat{\mathbf{u}} = [\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{N-1}]$. For a length N polar code, the conventional SC decoding algorithm's processing steps can be represented as a binary decoding tree with a depth of $\log_2 N$ [16], and the tree structure can be represented as a graph with $N\left(1 + \log_2 N\right)$ nodes. Fig. 1 illustrates the SC decoding tree's conceptual diagram for a (16, 8) polar code, where blue, white, and grey nodes correspond to nodes with information, frozen, and mixed leaves, respectively.

For a comprehensive illustration of the SC decoding process, let's view $\gamma_{i,j}$ as the j-th node within the i-th stage of the decoding tree, where $L_i = 2^{n-i}$ denotes the number of leaf nodes. At node $\gamma_{i,j}$, a soft-value vector $\boldsymbol{A}_{i,j} = [\alpha_0^{i,j}, \alpha_1^{i,j}, \ldots, \alpha_{L_i-1}^{i,j}]$ is received from its parent node, and the node then produces an estimated hard-value vector $\boldsymbol{B}_{i,j} = [\beta_0^{i,j}, \beta_1^{i,j}, \ldots, \beta_{L_i-1}^{i,j}]$.

The initial assignment of the soft computation vector $\boldsymbol{A}_{0,0}$ for the root node $\gamma_{0,0}$ is performed by utilizing the received vector $\mathbf{y}$ in the subsequent manner:

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

The LLR is a crucial parameter used in the decoding process of polar codes to make decisions about the transmitted bits based on received observations. The LLR on the node $\gamma_{i,j}$ is defined in [16] as:

$$\alpha_l^{i,j} = \ln \frac{Pr\left(\mathbf{y}, \hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{j-1} \mid u_j = 0\right)}{Pr\left(\mathbf{y}, \hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{j-1} \mid u_j = 1\right)}, \quad 0 \le l \le N-1, \tag{2}$$

where, $\alpha_l^{i,j}$ refers to the LLR value for the j-th bit in the i-th level of the polar code;
$P(\mathbf{y}, \hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{j-1} \mid u_j = 0)$ represents the conditional probability of receiving the sequence of observations $\mathbf{y}$, along with the estimated values $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{j-1}$, given that the j-th bit (u_j) in the polar code is set to 0;
$P(\mathbf{y}, \hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{j-1} \mid u_j = 1)$ represents the conditional probability of receiving the same sequence of observations and estimated values, but with the j-th bit (u_j) set to 1.

Within the SC decoding algorithm, node $\gamma_{i,j}$ sends computed soft-value vectors, $\boldsymbol{A}_{i+1,2j}$ and $\boldsymbol{A}_{i+1,2j+1}$, to its left and right child nodes correspondingly. The individual elements of the soft-value vectors are computed as follows:

$$\begin{aligned} \alpha_l^{i+1,2j} &= f\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}\right), \\ \alpha_l^{i+1,2j+1} &= g\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}, \beta_l^{i+1,2j}\right), \end{aligned} \tag{3}$$

where l represents the vector index $(0 \le l \le L_{i+1} - 1)$.
It is worth noting that while the encoder can be easily implemented with Boolean logic, the decoder involves soft decision propagation through these circuits. The formulations for the decoder functions can be expressed from [17],

$$\begin{aligned} f\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}\right) &= 2\tanh^{-1}\left(\tanh\left(\frac{\alpha_l^{i,j}}{2}\right)\tanh\left(\frac{\alpha_{l+L_{i+1}}^{i,j}}{2}\right)\right), \\ g\left(\alpha_l^{i,j}, \alpha_{l+L_{i+1}}^{i,j}, \beta_l^{i+1,2j}\right) &= \alpha_l^{i,j}(-1)^{\hat{s}} + \alpha_{l+L_{i+1}}^{i,j}. \end{aligned} \tag{4}$$

where $\hat{s} = 1$ if $\beta_l^{i+1,2j} > 0$ and $\hat{s} = 0$ otherwise. Using LLRs, function g can be easily implemented through adder and subtractor circuits. Proceeding towards the hardware-friendly approximation, f and g are defined in [18] as,

$$\begin{aligned} f(x, y) &\approx \operatorname{sgn}(x)\operatorname{sgn}(y)\min(|x|, |y|), \\ g(x, y, u) &= (-1)^u x + y, \end{aligned} \tag{5}$$

where $\operatorname{sgn}(x)$ is 1 for $x \ge 0$ and -1 in other cases. When given the hard-value vectors $\boldsymbol{B}_{i+1,2j}$ and $\boldsymbol{B}_{i+1,2j+1}$ from the left and right child nodes respectively, node $\gamma_{i,j}$ produces a hard-value vector $\boldsymbol{B}_{i,j}$ realized as partial sum in our text, using the following process:

$$\begin{aligned} \beta_l^{i,j} &= \beta_l^{i+1,2j} \oplus \beta_l^{i+1,2j+1}, \\ \beta_{l+L_{i+1}}^{i,j} &= \beta_l^{i+1,2j+1}. \end{aligned} \tag{6}$$

At the j-th leaf node $\gamma_{m,j}$, the hard-value vector $\boldsymbol{B}_{m,j}$ corresponds to an estimated bit, $\boldsymbol{B}_{m,j} = \left[\beta_0^{m,j}\right] = [\hat{u}_j]$, computed as

$$\hat{u}_j = \beta_0^{m,j} = \begin{cases} 0, & \text{if } j \in \mathcal{A}_c \\ 0, & \text{if } \alpha_0^{m,j} \ge 0 \\ 1, & \text{otherwise,} \end{cases} \tag{7}$$

where $\mathcal{A}_c$ represents the set of indices corresponding to frozen bit locations. The standard SC decoding process concludes its cycle when the rightmost leaf node $\gamma_{m,N-1}$ computes the final hard-decision estimate, resulting in the decoded output $\hat{\mathbf{u}} = [\beta_0^{m,0}, \beta_0^{m,1}, \ldots, \beta_0^{m,N-1}]$.

Since the operation g at $\gamma_{i,j}$ node depends on partial sums obtained from previously estimated bits, denoted as the vector $\boldsymbol{B}_{i+1,2j}$, the traditional SC decoding procedure frequently experiences significant processing latencies due to its sequential decoding sequence. To tackle this issue of latency, methods such as those explored in [13], [19], [20], [21], [22], and [23] present pruning-based algorithms

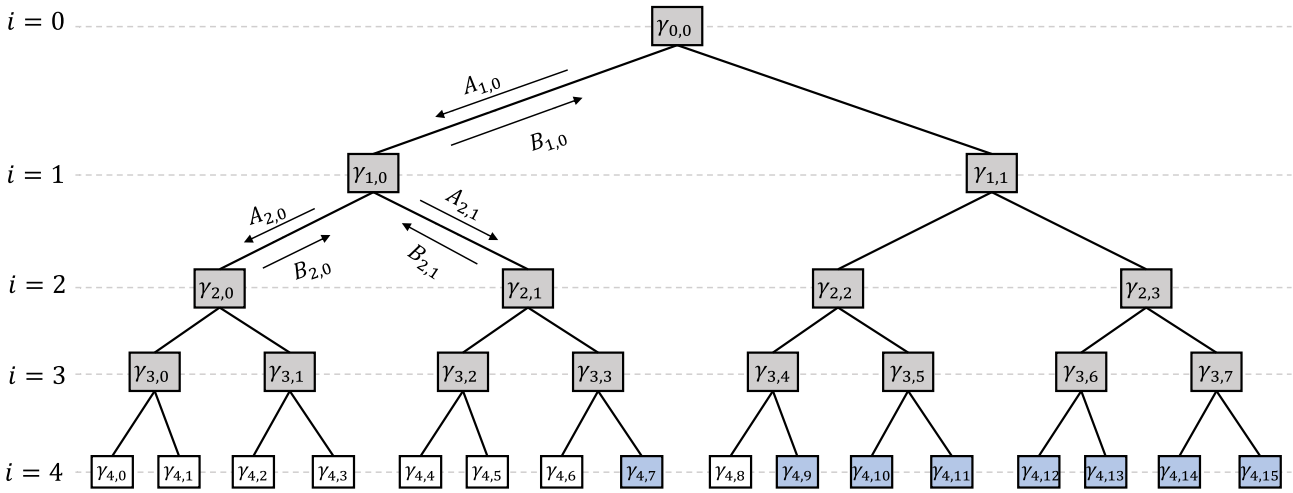


FIGURE 1. Illustrating the structure of the successive cancellation (SC) decoding tree designed for a (16,8) polar code, showcasing the hierarchical arrangement and interconnections of nodes involved in the decoding process.

for SC decoding. The pipelined decoding technique outlined in [24] employs a segmentation pruning approach for distinct patterns of consecutive leaf nodes. Enhancing this hardware implementation could involve incorporating auxiliary registers to facilitate overlapped pruning in the decoder, as demonstrated in Fig. 2. Nonetheless, despite the advancements of these pruning-based strategies, they still adhere to the original SC decoding tree structure, thus retaining inherent latency constraints arising from the sequential processing sequence.

III. RELATE WORK AND ARCHITECTURE CONSIDERATIONS

As previously discussed, the hardware development of SC decoders has presented notable difficulties. To alleviate the intricacies of SC decoding, Arikan [25] presented pipelined architectures incorporating identical reusable modules. This design approach enables recursive implementations with reduced complexity. Leroux et al. [17] demonstrated that SC decoders can be constructed using $O(N)$ processing elements, denoted as configurable units capable of executing either the f or g functions. In their subsequent publication [16], they introduced a semi-parallel architecture characterized by remarkably low processing complexity. Additionally, they proposed an alternative approach based on encoding, inspired by the framework introduced in [26], and put forward a fully parallel module for computing partial sums. Sarkis et al. [21] implemented a simplified version of the SC algorithm incorporating components codes [27]. Berhault et al. [28], [29] developed an effective module for partial sum computation that could also function as an encoder utilizing a linear feedback shift register. In a subsequent study [30], they introduced an innovative approach for storing intermediate results using computational logic. Giard et al. [31] introduced a novel FPGA architecture characterized by full unrolling and deep pipelining at the expense of increased memory requirements. They subsequently explored the landscape of polar decoders in [22] and [32], introducing unrolled architectures specifically tailored for quantized polar codes.

An outline of an SC flip decoder that features decreased memory demands and demonstrates enhanced Frame Error Rates (FERs) was presented in the work by Afisiadis et al. [33]. In [34], they exhibited both software and hardware realizations of adaptable polar decoders. Oommen and colleagues introduced a resource-efficient hardware design in their study [35], which incorporates stack SC principles [36] within an FPGA environment. In a significant breakthrough, Dizdar and Ar?kan [8] introduced an innovative design for an SC decoder acclaimed for its exceptional throughput and energy efficiency. Although the incorporation of lists to improve decoding performance results in elevated hardware complexity and latency, as noted by [37], contemporary hardware realizations have prioritized enhancing throughput and reducing area utilization by means of component enhancements [38], [39], the utilization of segmentation strategies [40], advancements in path metric processing [41], and the adoption of constituent decoding techniques [42].

In this study, we investigated methods for representing sequential patterns that rely on both the length of the code denoted as N and the quantization bit quantity Q assigned to LLR values. We chose to develop a high-speed decoder using a combinational approach, similar to Dizdar's concept [8]. However, we innovatively devised a CPPU based on algorithmic considerations for handling addition and subtraction in Sign-Magnitude (SM) representation, accounting for possible overflow and saturation scenarios in the output. Furthermore, we presented a less complex iteration of this CPPU and provided synthesis results for both variations. As a result, we introduce a more efficient decoder for $N = 4$, utilizing a combination of two CPPUs, two comparators, and a small number of additional

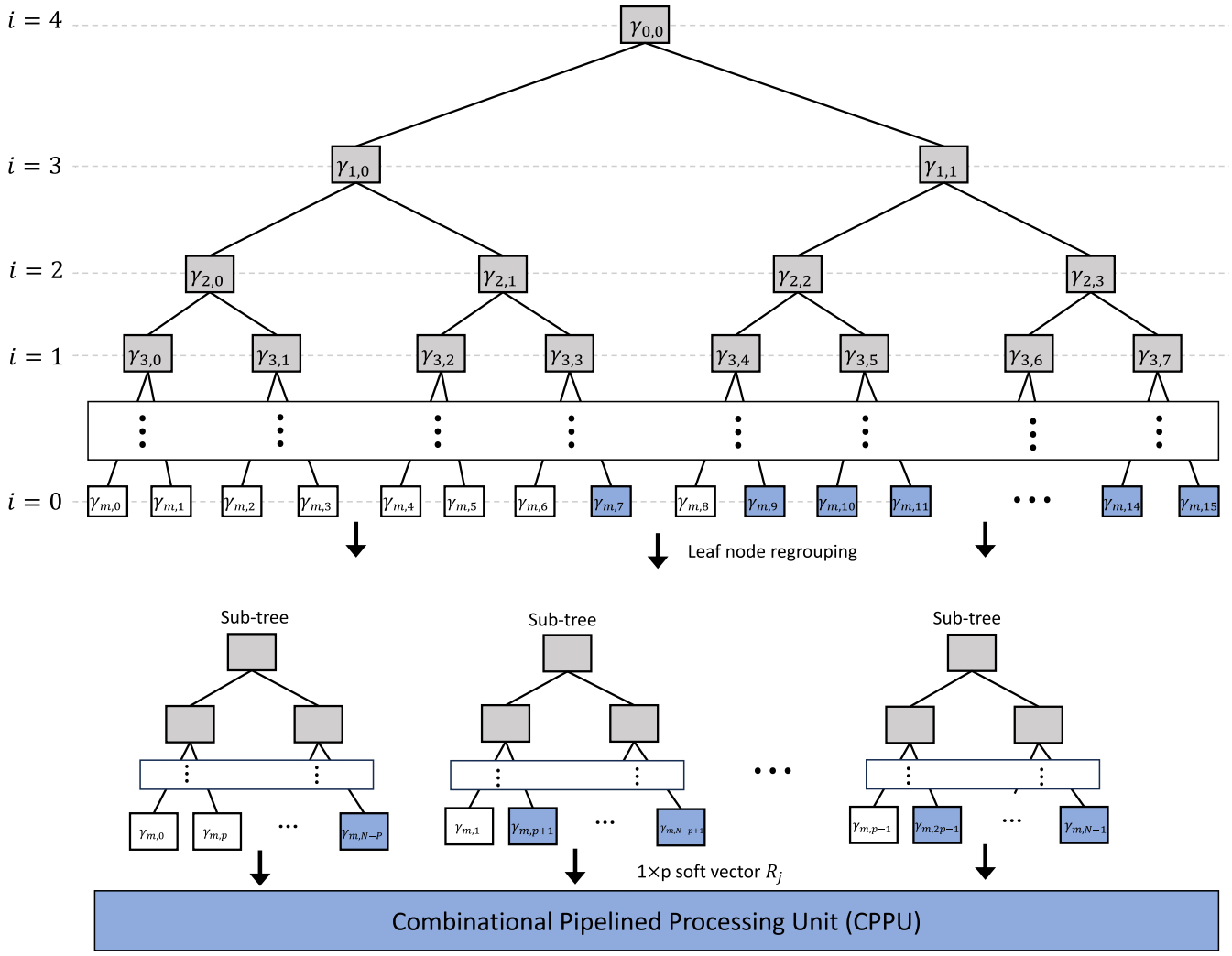


FIGURE 2. Illustration of SC decoder integration with Compound Pipeline Processing Unit (CPPU) to enable concurrent pruning within the decoder.

logic gates. This simplification significantly decreases the decoding delay and preserves hardware assets.

A. ENCODER ARCHITECTURE CONSIDERATIONS

Arikan in [25] introduced hardware adaptable architectures for polar codes that can be applied to any power-of-2 code length, denoted as N. These structures are constructed using reusable components, allowing for pipelining and providing a consistent graphical denotation of the $\mathbf{G}^{\otimes n}$ implementation. Within this array of architectures, one particular design incorporates a reverse-shuffle operator, which reorganizes an even-length input vector v_1^N with N components into a rearranged sequence $(v_1, v_3, \ldots, v_{N-1}, v_2, v_4, \ldots, v_N)$. This architecture also employs bitwise XOR ($\oplus$) operations, which transform binary vectors v_1^N of even length into $(v_1 \oplus v_2, v_2, v_3 \oplus v_4, v_4, \ldots, v_{N-1} \oplus v_N, v_N)$, where $\oplus$ denotes modulus 2 addition. When implementing an SC polar encoder on an FPGA, several considerations must be taken into account. First, the complex recursive nature of SC encoding requires careful pipelining and scheduling to ensure efficient hardware resource utilization and minimal latency. Efficient memory management is essential to store intermediate values and manage recursive computations. The choice of architectural parameters, such as the degree of parallelism and the depth of the processing pipeline, significantly impacts the throughput and latency of the encoder. Additionally, mapping the intricate polar code structure onto FPGA logic elements requires careful consideration of resource allocation, including LUTs, flip-flops, and DSP blocks. Leveraging dedicated high-speed interfaces and memory banks and registers can optimize data throughput and alleviate potential bottlenecks. As FPGA technology evolves, leveraging newer features like heterogeneous architectures, custom IP cores, and partial reconfiguration can further enhance the encoding performance. Thus, achieving an efficient SC polar encoder implementation on FPGA demands a careful balance of architectural, algorithmic, and resource management considerations. Fig. 3 illustrates the graph tree of the conventional encoder diagram for $N = 8$, serving as the basis for this architecture. Within this

depiction, a vector **u** undergoes three stages of transformation to yield a codeword **x**, entailing the interconnection of modulus 2 adders.

B. DECODER ARCHITECTURE CONSIDERATIONS

The architecture of SC decoder can be realized using combinational logic due to it does not operate any loops. This design permits the decoding of one codeword in each clock cycle, resulting in reduced power consumption when compared to synchronous decoding methods [8]. Streamlined versions of the SC algorithm [21], [27] also enable the creation of high-throughput decoders, yet they lack the flexibility to adapt to varying code rates. Nevertheless, the inherent recursive nature of polar codes simplifies their implementation. Specifically, the construction of a polar code with a length of N involves concatenating two polar codes of length $N/2$. This inherent property streamlines the development of decoders for codes with a length of N based on decoders designed for polar code of length $N/2$ [8].

In our design approach, we introduce a register transfer level schematic for a combinational decoder architecture designed for length $N = 16$, employing the recursive algorithm. The architecture, depicted in Fig. 4, comprises several fundamental components: a register dedicated to preserving frozen bits, encoders for generating partial sums, and elementary decoders responsible for computing the f and g functions. As the foundational case of the recursive structure, the decoder blocks for $N = 4$ establish the basis from which the overall decoder layout for any length can be extrapolated. The register is employed to hold the frozen bit indicator vector, wherein ''0'' signifies frozen bits while ''1'' designates information bits.

In contrast to sequential circuits, the previously explained combinational architecture doesn't necessitate any internal storage components. The clock period in such a circuit is determined by the longest path delay. This design choice conserves hardware resources by eliminating the need for memory at the cost of slowing down the decoding process. In our work, we introduce compound logic pipelining to enhance throughput, even if it requires additional hardware utilization. The outputs of the initial decoder block are used by the encoder to compute partial sums. For this reason, it's crucial for this decoder to retain its outputs once the respective beliefs are computed. However, the pipeline nature allows it to initiate the decoding process for another codeword as long as the associated partial sums, together with the pertinent channel observation LLRs, are stored and made available to the next decoding block for parallel processing. Incorporating auxiliary register blocks at strategic locations within the decoder's architecture facilitates the realization of parallel decoding and improves the decoding latency.

Within the decoder segment encompassing LLRs from l_0 to l_7, a recursive approach similar to the initial decoder's structure is maintained through the utilization of two elementary decoders having length $N = 4$ and an encoder of identical length. A CPPU generator block, adaptable through parameters, is introduced to facilitate combined-processing decoder blocks totaling $N/2 = 4$, responsible for the implementation of the f and g functions.

The underlined decoder integrates combinational encoders designed for various code lengths, serving the purpose of generating partial sums. These encoders uphold the bit-reverse reordering pattern, segmenting the encoding process at the transmission end. The foundational partial sum generator is instantiated as an XOR gate, operating as a pivotal component within the segmented decoder framework. When recursion comes into play, a decoder tailored for a particular code length N is formulated, and the pertinent partial sum generator for this context takes the form of an encoder with a length of $N/2$.

As an example, an 8-bit decoder requires the utilization of $2 \times e_2 + 1 \times e_4$ encoders. In addition, the decoder architecture encompasses three registers: a $(N \times 1)$ bit register for holding frozen bit positions, a $(N \times Q)$ bit register for storing the initial LLR values received from the channel, and another $(N \times 1)$ bit register responsible for storing the decoded message bits produced at the decoder's output.

Highlighting a crucial aspect, it's essential to stress that translating the f and g functions into hardware requires quantifying LLRs using a predetermined bit quantity denoted as Q. The selection of this bit-width detonation can influence the outcome of the decoder as opposed to the use of floating-point representations. Through the simulation of the SC decoding for varying Q values, it becomes evident that a 6-bit binary representation produces nearly indistinguishable performance compared to that of a floating-point representation. While a choice of $Q = 5$ leads to marginally inferior performance, it offers a reasonable compromise that helps in reducing hardware demands. Hardware implementations of polar codes frequently opt for fixed-point quantization, such as Sign-Magnitude (SM) representations, which allocate 1 bit for the sign and $Q - 1$ bits for representing the magnitude. The SM deployment, in particular, can mitigate hardware complexity, thus emerging as a recommended preference for the quantized operation.

IV. COMPOUND-LOGIC PIPELINED PROCESSING UNIT (CPPU)

This section introduces a distinct CPPU SC decoder architecture for polar codes step by step, namely the combinational decoder, the pipelined combinational decoder, the compound-logic pipeline processing decoder, and the simplified CPPU decoder. The recursive nature of the SC decoder enables the combinational decoder, initially designed for $N = 4$, as depicted in 5. This logic-based decoder serves as the basic building block for larger-scale decoders, elaborated upon in the subsequent subsection.

A. BASIC COMBINATIONAL-LOGIC IMPLEMENTATION

In a combinational SC decoder, the decoder outputs are directly determined by the inputs without any intervening memory units or registers. The implementation of this

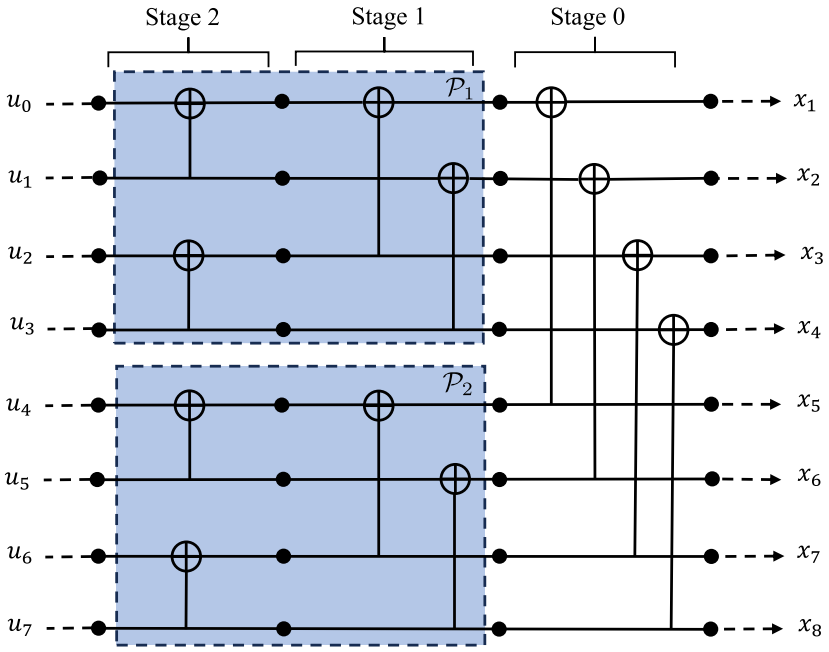


FIGURE 3. Diagram illustrating the polar encoder for the case of $N = 8$.

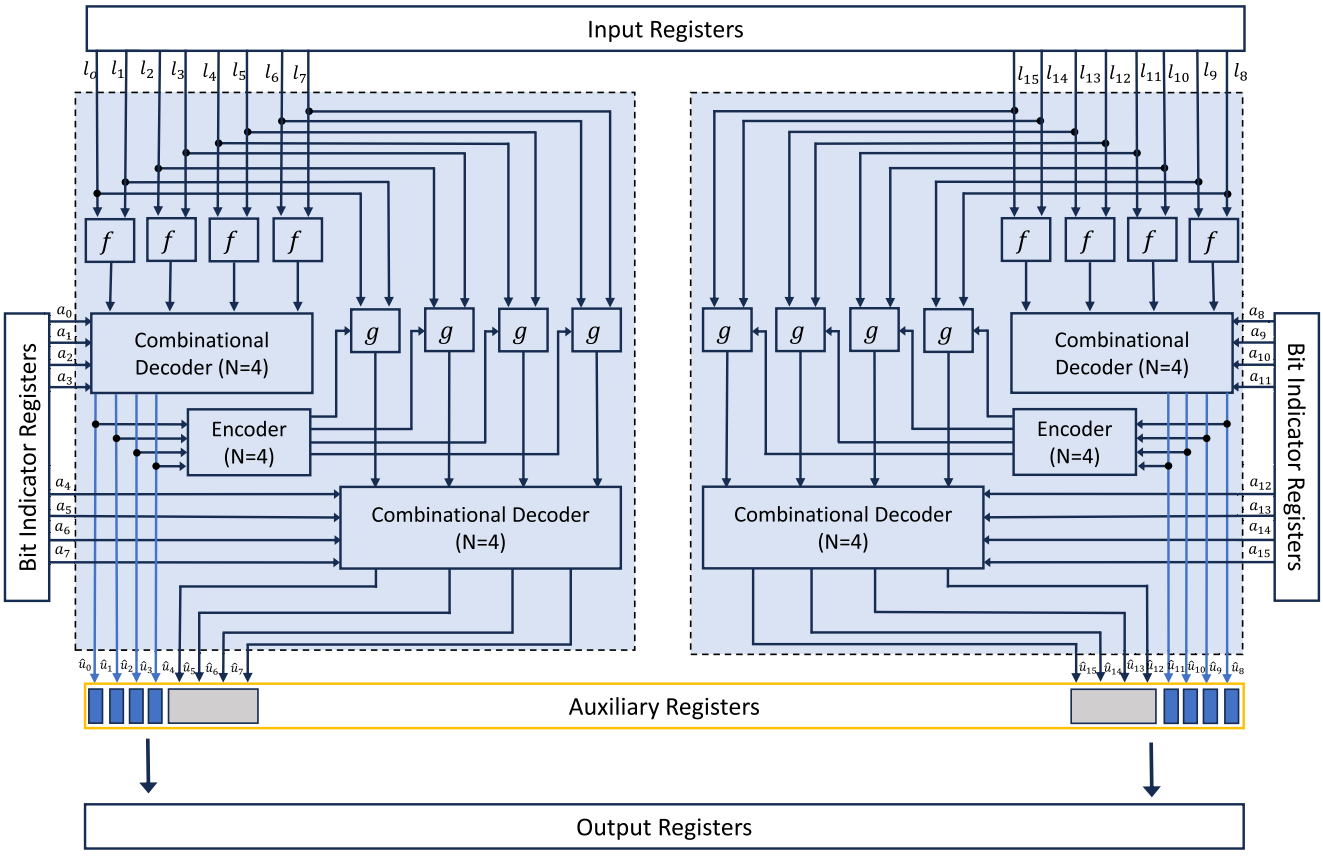


FIGURE 4. Register transfer level schematic depicting the combinational decoder for $N = 16$ and $K = 8$.

decoder, utilizing only comparators and adders, is depicted in Fig. 5. This design adopts a sign-magnitude representation, similar to the approach in [16], which alleviates the need for frequent representation conversions. Throughout the decoder, both channel observation LLRs and calculations are carried out using Q bits. The computation of the function g from (5)

follows the latency-reduction technique proposed in [39]. In Fig. 5, we showcase the combinational logic decoder for a size of N = 4, while Fig. 6 illustrates its signal flow graph alongside the decoding expressions.

B. COMBINATIONAL SC DECODER

Illustrated in the Register Transfer Level (RTL) schematic in Fig. 4, the combinational decoder architecture for $N = 16$ could be extended to any block length N, employing the recursive nature of the SC decoding. This design integrates two combinational decoders with dimensions of $N/2$, supported by input/output registers and bit indicator resistors. The sub-modules of size 4 within the decoder remain consistent with those in Fig. 5. The encoder with a size of 4 employs a combinational circuit constructed using XOR gates. The components within a combinational decoder are directly interconnected, bypassing any intermediate synchronous logic elements. This design choice not only saves time and power by eliminating memory read/write operations but also simplifies the complexity of the hardware. In each clock cycle, a fresh channel observation LLR vector is extracted from input registers, and a decision vector is written into output registers. The duration of each clock cycle aligns with the overall delay of the combinational circuit, effectively determining the decoder's throughput. To distinguish between frozen and data bits, AND gates are utilized, leveraging frozen bit indicators denoted as a_i. At the commencement of each decoding process, the frozen bit indicator vector can be modified, allowing for real-time adjustments to the code configuration.

Combinational decoders exhibit a recurring recursive structure composed of multiple essential components. The architecture of the 16-bit decoder is established by interconnecting $2 \times 2 \times 4$-bit decoders, creating links through a 16-bit input-output register block. This concept can be extended further; for example, a decoder targeting $N = 1024$ requires 64 CPPU units in conjunction with 1×4-bit encoders for each CPPU.

C. PIPLINED COMBINATIONAL DECODER

In contrast to sequential circuitry, the previously described combinational architecture lacks internal storage elements, with the clock period defined by the most extended path delay. This strategy saves hardware resources by avoiding memory usage but inevitably decelerates the decoder's performance. The present subsection introduces pipelining to enhance throughput at the cost of introducing additional hardware complexity in auxiliary registers.

As emphasized in Fig. 4, the outputs of the initial decoder block are utilized by the encoder to compute partial sums. For this reason, it's crucial for this decoder to retain its outputs once it completes the pruning of the decoding tree. However, the pipeline nature allows it to initiate the decoding process for another codeword as long as the associated partial sums, together with the pertinent channel observation LLRs, are stored and made available to the second decoder block for parallel processing. Incorporating auxiliary register blocks at strategic locations within the decoder's architecture facilitates the realization of pipelined decoding. In the exiting designs [8], [12], [24], [25] employing synchronous logic with pipelining, shared resources at specific decoding stages necessitate duplication to avoid calculation conflicts when processing multiple codewords concurrently. The number of repetitions and placement depends on the number of codewords processed simultaneously. Usually, in combinational decoder principles, resource sharing becomes unnecessary, eliminating the need for resource duplication. Instead, pipelined combinational decoders aim to maximize existing resources. This is achieved by incorporating storage elements to capture outputs from smaller combinational decoder components, and these stored outputs are then repurposed in decoding subsequent codewords.

As demonstrated in Fig. 4, each block presents a pipelined combinational decoder, where channel observation LLR vectors $(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4)$ are saved in the memory units of the auxiliary registers upon completion of the f-function pruning, until the time the decoding block completes the second half $(\hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7)$ LLR pruning. The decoding schedule for this pipelined combinational decoder is outlined in Fig. 7.

The decoding expressions for $N = 8$ are formulated using the connection of the trellis of the butterfly tree. Suppose in Fig. 6:

$$
\begin{aligned}
l'_0 &= f\left(l_0, l_1\right), \quad l'_1 = f\left(l_2, l_3\right), \\
l'_2 &= f\left(l_4, l_5\right), \quad l'_3 = f\left(l_6, l_7\right), \\
l''_4 &= g\left(l_0, l_1, \hat{u}_0 \oplus \hat{u}_1\right), \\
l''_5 &= g\left(l_2, l_3, \hat{u}_1\right), \\
l''_6 &= g\left(l_4, l_5, \hat{u}_2 \oplus \hat{u}_3\right), \\
l''_7 &= g\left(l_6, l_7, \hat{u}_3\right).
\end{aligned}
$$

The outputs are:

$$
\begin{aligned}
\hat{u}_0 &= s\left[f\left(f\left(l'_0, l'_1\right), f\left(l'_2, l'_3\right)\right)\right] \cdot a_0, \\
\hat{u}_1 &= s\left[g\left(f\left(l'_0, l'_1\right), f\left(l'_2, l'_3\right), \hat{u}_0\right)\right] \cdot a_1, \\
\hat{u}_2 &= s\left[f\left(g\left(l'_0, l'_1\right), g\left(l'_2, l'_3\right)\right)\right] \cdot a_2, \\
\hat{u}_3 &= s\left[g\left(g\left(l'_0, l'_1\right), g\left(l'_2, l'_3\right), \hat{u}_2\right)\right] \cdot a_3, \\
\hat{u}_4 &= s\left[f\left(f\left(l''_4, l''_5\right), f\left(l''_6, l''_7\right)\right)\right] \cdot a_4, \\
\hat{u}_5 &= s\left[g\left(f\left(l''_4, l''_5\right), f\left(l''_6, l''_7\right), \hat{u}_4\right)\right] \cdot a_5, \\
\hat{u}_6 &= s\left[f\left(g\left(l''_4, l''_5\right), g\left(l''_6, l''_7\right)\right)\right] \cdot a_6, \\
\hat{u}_7 &= s\left[g\left(g\left(l''_4, l''_5\right), g\left(l''_6, l''_7\right), \hat{u}_6\right)\right] \cdot a_7.
\end{aligned}
$$

When analyzing the scheduling of the butterfly-based SC decoder for $N = 8$ shown in Fig. 7, it becomes evident that pipelined combinational decoders, much like their non-pipelined counterparts, undertake the decoding of one codeword within each clock cycle. However, it's worth noting that the maximum path delay for a pipelined combinational decoder with a block length of N is roughly comparable to the delay of a combinational decoder with a block length of $N/2$. Consequently, the single-stage pipelined

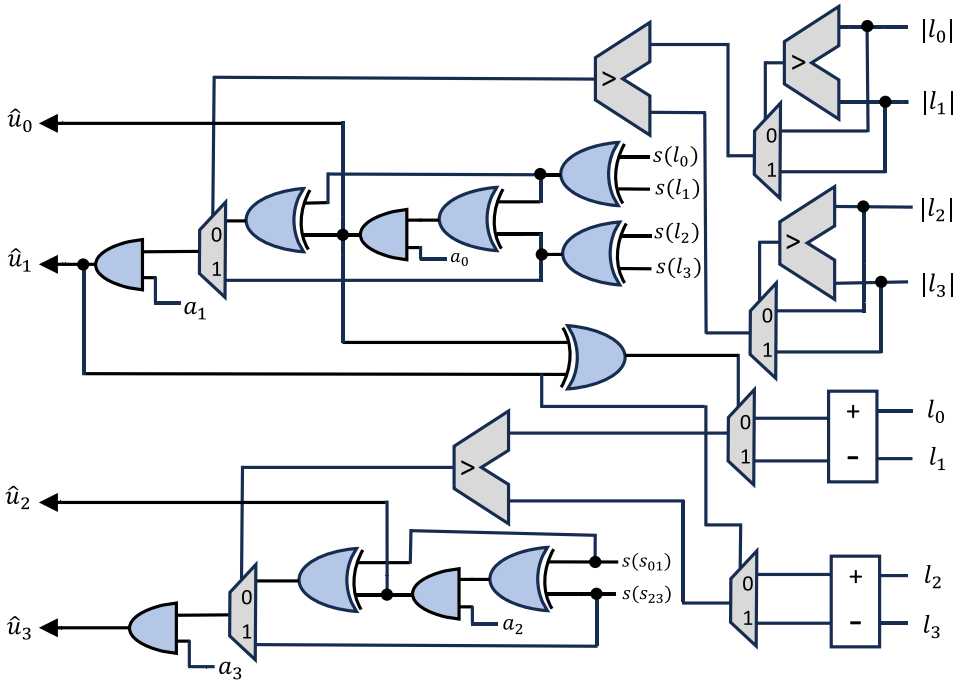


FIGURE 5. Implementation of the basic combinational decoder for N=4.

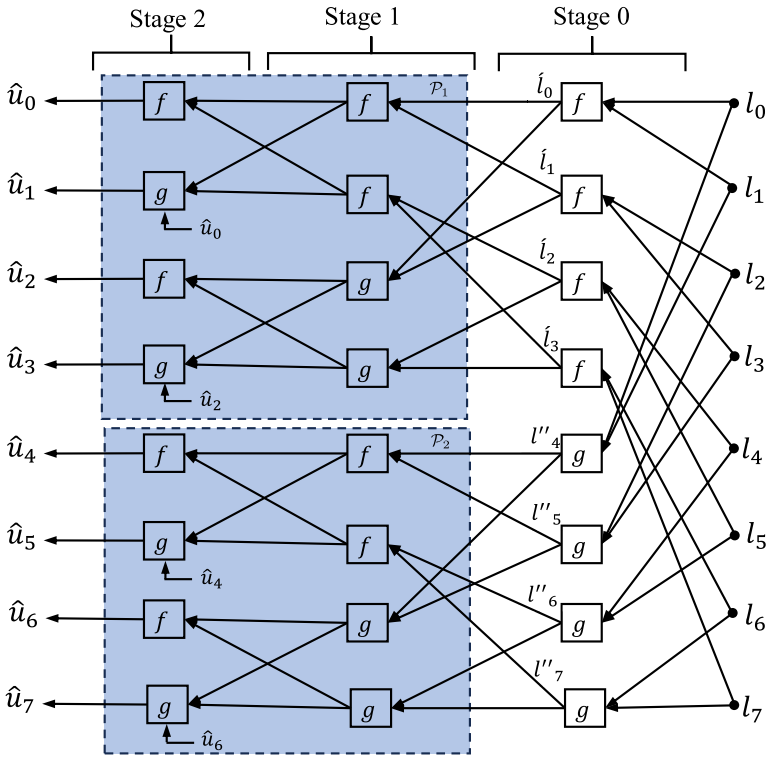


FIGURE 6. Butterfly tree for the decoder of length $N = 8$.

combinational decoder illustrated in Fig. 6 achieves nearly twice the throughput of a combinational decoder with an equivalent block length. However, this increase in throughput comes at the cost of heightened power consumption and an upswing in hardware utilization due to the inclusion of storage elements and the subsequent rise in operational frequency. The potential to elevate throughput further lies in expanding the number of pipelining stages and applying

similar pipelining techniques to the two combinational decoders of size $N/2$.

D. COMPOUND-LOGIC PIPELINED PROCESSING UNIT

In this section, we introduce an architecture that compounds both synchronous and combinational decoders to execute decoding operations for constituent codes. In the conventional sequential SC decoding process for polar codes, the decoder's speed diminishes as it nears the decision level. This level involves sequential decision-making and a reduction in parallel calculations. The compound-logic SC decoder accelerates on the inherent structure known as Generalized Concatenated Code (GCC) present in polar codes. This approach incorporates combinational decoding close to the decision level, effectively enhancing the efficiency of the SC decoder. The GCC structure is depicted in Fig. 6, where a polar code $\mathcal{P}$ with a length of $N = 8$ is composed of two polar codes, $\mathcal{P}_1$ and $\mathcal{P}_2$, each having a length of $\mathcal{P}' = N/2$.

The dashed boxes within the encoder diagram depict the component codes $\mathcal{P}_1$ and $\mathcal{P}_2$ derived from the parent code $\mathcal{P}$. Input bits for the first and second component codes are denoted as (u_0, u_1, u_2, u_3) and (u_4, u_5, u_6, u_7), respectively. The encoding procedure for $\mathcal{P}$ encompasses encoding the two groups separately using encoders configured for a block length of 4. This yields coded outputs (x_0, x_1, x_2, x_3) and (x_4, x_5, x_6, x_7), correspondingly. In the case of a polar code with a block length of 8 and a code rate of $R = 1/2$, the frozen bits are u_0, u_1, u_2, and u_4. Consequently, 3 input bits of $\mathcal{P}_1$ and 1 input bit of $\mathcal{P}_2$ are frozen. This results in $\mathcal{P}_1$ being a code with $R = 3/4$ and frozen bits u_0, u_1, u_2, while $\mathcal{P}_2$ becomes a code with $R = 1/4$ and frozen bit u_0.

The decoding procedure of $\mathcal{P}$ follows a reverse path to its encoding. Fig. 6 provides a depiction of the decoding tree graph for the block length $N = 8$. To decode the component codes $\mathcal{P}_1$ and $\mathcal{P}_2$, two separate decoding sessions for block length 4 are required. The LLRs for the input of the component codes are represented as $\left(l'_0, l'_1, l'_2, l'_3\right)$ and $\left(l''_4, l''_5, l''_6, l''_7\right)$. These inputs are computed via operations at stage 0. The frozen bit indicator vector for $\mathcal{P}$ is $\boldsymbol{a} = (0, 0, 0, 1, 0, 1, 1, 1)$, while for the first and second component codes, the vectors are $(0, 0, 0, 1)$ and $(0, 1, 1, 1)$, respectively. Notably, the input to the second decoder block $\left(l''_4, l''_5, l''_6, l''_7\right)$ is dependent on the decoded outputs of $\mathcal{P}_1$, as g functions are employed to compute the output based on input LLRs in conjunction with them.

The dashed boxes depicted in Fig. 6 encompass operations carried out by a combinational decoder for $\mathcal{P}' = 4$, whereas operations situated outside these boxes are conducted by a synchronous decoder.

The operational sequence of this integrated logic decoder works in this manner: Initially, a synchronous decoder employs channel observation LLRs to calculate intermediate LLRs at stage 0 without the need for partial sums. Once the synchronous decoder concludes the calculations for stage 0, the derived intermediate LLRs are fed into a combinational decoder designed for block length 4 each. This particular decoder produces $\hat{u}_0, \ldots, \hat{u}_3$ (representing the decoded bits of the first component code), prompting the synchronous decoder to pause for a duration equivalent to the maximum path delay of the combinational decoder. After this interval, the deciphered bits are conveyed back to the synchronous decoder for incorporation into partial sums ($\hat{u}_0 \oplus \hat{u}_1 \oplus \hat{u}_2 \oplus \hat{u}_3$, $\hat{u}_1 \oplus \hat{u}_3$, $\hat{u}_2 \oplus \hat{u}_3$, and $\hat{u}_3$). Using these partial sums alongside channel observation LLRs, the synchronous decoder computes intermediate LLRs, forwarding the calculated LLRs to the combinational decoder. In the combinational decoder, these LLRs contribute to the decoding of $\hat{u}_4, \ldots, \hat{u}_7$ (decoded bits of the second component code). The versatility of the introduced combinational decoder architecture allows adaptation to various code sets through the utilization of the frozen bit indicator vector input. This adaptability ensures that a sole combinational decoder proves adequate for the comprehensive task of decoding all bits.

While the combinational decoder is operational, the synchronous decoder remains idle for a duration of $T_{\mathcal{P}'} \times f_{op}$ clock cycles. Here, f_{op} denotes the operating frequency of the synchronous decoder, and $T_{\mathcal{P}'}$ represents the delay of a combinational decoder for block $\mathcal{P}_1$. The approximate reduction in latency achieved by the compound-logic decoder, compared to the corresponding synchronous decoder, can be estimated as follows:

let $L_S(\mathcal{P})$ denote the latency of a synchronous decoder for block length $\mathcal{P}$. The latency reduction in a single iteration for a component code of length $\mathcal{P}_1$ is given by $L_r(\mathcal{P}') = L_S(\mathcal{P}') - \left(T_{\mathcal{P}'} \times f_{op}\right)$. The latency reduction factor can then be approximated as

$$g(\mathcal{P}, \mathcal{P}') \approx \frac{L_S(\mathcal{P})}{L_S(\mathcal{P}) - \left(\frac{\mathcal{P}}{\mathcal{P}'}\right) \times L_r(\mathcal{P}')} \tag{8}$$

This latency reduction factor is applied to the throughput of the synchronous decoder, yielding.

$$T_{\cdot\text{put}_{(CL)}}(\mathcal{P}, \mathcal{P}') = g(\mathcal{P}, \mathcal{P}') \times T_{\cdot\text{put}_{(S)}}(\mathcal{P}),$$

where $T_{\cdot\text{put}_{(S)}}(\mathcal{P})$ and $T_{\cdot\text{put}_{(CL)}}(\mathcal{P})$ represent the throughputs of the synchronous and compound-logic decoders, respectively.

V. SIMPLIFIED CPPU ARCHITECTURE

A. CPPU BASED DECODER

The architecture of the basic decoder depicted in Fig. 8 is a basic combinational decoder specifically defined for the case $N = 4$. In the aforementioned design framework, the f and g functions involve operations of sign detection of LLR from the preceding stage and comparisons of values to sign detection of LLR at the final stage. Consequently, the need for additions, subtractions, and auxiliary processes at the decision stage is eliminated. These modifications resulted in increasing throughput and more efficient utilization of hardware resources. In the subsequent sections, we introduce a refined architecture for the f and g function simplifications

CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	⋯
S_0	$f_{0,1}$							$g_{0,4}$							
	$f_{0,1}$							$g_{0,5}$							
	$f_{0,2}$							$g_{0,6}$							
	$f_{0,3}$							$g_{0,7}$							
S_1		$f_{1,0}$			$g_{1,2}$				$f_{1,4}$			$g_{1,6}$			
		$f_{1,1}$			$g_{1,3}$				$f_{1,5}$			$g_{1,7}$			
S_2			$f_{2,0}$	$g_{2,1}$		$f_{2,2}$	$g_{2,3}$			$f_{2,4}$	$g_{2,5}$		$f_{2,6}$	$g_{2,7}$	
out			$\hat{u}_0$	$\hat{u}_1$		$\hat{u}_2$	$\hat{u}_3$			$\hat{u}_4$	$\hat{u}_5$		$\hat{u}_6$	$\hat{u}_7$	

FIGURE 7. Scheduling for the butterfly-based SC decoder with $N = 8$.

within the CPPU framework, employing logic gates to enhance the efficiency of addition and subtraction operations.

Instead of employing only four individual units for each LLR input, as illustrated in the basic decoder shown in Fig. 5, we present a design that introduces two CPPUs in conjunction with two comparators and several supplementary logic gates. To achieve this proposed modification, we can analyze the fundamental decoder to uncover simplified relationships. In this context, $a(i)$ signifies the frozen bit at position i in the array $\boldsymbol{a}$, s represents the sign function, and f' as well as g' denote the outcomes of the f and g functions, respectively. Referring to the expression defined for the basic combinational decoder, the output of $\hat{u}_0$ is formulated as follows:

$$\begin{aligned}
\hat{u}_0 &= s\left[f\left(f\left(l_0, l_1\right), f\left(l_2, l_3\right)\right)\right] \cdot a(0) \\
\hat{u}_0 &= s\left[f\left(s\left(l_0\right) s\left(l_1\right) \min\left(\left|l_0\right|, \left|l_1\right|\right)\right.\right. , \\
&\quad \left.\left. s\left(l_2\right) s\left(l_3\right) \min\left(\left|l_2\right|, \left|l_3\right|\right)\right)\right] \cdot a(0) \\
\hat{u}_0 &= \left[\left(s\left(l_0\right) \oplus s\left(l_1\right)\right) \oplus \left(s\left(l_2\right) \oplus s\left(l_3\right)\right)\right] \cdot a(0)
\end{aligned}$$

Taking advantage of the associative property of the XOR operation, we get,

$$\hat{u}_0 = \left(s\left(f'_{01}\right) \oplus s\left(f'_{23}\right)\right) \cdot a(0), \tag{9}$$

where, $s\left(f'_{01}\right) = \left(s\left(l_0\right) \oplus s\left(l_3\right)\right)$, and, $s\left(f'_{23}\right) = \left(s\left(l_2\right) \oplus s\left(l_1\right)\right)$.

Similarly for $\hat{u}_1$,

$$\begin{aligned}
\hat{u}_1 &= s\left[g\left(f\left(l_0, l_1\right), f\left(l_2, l_3\right), \hat{u}_0\right)\right] \cdot a(1) \\
\hat{u}_1 &= s\left[s\left(l_2\right) s\left(l_3\right) \min\left(\left|l_2\right|, \left|l_3\right|\right)\right. \\
&\quad \left. + (-1)^{\hat{u}_0} \cdot s\left(l_0\right) s\left(l_1\right) \min\left(\left|l_0\right|, \left|l_1\right|\right)\right] \cdot a(1) \\
\hat{u}_1 &= s\left(g\left(f'_{01}, f'_{23}, \hat{u}_0\right)\right) \cdot a(1)
\end{aligned}$$

$$\hat{u}_1 = s\left(f'_{23} + (-1)^{\hat{u}_0} \cdot f'_{01}\right) \cdot a(1) \tag{10}$$

A comparator is deployed to compute the smallest magnitude among $\left|f'_{01}\right|$ and $\left|f'_{23}\right|$. Then we received the signal from comparator $\mathcal{S}$, which is set high when $\left|f'_{01}\right| > \left|f'_{23}\right|$. Taking into account all conceivable combinations:

$$\hat{u}_1 = \begin{cases} s\left(f'_{23}\right) \cdot a(1), & \text{if } \mathcal{S} = 0 \\ \left(\hat{u}_0 \oplus s\left(f'_{01}\right)\right) \cdot a(1), & \text{if } \mathcal{S} = 1 \end{cases}$$

Similarly for $\hat{u}_2$,

$$\begin{aligned}
\hat{u}_2 &= s\left[f\left(g\left(l_0, l_1, \hat{u}_0 \oplus \hat{u}_1\right), \quad g\left(l_2, l_3, \hat{u}_1\right)\right)\right] \cdot a(2) \\
\hat{u}_2 &= \left[s\left(g'_{01}\left(\hat{u}_0, \hat{u}_1\right)\right) \oplus s\left(g'_{23}\left(\hat{u}_1\right)\right)\right] \cdot a(2)
\end{aligned}$$

Now,

$$\hat{u}_3 = s\left[g\left(g\left(l_0, l_1, \hat{u}_0 \oplus \hat{u}_1\right), \quad g\left(l_2, l_3, \hat{u}_1\right), \hat{u}_2\right)\right] \cdot a(3)$$

To ascertain the sign of this expression, a comparison between $\left|g'_{01}\right|$ and $\left|g'_{23}\right|$ is required. For this purpose, the signal $\mathcal{S}'$ is generated:

$$\hat{u}_3 = \begin{cases} s\left(g'_{23}\left(\hat{u}_1\right)\right) \cdot a(3), & \text{if } \mathcal{S}' = 0 \\ \left[\hat{u}_2 \oplus s\left(g'_{01}\left(\hat{u}_0, \hat{u}_1\right)\right)\right] \cdot a(3), & \text{if } \mathcal{S}' = 1 \end{cases}$$

These simplifications entail manipulation of the LLR's sign coming from the preceding operation and utilizing a comparator to determine the signs of the values at the decision level. This approach effectively eliminates the requirement for addition/subtraction operations at the final stage, as well as their related auxiliary processes. As a result of these simplifications, decoder latency is reduced, and hardware resources are better used. In each example, the last stage combines an AND operation with the associated frozen bit.

B. SIMPLIFIED CPPU

To optimize the hardware efficiency of the CPPU while accepting a slight extension in the functional path of the g function, we can simplify the output stages of the CPPU

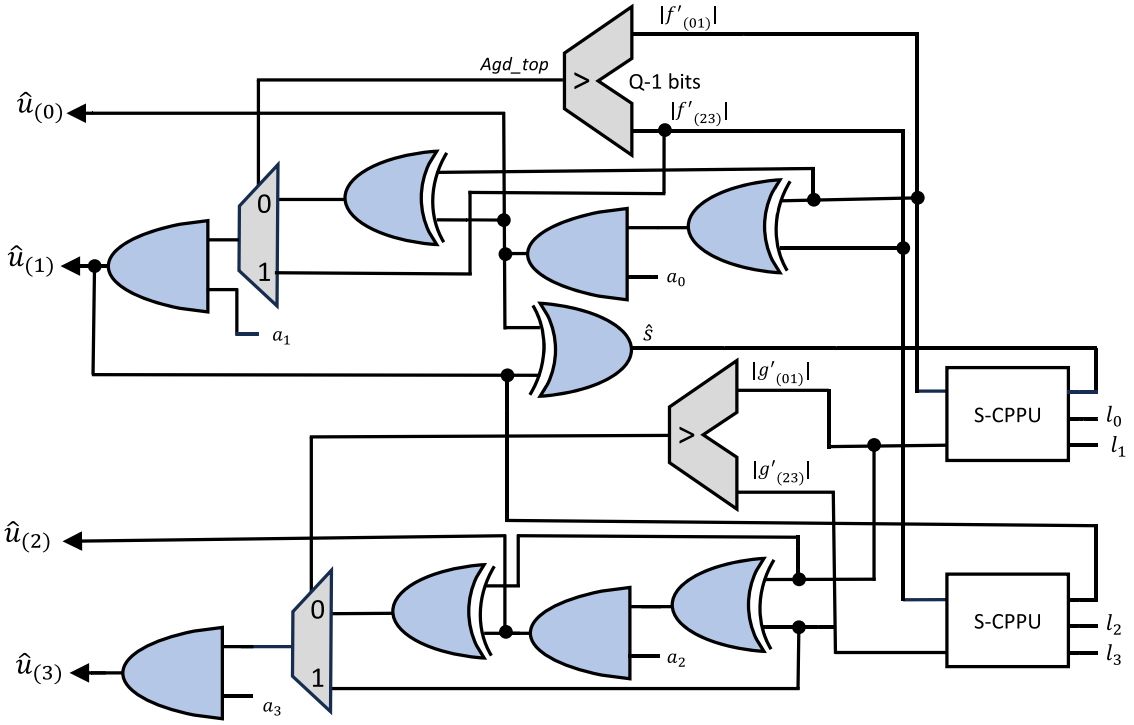


FIGURE 8. Demonstration of the simplified compound pipeline processing decoder for N=4, based on CPPU.

and subsequently implement it with the proposed auxiliary registers. The streamlined CPPU architecture illustrated in Fig. 9 comprises four key components: the Comparator module, Sign Detector, Adder/Subtractor unit, and Saturator. The role of the Comparator module is to ascertain the smaller magnitude among two input LLRs and provide it as $Max = \max(|X|, |Y|)$ and $Min = \min(|X|, |Y|)$ to the Adder/Subtractor unit. Additionally, this element computes the magnitude of the f function, corresponding to the minimum magnitude between the inputs. It also generates an internal indicator for the Sign Detector module, signaling the need for input swapping in situations where the smaller magnitude is detected. Thus, $\mathcal{S}_g = S_{(Y)}$ is valid when $|Y| > |X|$, otherwise $\mathcal{S}_g = S_{(X)}$. The binary sign function S for LLR l is $S(l) = 0$ when, $l \geq 0$ and $S(l) = 0$ otherwise.

The Sign Detector module derives the sign values of the two potential outcomes of the g function, denoted as $\mathcal{S}_g$, based on partial sum $\hat{s}$. Determination of the sign g involves considering all feasible combinations of input signs, magnitudes, and accurate outputs. If $S_{(Y)}$ represents the sign of the upper CPPU input, $S_{(X)}$ represents the sign of the lower CPPU input, and Ag is the flag produced by the Comparator module, $\mathcal{S}_g$ is expressed as follows:

$$\mathcal{S}_g = \overline{Ag}S(X) + AgS(Y)$$
$$\mathcal{S}'_g = \overline{Ag}S(X) + Ag\overline{S(Y)}$$

These formulations can be realized using two multiplexers, and due to the operation of $\mathcal{S}'_g = \mathcal{S}_g \oplus Ag$, one of them can be replaced with a separate XOR gate, simplifying the hardware complexity without affecting the effective path of the CPPU.

The g function's output is determined using a dual set of multiplexers in a two-step process. These multiplexers possess two inputs and one output, enabling the selection of the appropriate computed output value. In the initial step, a pair of multiplexers with a bit-size of $(Q-1)$ are utilized, and their choice signal relies on the signs of the input LLRs. The ultimate multiplexer, which is Q bits wide, employs the partial sum $\hat{s}$ as its selector. For enhanced hardware design efficiency, these three multiplexers can be merged into a single stage of simplified multiplexers within the architecture of a Simplified CPPU (S-CPPU), as illustrated in Fig. 9. Within this arrangement, the two multiplexers having $(Q-1)$ bit quantization, responsible for determining the magnitude of the g function's outcome, are substituted by a solitary multiplexer of $(Q-1)$ bit quantization. The connection between the selector signal of this newly suggested multiplexer and the output of the partial sum $\hat{s}$ is forwarded in the form of $\hat{s} \oplus d_f$. Introducing this selector signal requires including an XOR gate, which is a less complex element when contrasted with the multiplexer it substitutes, even slightly extending the decoding path. Steering the output's magnitude is overseen by a multiplexer of Q bit quantization, whereas in the original CPPU, the sign is exchanged for a straightforward multiplexer of one-bit quantization, within the S-CPPU, where this multiplexer utilizes $\hat{s}$ as its selector, efficiently providing the precise sign for the function g.

By adopting these simplifications, we have the opportunity to substitute the multiplexer having quantization of Q bit & two $(Q-1)$ bit in the initial CPPU set-up with just a single multiplexer of $(Q-1)$ bit, a 1-bit multiplexer, and an XOR gate within the S-CPPU. However, the S-CPPU brings about efficiency improvements for decoder implementations constrained by hardware limitations.

VI. HARDWARE IMPLEMENTATIONS ON FPGA

The hardware realization and synthesis of the proposed decoder were conducted using the targeted platform of the Virtex UltraScale - XCVU190 Field Programmable Gate Array (FPGA). The Virtex UltraScale XCVU190 is a high-end FPGA manufactured by Xilinx, known for its high performance and capabilities, and is often used in applications that require significant parallelism, low latency, rich resources such as logic elements, and DSP blocks. The device utilizes a maximum of 5.5 million system logic cells, employing a 20nm process technology with a 2nd generation 3D integrated circuit design. This design includes integrated cores for both 100G Ethernet MAC and 150G Interlaken communication protocols. Due to its higher processing power and parallelism, the Virtex UltraScale XCVU190 FPGA is likely to exhibit lower latency in polar code decoding compared to the Intel DE10-Standard used in [15]. The VHDL language is used, and the decoder was constructed through a recursive programming approach [43]. The placing and routing are performed by placing logic elements such as Lookup Tables (LUTs), Flip-Flops (FFs), and memory blocks onto the target XCVU190 FPGA and establishing the necessary connections between these elements to create a functional and efficient digital circuit. Afterward, synthesis outcomes were obtained using Xilinx Vivado 2022.2. A demonstration of the implementation outline of the proposed CPPU-based decoder on the targeted platform is illustrated in Fig. VI.

In the context of combinational decoders, FFs are employed to manage simple logic circuits and retrieve outputs from memory storage. In the scenario of pipelined decoders, FFs also play a role in retaining input LLRs and partial sums that play a crucial part in the decoding process of the second constituent code. It is evident that the operational capacity of combinational decoders experiences a substantial decline in throughput during FPGA implementation. This reduction is attributable to the significant delays introduced by routing complexities inherent in the FPGA realization of combinational decoders, contributing to a substantial proportion, potentially up to 90%, of the overall delay. Pipelined combinational decoders can achieve data transfer rates in the magnitude of gigabits per second, with the trade-off of employing a greater number of FFs. By introducing additional pipeline stages, it's possible to enhance the throughput, although this comes at the expense of using more FFs.

The synthesis outcomes for combinational encoder with different code lengths are presented in Table 1. Resource consumption is quantified in terms of the necessary Adaptive Logic Modules (ALMs) and LUTs. The expansion of ALMs and LUTs follows a nearly linear pattern as the code lengths increase. The count of 1-bit registers is directly linked to the input/output storage demands of the encoder. As the hardware intricacy grows, the highest achievable operational frequency declines, yet the throughput gets better due to the balancing impact of larger values of N.

TABLE 1. Synthesis outcomes for the encoder on the Xilinx XCVU190 FPGA with varying code lengths *N*., where *Rgtrs* are the registers, and *T.put* is the throughput.

N	ALMs	LUTs	$Rgtrs$ (bits)	f_{max} (MHz)	$T.put$ (Mbps) $\cdot 10^3$
2^4	24	21	32	850	9.34
2^5	58	45	64	696	15.56
2^6	128	166	128	541	22.68
2^7	278	258	256	412	55.26
2^8	595	653	512	395	74.12
2^9	1292	1368	1024	211	114.03
2^{10}	2784	2942	2048	120	202.83
2^{11}	6344	6224	4096	149	410.63

Table 2 displays the synthesis outcomes for the combinational CPPU and its simplified version of CPPU, utilizing different quantization lengths Q-bits for LLR. Additionally, in the next column, the table showcases the results of the CPPUs we designed. This enhancement is attributed to the synthesis tool's more effective utilization of available LUTs within the ALMs when organizing the logic of our designs for that specific quantization Q, employing functions with more inputs. The disparity in the count of LUTs between the original and simplified CPPUs is minimal. However, the number of ALMs changes based on the chosen Q value. The throughput results demonstrate negligible differences when considering maximum frequency values. Notably, it's crucial to mention that the logic within the adder and subtractor block is entirely executed using LUTs. Outcomes of S-CPPU don't consistently display resource reductions compared to the standard design due to the synthesis procedure and the FPGA's ALMs and LUTs. Nevertheless, more favorable outcomes might be anticipated when focusing on ASICs, where the translation of logic to hardware is more straightforward.

As detailed in the above sections, the enhancement of a combinational decoder's throughput is achievable by merging it with a synchronous decoder, leading to an augmentation of magnitude denoted by $g\left(\mathcal{P}, \mathcal{P}'\right)$ as articulated in equation (8). Within this segment, we deliver analytical computations for evaluating the throughput of a compound-logic decoder.

Table 3 presents a comparison of the outcomes from the placing and routing process for the proposed decoder in comparison with the referred design featuring identical combinational architectures. These alternative designs employ the SM notation for LLRs with quantization kept at 5 bits [13], [14], [15]. The throughput values we have calculated are

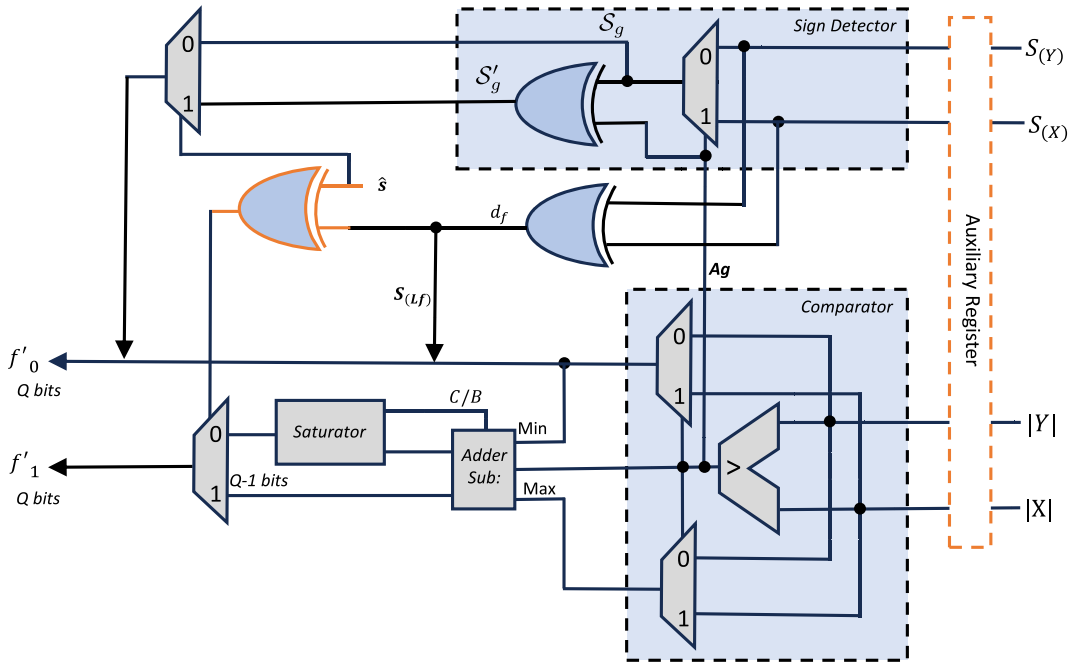


FIGURE 9. Simplified compound pipeline processing decoder for $N = 2$.

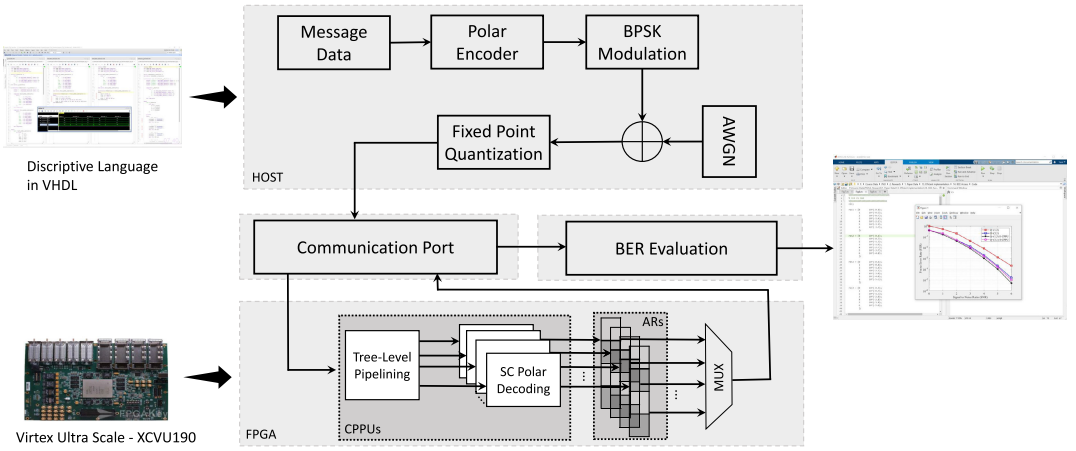


FIGURE 10. Implementation outline of the proposed CPPU-based decoder on the targeted platform of Virtex Ultra Scale - XCVU190 FPGA.

obtained based on the maximum predicted frequencies. Additionally, it becomes evident that the multiplicative enhancement escalates proportionally with the growth of the combinational decoder's size. The degree of this escalation is contingent upon the parameter $\mathcal{P}$, as it defines the decoding stage where the count of constituent computations reduces below the available hardware resources, instigating a bottleneck in throughput. Importantly, it should be emphasized that the extent of this gain might be more restrained for decoders that consume fewer clock cycles during the concluding phases of the decoding trellis. Notably, S-CPPU decoders demonstrate notable utility, particularly in the context of short codewords decoding, where a combinational architecture's hardware utilization is substantial, and synchronous decoders entail heightened latency.

A. ASSESSMENT OF THE BIT ERROR RATE

Following the successful implementation of the decoder, a series of tests were executed to validate the system's BER efficiency. Frozen bits were determined based on a specific signal-to-noise ratio (SNR) relevant to the Additive White Gaussian Noise (AWGN) channel. These computations utilized the sequential algorithm and were guided by the Bhattacharyya parameter in the context of these tests. Messages were subjected to examination using the Monte Carlo technique along with Binary Phase Shift Keying

TABLE 2. The synthesis outcomes for combinational decoders utilizing Compound-Logic and S-CPPUs at $N = 16$ on the designated Xilinx XCVU190 FPGA platform are compared across various LLR quantization bit values Q.

Q	Compound-Logic ($\mathcal{P}' = N/2$)			S-CPPU ($\mathcal{P}' = N/2$)		
	ALMs	LUTs	$f_{\max}$ (MHz)	ALMs	LUTs	$f_{\max}$ (MHz)
5	9	24	360	9	18	367
6	18	38	351	19	48	344
7	20	43	295	26	55	290
8	24	49	270	38	60	218
16	46	118	239	58	158	154
32	61	208	180	74	277	120
64	63	322	168	90	402	101

(BPSK) modulation. Each received sample, denoted as y_i, underwent a processing procedure to determine the LLRs obtained from the AWGN channel. This computation was carried out using the following expressions:

$$r_k = -(2x_k - 1) + w_k$$

where, r_k represents the received signal sample at time k, x_k is the transmitted bit of the code word at time k. It's a binary value (0 or 1) indicating the transmitted symbol. The term $2x_k - 1$ is a mapping from binary values to -1 or 1, where $2x_k - 1 = -1$ when $x_k = 0$ and $2x_k - 1 = 1$ when $x_k = 1$. w_k is the noise component added to the received signal. It represents the effect of noise in the channel.

$$w_k = \sigma \times \nabla_k$$

w_k is the noise component at time k, σ is the standard deviation of the noise. It is calculated using the parameters of the communication system, ∇_k is a randomness. This randomness simulates the noise in the communication channel.

$$\sigma = \frac{1}{\sqrt{2 \times R \times \left(\frac{E_b}{N_0}\right)}} \tag{11}$$

R is the code rate, E_b/N_0 is the per-bit energy to noise power spectral density ratio. It's a measure of signal quality. Finally,

$$l_k(r_k) = \frac{2r_k}{\sigma^2} = 4r_k R_c 10^{\frac{SNR}{10}} \tag{12}$$

$l_k(r_k)$ is the LLR associated with the received signal r_k, essentially comparing the received signal with the expected noise level.

The SM quantization within S-CPPU utilizing $Q = 5$ bits introduces an upper limit to the representation range, leading to an investigation into performance outcomes for different bit configurations of the integer i and decimal f in the $Q_{i,f}$ values. It's worth noting that when maintaining a constant Q value, augmenting the integer bits enhances the saturation limit's magnitude at the expense of reduced precision. Fig. 11 presents the BER curve for a configuration of $N = 128$ and $K = 64$ within the devised system. This performance

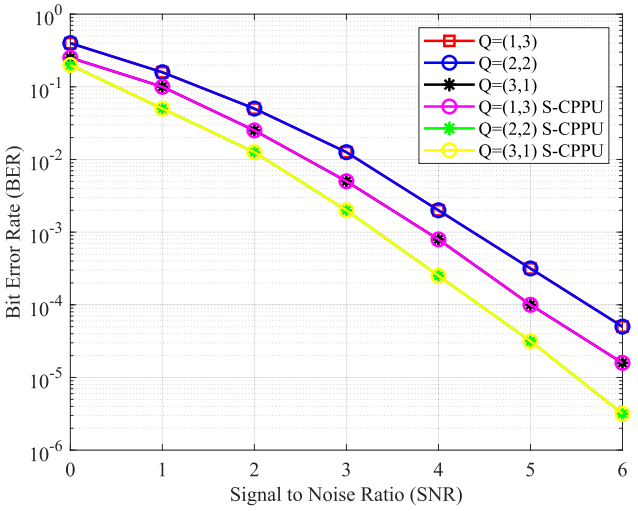


FIGURE 11. Assessing the BER for $N = 128$ while varying $Q(i, f)$, where Q comprises an integer component i and a fractional part f.

is compared with software simulations utilizing fixed-point representations. Optimal quantization is realized through $Q_{3,1}$ selection, as it diminishes the likelihood of saturation occurrences and curtails the potential for LLR saturation as SNR escalates. On the other hand, if LLR magnitudes are high, quantifying the integer part with more bits, even at the cost of resolution, becomes important.

Fig. 12 and Fig. 13 illustrate the BER and FER outcomes for distinct quantized fixed-point representations, employing both methods to calculate LLRs. Notably, the conventional LLR strategy with $Q_{1,3}$ demonstrates poor performance, while the BER for the $Q_{3,1}$ curve closely approximates that of the estimation of LLR in S-CPPU. The LLR estimation in S-CPPU is simplified in hardware by obviating the requirements for measuring channel noise levels. Additionally, a comparison of Equations (11) and (12) reveals that excluding SNR and code rate from the equation reduces the computational complexity of this simplified LLR calculation.

Moreover, the integration of the simplified LLR S-CPPU approach with the $Q_{1,3}$ representation demonstrates the most optimal BER performance. Considering that the LLR in S-CPPU involves a simple multiplication of received samples by a fixed n for 2^n, there is a possibility to significantly reduce the computational requirements for LLR calculations in both software and hardware. The enhancements seen in scenarios with elevated SNR values stem from the insignificance of noise, thereby favouring heightened resolution over an extended dynamic range in such contexts.

B. TESTING AND ANALYSIS

For the implementation of the encoder/decoder system and simulating communication over channels, we utilized the resource-rich platform of Virtex UltraScale XCVU190. The FPGA board can then operate autonomously at high speed, storing results in a file and providing feedback through simple

TABLE 3. Comparing the synthesis outcomes for the decoder on the Xilinx XCVU190 FPGA with different FPGAs across various code lengths N and referencing the relevant works at $Q = 5$.

N	Xilinx XCVU190 FPGA								Referred Work								
	Compound-Logic ($\mathcal{P}' = N/2$)				S-CPPU ($\mathcal{P}' = N/2$)				SP Badar [13]			YZ Fan [14]			FG Krasser [15]		
	LUTs	FFs	RAM (bits)	$T.put$ (Mbps)	LUTs	FFs	RAM (bits)	$T.put$ (Mbps)	LUTs	RAM (bits)	$T.put$ (Mbps)	LUTs	RAM (bits)	$T.put$ (Mbps)	LUTs	RAM (bits)	$T.put$ (Mbps)
2^4	1460	261	114	2672	1373	204	110	2565	722	127	1245	-	23	105	625	112	626
2^5	1906	358	212	2326	1645	330	206	2291	2231	233	1201	-	23	103	1682	224	565
2^6	4991	428	490	2191	4627	401	458	2120	5265	485	1169	4317	23	100	4308	448	532
2^7	13289	629	867	1932	11206	501	760	1880	14456	796	1042	4321	23	101	9585	896	473
2^8	33162	1631	1648	1837	29312	1411	1520	1690	33432	1621	996	3977	23	100	22782	1792	430
2^9	76044	3886	3478	1750	71696	3483	3378	1537	80535	3479	658	4448	46	99	-	-	-
2^{10}	192458	7118	8032	1612	167640	6512	7035	1396	-	7155	744	3721	56	88	-	-	-

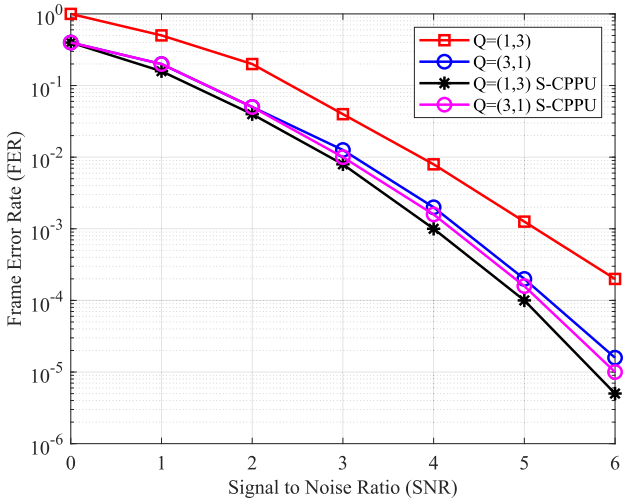


FIGURE 12. FER evaluations for $N = 128$ is examined in relation to $Q(i, f)$, with Q consists of an integer part i and a decimal part f, and keeping information bits to 64.

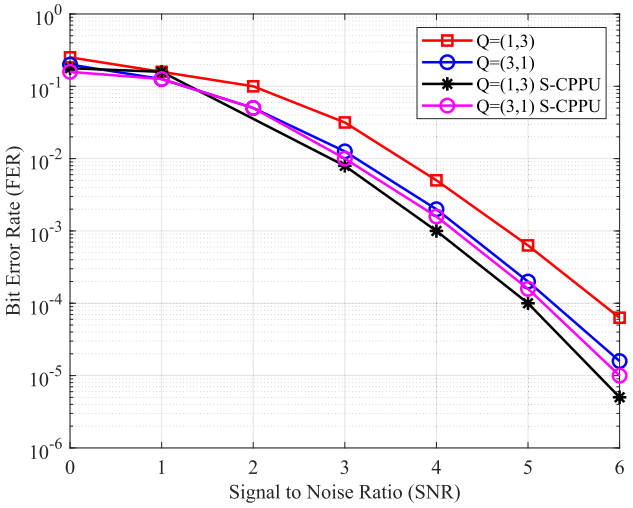


FIGURE 13. BER evaluations for $N = 128$ is examined in relation to $Q(i, f)$, with Q consists of an integer part i and a decimal part f, and keeping information bits to 64.

peripherals like LEDs. The processor's operating frequency of up to 1GHz and 14,490 Kb of maximum distributed RAM using the 20nm process technology, encompassing high serial I/O bandwidth and logic capacity.

When contrasted with the average test cycle duration per message, which amounts to 300 μs, the encoding duration of 11.5 ns holds minimal significance. Concerning the decoder, the effective operational frequency surpasses the delay estimate of the synthesis tool by nearly twice the amount. Consequently, the actual throughput achieved by the decoder exceeds the performance of the comparative study delineated in Table 3. Moreover, it's worth emphasizing that roughly 71% of the designated area resources are utilized for the encoder and decoder modules, leaving 29% allocated for the FPGA interfaces and indispensable control logic.

In order to carry out the experimentation, the setup was tested in a mode where the upper limit of the count was established through switches available in physical form, functioning as a frequency splitter. This count was systematically decreased until instances of decoding errors became apparent. The testing process was executed under optimal channel conditions, simulating a high signal-to-noise ratio (SNR) scenario in an Additive White Gaussian Noise (AWGN) channel. By operating the decoder at lower frequencies, the error correction performance was assessed across various SNR levels. The outcomes of these experiments were recorded, revealing a strong alignment with the simulation results. The frequency divider was then configured to the maximum clock frequency. The decoder's error-correcting performance was tested and graphed against the maximum frequency to confirm its nominal operation. When compared to the reference combinational decoder, the experimentally realized decoder throughput was much higher, by roughly 17.34%.

To demonstrate the impact of our tree-level CPPU proposed, we assessed the achieved decoding latency in terms of processing cycles in Fig. 13. For a fair evaluation, we employed the proposed pipelined decoder architecture to assess different decoding strategies. This architecture comprises 16 decoding blocks, each equipped with 64 CPPUs, configured to 1024-bit polar codes. It's worth noting that the introduced tree-level CPPU, coupled with single-cycle CPPU units, significantly contributes to achieving low latency in SC decoding across various code rates. This improvement is particularly notable when compared to serialized algorithms [9] and [15]. The S-CPPU-based design

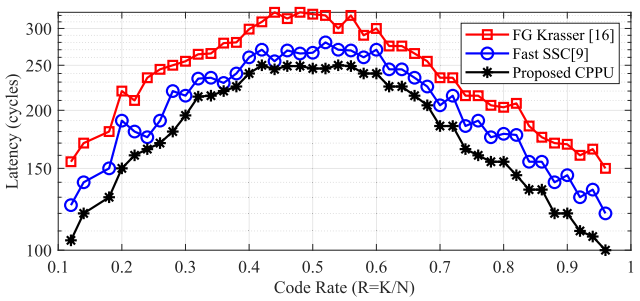


FIGURE 14. Assessments of latency for various SC decoding algorithms within the pipelined decoder framework, featuring 16 decoding blocks and 64 CPPUs for $N = 1024$ length code.

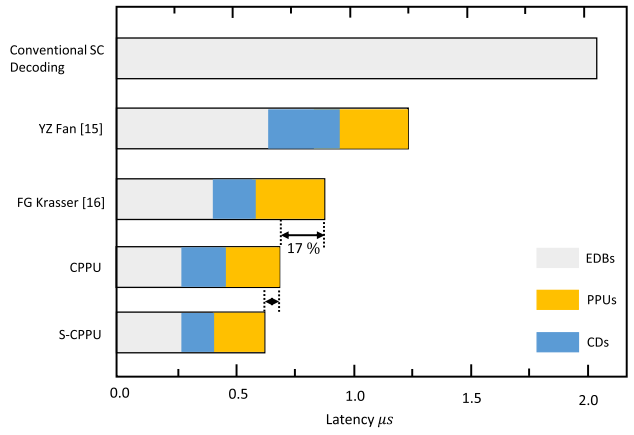


FIGURE 15. Latency Improvement Analysis: Assessing the latency gain achieved by the proposed decoding algorithms within the pipelined decoder framework, comparing them with conventional SC decoding, as well as other referenced parallel and merged decoding methodologies.

goes a step further in reducing decoding cycles by enabling the concurrent update of partial-sum registers within a single clock cycle. Consequently, the fully optimized parallel SC decoder requires only 143 clock cycles to decode a 0.5-rate 1024-bit. This represents a speedup of 17.34 and 1.32 times compared to conventional SC decoding [15] and our [9], respectively.

A similar illustration is also depicted in Fig. VI-B, when compared to the reference combinational decoder, the experimentally realized decoder throughput has improved by 17.34% compared to work [16].

VII. CONCLUSION

This paper proposes a compound logic and its simplified counterpart SC encoder and decoder architecture designs. These architectures combine a combinational SC decoder with a synchronous SC decoder and demonstrate notable advantages over the conventional sequential decoding algorithm in terms of low latency and resource consumption. The proposed approach is implemented on a designated Xilinx XCVU190 FPGA platform. The synthesis results reveal that the simplified combinational architectures are capable of achieving a throughput of approximately 2672 Mbps for a code rate of 2^4. The architecture's flexibility is highlighted as it can incorporate additional pipelining stages at varying depths to enhance throughput along with the auxiliary registers. The simplified counterpart reduces resource requirements for specific LLR quantization, even with slightly extended latencies, with a notable 17% reduction in LUTs consumption, particularly significant for the commonly used 5-bit LLR quantization. Within this compound structure, the combinational part acts as an accelerator for the synchronous decoder, effectively boosting throughput while keeping complexity manageable. Compared to the reference designs, our decoder improves speed by about 17.34% for a 128-bit code length. Resource allocation within the system is distributed approximately 71% to the encoding and decoding blocks, with the remaining 29% allocated to the processor interface and control logic. Experiments evaluating error-correcting performance emphasize the importance of LLR quantization and bit arrangements in both conventional and simplified computational cases.

A prospective avenue for further exploration in our proposed approach involves the practical implementation of both the CPPU-based decoder and its simplified counterpart in a detailed case study application.

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] Y. Ren, A. T. Kristensen, Y. Shen, A. Balatsoukas-Stimming, C. Zhang, and A. Burg, "A sequence repetition node-based successive cancellation list decoder for 5G polar codes: Algorithm and implementation," *IEEE Trans. Signal Process.*, vol. 70, pp. 5592–5607, 2022.

[3] Z. Liu, R. Liu, and H. Zhang, "High-throughput adaptive list decoding architecture for polar codes on GPU," *IEEE Trans. Signal Process.*, vol. 70, pp. 878–889, 2022.

[4] Y. Ali, Y. Xia, L. Ma, and A. Hammad, "Secure design for cloud control system against distributed denial of service attack," *Control Theory Technol.*, vol. 16, pp. 14–24, Feb. 2018.

[5] C. Yan, Y. Cui, K. Chen, B. Wu, and W. Liu, "Hardware efficient successive-cancellation polar decoders using approximate computing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 189–200, Mar. 2023.

[6] A. Ç. Arli and O. Gazi, "A survey on belief propagation decoding of polar codes," *China Commun.*, vol. 18, no. 8, pp. 133–168, Aug. 2021.

[7] H. Rezaei, N. Rajatheva, and M. Latva-Aho, "High-throughput rate-flexible combinational decoders for multi-kernel polar codes," 2023, *arXiv:2301.10445*.

[8] 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.

[9] 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.

[10] 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.

[11] I. Tal and A. Vardy, "List decoding of polar codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2011, pp. 1–5.

[12] V. Bioglio, C. Condo, and I. Land, "Design of polar codes in 5G new radio," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 29–40, 1st Quart., 2021.

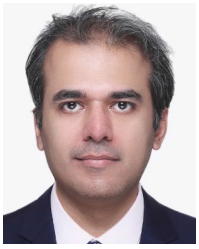
[13] S. P. Badar and K. Khanchandani, "Successive cancellation polar decoder implementation using processing elements," in *Proc. IEEE Region 10 Symp. (TENSYMP)*, Jul. 2022, pp. 1–6.

[14] 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.
[15] F. G. Krasser, M. C. Liberatori, L. Coppolillo, L. Arnone, and J. C. Moreira, "Fast and efficient FPGA implementation of polar codes and soc test bench," *Microprocessors Microsyst.*, vol. 84, Jul. 2021, Art. no. 104264.
[16] 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.
[17] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *J. Signal Process. Syst.*, vol. 69, pp. 305–315, Dec. 2012.
[18] 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.*, May 2011, pp. 1665–1668.
[19] 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.
[20] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725–728, Apr. 2013.
[21] 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.
[22] 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.
[23] 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.
[24] C. Zhang, J. Yang, X. You, and S. Xu, "Pipelined implementations of polar encoder and feed-back part for SC polar decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 3032–3035.
[25] E. Arıkan, "Polar codes: A pipelined implementation," in *Proc. 4th Int. Symp. Broad. Commun. (ISBC)*, 2010, pp. 11–14.
[26] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 3471–3475.
[27] 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.
[28] G. Berhault, C. Leroux, C. Jego, and D. Dallet, "Partial sums generation architecture for successive cancellation decoding of polar codes," in *Proc. SiPS*, Taipei, Taiwan, Oct. 2013, pp. 407–412.
[29] G. Berhault, C. Leroux, C. Jego, and D. Dallet, "Partial sums computation in polar codes decoding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 826–829.
[30] G. Berhault, C. Leorux, and C. Jego, "Memory requirement reduction method for successive cancellation decoding of polar codes," *J. Signal Process. Syst.*, vol. 88, no. 3, pp. 425–438, 2016.
[31] P. Giard, G. Sarkis, and C. Thibeault, "A 237 Gbps unrolled hardware polar decoder," *Electron. Lett.*, vol. 51, no. 10, pp. 762–763, 2014.
[32] P. Giard, G. Sarkis, and C. Thibeault, "Multi-mode unrolled hardware architectures for polar decoders," *IEEE Trans. Circuits Syst.*, vol. 63, no. 9, pp. 1443–1453, Aug. 2016.
[33] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Proc. 48th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2014, pp. 2116–2120.
[34] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Flexible and low-complexity encoding and decoding of systematic polar codes," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2732–2745, Jul. 2016.
[35] M. S. Oommen and S. Ravishankar, "FPGA implementation of an advanced encoding and decoding architecture of polar codes," in *Proc. Int. Conf. VLSI Syst., Archit., Technol. Appl. (VLSI-SATA)*, Bangalore, India, Jan. 2015, pp. 1–6.
[36] K. Niu and K. Chen, "Stack decoding of polar codes," *Electron. Lett.*, vol. 48, no. 12, pp. 695–697, Jun. 2012.
[37] Z. Piao, C.-M. Kim, and J.-G. Chung, "An efficient list successive cancellation decoder for polar codes," *J. Semiconductor Technol. Sci.*, vol. 16, no. 5, pp. 550–556, Oct. 2016.
[38] C. Xia, J. Chen, Y. Fan, C.-Y. Tsui, J. Jin, H. Shen, and B. Li, "A high-throughput architecture of list successive cancellation polar codes decoder with large list size," *IEEE Trans. Signal Process.*, vol. 66, no. 14, pp. 3859–3874, Jul. 2018.
[39] B. Le Gal, Y. Delomier, C. Leroux, and C. Jégo, "Low-latency sorter architecture for polar codes successive-cancellation-list decoding," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2020, pp. 1–5.
[40] S. A. Hashemi, M. Mondelli, S. H. Hassani, C. Condo, R. L. Urbanke, and W. J. Gross, "Decoder partitioning: Towards practical list decoding of polar codes," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3749–3759, Sep. 2018.
[41] D. Kim and I.-C. Park, "A fast successive cancellation list decoder for polar codes with an early stopping criterion," *IEEE Trans. Signal Process.*, vol. 66, no. 18, pp. 4971–4979, Sep. 2018.
[42] X. Dong, R. Liu, and Z. Huang, "Fast simplified multi-bit successive-cancellation list decoding of polar codes and implementation," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2019, pp. 1–5.
[43] P. Ashenden, "Recursive and repetitive hardware models in VHDL," Dept. Elect. Eng. Comput. Sci., Cincinnati Univ., Tech. Rep. TR160/12/93/ECE, 1993.

YASIR ALI received the B.Sc. degree in electrical engineering (communication) from the University of Engineering and Technology, Peshawar, Pakistan, and the M.S. degree in control science and engineering from the School of Automation, Beijing Institute of Technology, China. He is currently pursuing the Ph.D. degree in secure and efficient implementation of the polar decoder. His current research interests include polar decoding, cyber-physical system security, and unmanned aerial vehicles. He received the Best Paper Award 2019 from the *Control Theory and Technology* journal (Springer).

YUANQING XIA (Fellow, IEEE) received the Ph.D. degree in control theory and control engineering from Beihang University (previously known as Beijing University of Aeronautics and Astronautics), Beijing, China, in 2001.

He was a Research Fellow in several academic institutions, from 2002 to 2008, including the National University of Singapore and the University of Glamorgan, U.K. Since 2004, he has been with the Beijing Institute of Technology, China, where he is currently a Full Professor. He is also the President of the Zhongyuan University of Technology. His research interests include cloud control systems, networked control systems, robust control and signal processing, active disturbance rejection control, and flight control. He is the Director of the specialized committee on cloud control and decision of the Chinese Institute of Command and Control (CICC), a member of the 8th Disciplinary Review Group of the Academic Degrees Committee of the State Council, a member of the Big Data Expert Committee of the Chinese Computer Society, and the Vice Chairperson of the Internet of Things Working Committee of the Chinese Institute of Instrumentation. He is a Deputy Editor of the *Journal of Beijing Institute of Technology*, an Associate Editor of *Acta Automatica Sinica*, *International Journal of Automation and Computing*, *Gyroscopy and Navigation*, and *IET Control Theory and Applications*.



TAYYAB MANZOOR received the M.S. degree in control and signal processing from the University of Leicester, Leicester, U.K., in 2011, and the Ph.D. degree in control science and engineering from the Beijing Institute of Technology, China, in 2021. From 2011 to 2016, he was a Lecturer and an Assistant Professor with the University of South Asia, Lahore, Pakistan, and the Imperial College of Business Studies. From June 2021 to December 2023, he was a Postdoctoral Fellow with the School of Automation Science and Engineering, South China University of Technology, Guangzhou, China, where his project was directly funded by the Ministry of Science and Technology of China under the Foreign Young Talent Program (2022 National Foreign Expert Project). He is currently a Faculty Member with the School of Automation and Electrical Engineering, Zhongyuan University of Technology, Zhengzhou, China. His current research interests include model predictive control, machine learning techniques, and their applications.

SHAHZAD ALI received the master's degree in control theory and control engineering from North China Electric Power University, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree in control science and engineering with the Beijing Institute of Technology, Beijing. His research interests include machine learning, load frequency control, and control algorithms.

MOHAMED ABOUHAWWASH received the B.Sc. and M.Sc. degrees in statistics and computer science from Mansoura University, Mansoura, Egypt, in 2005 and 2011, respectively, and the joint Ph.D. degree in statistics and computer science with the Channel Program between Michigan State University, East Lansing, MI, USA, and Mansoura University, in 2015. In 2018, he was a Visiting Scholar with the Department of Mathematics and Statistics, Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada. He is currently with Michigan State University. He is also an Associate Professor with the Department of Mathematics, Faculty of Science, Mansoura University. His current research interests include evolutionary algorithms, machine learning, image reconstruction, and mathematical optimization. He was a recipient of the Best Master's and Ph.D. Thesis Awards from Mansoura University in 2012 and 2018, respectively.

S. S. ASKAR received the B.Sc. degree in mathematics and the M.Sc. degree in applied mathematics from Mansoura University, Egypt, in 1998 and 2004, respectively, and the Ph.D. degree in operation research from Cranfield University, U.K., in 2011. In 2012, he joined King Saud University, where he is currently a Professor with the Department of Statistics and Operation Research. Since 2016, he has been an Associate Professor with Mansoura University. His research interests include game theory and its applications, including mathematical economy, dynamical systems, and network analysis.

AMIT KRISHAN KUMAR (Member, IEEE) received the Ph.D. degree in control science and engineering from the Beijing Institute of Technology, Beijing, China, and the M.Sc. degree in engineering from The University of the South Pacific, Suva, Fiji. His research interests include pattern recognition, clustering, computer vision, respiratory systems modeling, quantum theories, cymatics, and bimodal intelligent systems that apply to health care, water purification, data science, and system optimization.

RUIFENG MA received the B.S. degree from Shandong University, in 2021. He is currently pursuing the M.S. degree with the Beijing Institute of Technology. His research interests include cloud-native technology, cloud resource management, and machine-learning operations.