Implementation of a High-Throughput Fast-SSC Polar Decoder with Sequence Repetition Node

Haotian Zheng, Alexios Balatsoukas-Stimming, Zizheng Cao, Ton Koonen

Department of Electrical Engineering, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands {h.zheng, a.k.balatsoukas.stimming, z.cao, a.m.j.koonen}@tue.nl

Abstract—Even though polar codes were adopted in the latest 5G cellular standard, they still have the fundamental problem of high decoding latency. Aiming at solving this problem, a fast simplified successive cancellation (Fast-SSC) decoder based on the new class of sequence repetition (SR) nodes has been proposed recently in [1] and has a lower required number of time steps than other existing Fast-SSC decoders in theory. This paper focuses on the hardware implementation of this SR node-based fast-SSC (SRFSC) decoder. The implementation results for a polar code with length 1024 and code rate 1/2 show that our implementation has a throughput of 505 Mbps on an Altera Stratix IV FPGA, which is 17.9% higher with respect to the previous work.

I. INTRODUCTION

Polar codes are the first provably capacity-achieving channel codes with an explicit construction, low-complexity encoding and decoding algorithms, and easily adaptable coding rate [2]. Although the capacity of binary symmetric memoryless channels can be achieved using the low-complexity successive cancellation (SC) decoding algorithm, the sequential nature of SC decoding typically leads to a large decoding latency, which constrains its application in high-throughput and low-latency communication scenarios such as 5G and optical wireless communications [3]. A simplified successive cancellation (SSC) decoder was proposed in [4], where fast decoding methods are described for subcodes of the polar code (called constituent codes) that consist either of only information bits or of only frozen bits. Following this idea, other constituent codes with special information bit patterns and their corresponding fast decoders were identified in [5]-[8]. The family of these decoding algorithms is often referred to as *fast-SSC* decoding. To increase the number of fast decoding constituent codes, [9], [10] altered the polar code construction to further improve latency at the cost of a small error-correcting performance degradation. Methods to optimize the memory footprint of fast-SSC decoders were described in [11].

The work of [1] proposed a new class of *sequence repetition* (SR) constituent codes, which is a generalization of most existing constituent codes. It was also shown that the decoding of SR constituent codes can be highly parallelized to achieve further latency reduction compared to the state of the art without tangibly affecting the error-correcting performance. However, the work of [1] only focused on the algorithmic aspects of SR constituent codes and no hardware implementation has been reported in the literature.

Contribution: In this work, we describe a hardware architecture for a fast-SSC decoder that exploits the SR constituent codes described in [1] and we provide FPGA implementation results. Even though our proposed implementation is not yet



1

Fig. 1. SC decoding tree representation of a polar code with N = 8.

highly optimized, it still achieves a 17.9% higher decoding throughput than the state of the art.

II. BACKGROUND

A. Polar Codes

A polar code with code length $N = 2^n$ and information length K is denoted by $\mathcal{P}(N, K)$ and has rate R = K/N. The input bit sequence u consists of K information bits whose positions form set \mathbb{A} and N - K frozen bits whose positions form \mathbb{A}^c . The values of the frozen bits are usually set to 0. The encoded bit sequence can be calculated as $x = u \mathbf{G}_N$, where $\mathbf{G}_N = \mathbf{R}_N \mathbf{F}_2^{\otimes n}$ is the generator matrix of the polar code, \mathbf{R}_N is a bit-reversal permutation matrix and $\mathbf{F}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$.

B. SC and Fast-SSC Decoding

1) Algorithm: SC decoding of polar codes can be represented as the traversal of a binary tree as in Fig. 1. The *i*-th node at level j $(1 \le i \le 2^{n-j})$ of the SC decoding tree corresponds to a constituent code with bit index from $2^j \cdot (i-1) + 1$ to $2^j \cdot i$, and is denoted as \mathcal{N}_j^i . The left and the right child nodes of \mathcal{N}_j^i are \mathcal{N}_{j-1}^{2i-1} and \mathcal{N}_{j-1}^{2i} , respectively. For \mathcal{N}_j^i , the symbol $\alpha_j^i[k]$, $1 \le k \le 2^j$, denotes the *k*-th input logarithmic likelihood ratio (LLR) value, and $\beta_j^i[k]$, $1 \le k \le 2^j$, denotes the *k*-th output binary hard-valued message. The SC decoding follows a depth-first principle, with priority to the left branch. When LLR messages pass to the left and right child nodes, f and g functions over the LLR domain are executed, respectively, which are given by

$$\alpha_{j-1}^{2i-1}[k] \approx \operatorname{sign}\left(\alpha_{j}^{i}[2k-1]\right) \operatorname{sign}\left(\alpha_{j}^{i}[2k]\right) \\ \cdot \min\left(\alpha_{i}^{i}[2k-1], \alpha_{i}^{i}[2k]\right),$$
(1)

$$\alpha_{j-1}^{2i}\left[k\right] = (-1)^{\beta_{j-1}^{2i-1}\left[k\right]} \alpha_{j}^{i}\left[2k-1\right] + \alpha_{j}^{i}\left[2k\right].$$
(2)

When the LLR value of the k-th bit at level zero α_0^k , $1 \le k \le N$, is calculated, the estimation of u[k], denoted as $\hat{u}[k]$, is

$$\hat{u}[k] = \hat{\beta}_0^k = \begin{cases} 0, & \text{if } k \in \mathbb{A}^c, \\ \frac{1 - \operatorname{sign}(\alpha_0^k)}{2}, & \text{otherwise.} \end{cases}$$
(3)

The hard messages are propagated back to the parent node as

$$\hat{\beta}_{j}^{i}[k] = \begin{cases} \hat{\beta}_{j-1}^{2i-1}\left[\frac{k+1}{2}\right] \oplus \hat{\beta}_{j-1}^{2i}\left[\frac{k+1}{2}\right], & \text{if } \mod(k,2) = 1, \\ \hat{\beta}_{j-1}^{2i}\left[\frac{k}{2}\right], & \text{if } \mod(k,2) = 0. \end{cases}$$
(4)

The estimation of each bit depends on the estimation of all previous bits in the SC decoding algorithm, which leads to a large latency. It was pointed out in [12] that for a node \mathcal{N}_j^i , the maximum-likelihood (ML) estimate of the vector $\beta_j^i [1:2^j]$ can be calculated in parallel by evaluating

$$\hat{\beta}_{j}^{i}\left[1:2^{j}\right] = \arg\max_{\beta_{j}^{i}\left[1:2^{j}\right] \in \mathbb{C}_{j}^{i}} \sum_{k=1}^{2^{j}} \left(-1\right)^{\beta_{j}^{i}\left[k\right]} \alpha_{j}^{i}\left[k\right], \qquad (5)$$

where \mathbb{C}_{j}^{i} is the set of all the codewords associated with node \mathcal{N}_{j}^{i} . The complexity of evaluating (5) is generally very high. However, the main idea behind fast-SSC decoding is that for some nodes with special frozen and non-frozen bit patterns the evaluation of (5) can be simplified significantly. Some prominent examples of such nodes include the Rate-0 node (2^{j} frozen bits), the Rate-1 node (2^{j} information bits), the repetition (REP) node (one information bit and $2^{j} - 1$ frozen bits), and the single parity-check (SPC) node ($2^{j} - 1$ information bits and one frozen bit).

The key advantage of using specific parallel decoders for the aforementioned special nodes is that, since the SC decoding tree is not traversed when one of these nodes is encountered, a significant latency reduction can be achieved. For example, if \mathcal{N}_j^i is a Rate-1 node, hard decision decoding can be used to immediately obtain the decoding result as

$$\hat{\beta}_{j}^{i}[k] = h\left(\alpha_{j}^{i}[k]\right) = \begin{cases} 0, & \text{if } \alpha_{j}^{i}[k] \ge 0, \\ 1, & \text{otherwise.} \end{cases}$$
(6)

If \mathcal{N}_j^i is a REP node, all its bits are either equal to one or equal to zero. According to (5), estimation can be obtained by extracting the sign bit of the sum of its LLR values. If \mathcal{N}_j^i is an SPC node, a hard decision based on (6) is first performed, which is followed by the calculation of the parity of the output using modulo-2 addition. The hard decision value with the index of the least reliable bit will be flipped if the parity check constraint is not met.

2) Fast-SSC Decoder Hardware Architectures: A typical fast-SSC decoder contains three main modules [5]: a memory, an arithmetic logical unit (ALU), and a controller. The memory consists of five separate sub-modules. The channel LLR, internal LLR α , and estimation β sub-modules feed the ALU. The instruction sub-module stores the operations to be executed and is routed into the controller. Finally, the codeword sub-module stores and outputs the final codeword.



Fig. 2. General structure of a sequence repetition node.

The ALU implements the f function given in (1), the g function given in (2), the combining operation given in (4), as well as the update rules for various special nodes like the rate-1 node given in (6). Finally, the controller tracks which node in the decoding tree is currently being decoded by using a list of instructions that is pre-compiled based on \mathbb{A} and \mathbb{A}^c .

III. FAST-SSC DECODING WITH SEQUENCE REPETITION NODES

A. Sequence Repetition Node

Let \mathcal{N}_j^i be a node at level j of the binary tree representation of SC decoding as shown in Fig. 1. An SR node is any node at stage j for which all its descendants are either Rate-0 or REP nodes, except the rightmost one at a certain stage $r, 0 \le r \le j$, that is a generic node of rate C. The general structure of an SR node is depicted in Fig. 2. The rightmost node $\mathcal{N}_r^{i \times 2^{j-r}}$ at stage r is denoted as the source node of the SR node \mathcal{N}_i^i . Let $E = i \times 2^{j-r}$ so the source node can be denoted as \mathcal{N}_r^E .

An SR node can be represented by three parameters as SR(v, SNT, r), where r is the level of the SC decoding tree in which \mathcal{N}_r^E is located. SNT is the source node type, and as shown in [1], SNT \in {Rate-0, Rate-1, EG-PC, Rate-C}. The EG-PC node is a node at level j having all its descendants as Rate-1 nodes except the leftmost one at a certain level r < j, that is a Rate-0 or REP node. Rate-C is a generic node of rate C. When SNT \in {Rate-0, Rate-1, EG-PC}, the source node is a special node whose bits are all non-frozen except the leftmost b bits, where

$$b = \begin{cases} 0, & \text{if SNT=Rate-1,} \\ 1, & \text{if SNT=Rate-0,} \\ 2^{h} \text{ or } 2^{h} - 1, & \text{if SNT=EG-PC,} \end{cases}$$
(7)

and where h < r - 1 is the level of the leftmost Rate-0/REP node of the EG-PC node. Note that the source node has a minimum length of 2 as all the possible frozen bit patterns with length 2 fall into the above category. The vector $\boldsymbol{v} =$ $(v[j], v[j-1], \ldots, v[r+1])$ has length (j-r) such that for the left child node of the parent node of \mathcal{N}_r^E at level k, r < $k \leq j, v[k]$ is calculated as

$$v[k] = \begin{cases} 0, & \text{if the left child node is a Rate-0 node,} \\ 1, & \text{if the left child node is a REP node.} \end{cases}$$
(8)

Note that when r = j, \mathcal{N}_j^i is a source node and thus v is an empty vector denoted as $v = \emptyset$.

B. Repetition Sequence

In this subsection, we define repetition sequences, which can be used to calculate the output bit estimates of an SR node based on the estimates of its source node. To derive the repetition sequences, v is used to generate all the possible sequences that have to be XORed with the output of the source node to generate the output bit estimates of the SR node. Let η_k denote the rightmost bit value of the left child node of the parent node of \mathcal{N}_r^E at level k+1. When v[k+1] = 0, the left child node is a Rate-0 node so $\eta_k = 0$. When v[k+1] = 1, the left child node is a REP node, thus η_k can take the value of either 0 or 1. The number of repetition sequences is dependent on the number of different values that η_k can take. Let W_v denote the number of 1's in v. The number of all possible repetition sequences is thus 2^{W_v} . Let $\mathbb{S} = \{s_1, \ldots, s_{2^{W_v}}\}$ denote the set of all possible repetition sequences.

The output bits of SR node $\beta_i^j [1:2^j]$ have the property that their repetition sequence is repeated in blocks of length 2^{j-r} . Let $\beta_r^E [1:2^r]$ denote the output bits of the source node of an SR node \mathcal{N}_j^i . The output bits for each block of length 2^{j-r} in \mathcal{N}_j^i with respect to $\beta_r^E [1:2^r]$ can be written as

$$\beta_{j}^{i}\left[\left(k-1\right)2^{j-r}+1:k2^{j-r}\right] = \beta_{r}^{E}\left[k\right] \oplus \boldsymbol{s}_{l}, \qquad (9)$$

where $k \in \{1, ..., 2^r\}$ and $s_l = \{s_l[1], ..., s_l[2^{j-r}]\}$ is the *l*-th repetition sequence in S. To obtain the repetition sequence s_l and with a slight abuse of terminology and notation for convenience, the Kronecker sum operator \boxplus is used, which is equivalent to the Kronecker product operator, except that addition in GF(2) is used instead of multiplication. For each set of values that η_k 's can take, s_l can be calculated as

$$s_l = (\eta_r, 0) \boxplus (\eta_{r+1}, 0) \boxplus \cdots \boxplus (\eta_{j-1}, 0).$$
 (10)

For a given code, the locations of SR nodes in the decoding tree are fixed and can be determined offline. Therefore, the repetition sequences in S of all of the SR nodes can be precomputed and used in the course of decoding.

C. Decoding of SR Nodes

To decode SR nodes, the LLR values $\alpha_{r_l}^E[1:2^r]$ of the source node \mathcal{N}_r^E associated with the *l*-th repetition sequence s_l are calculated based on the LLR values $\alpha_j^i[1:2^j]$ of the SR node \mathcal{N}_j^i and repetition sequence s_l by the following equation which is proved in [1, Proposition 1]

$$\alpha_{r_{l}}^{E}[k] = \sum_{m=1}^{2^{j-r}} \alpha_{j}^{i} \left[(k-1) \, 2^{j-r} + m \right] (-1)^{s_{l}[m]} \,. \tag{11}$$

Using (9) and (11), (5) can be written as [1, (20)]

$$\hat{\beta}_{j}^{i} = \underset{\substack{\beta_{r}^{E}[1:2^{r}] \in \mathbb{C}_{r}^{E} \\ l \in \{1,...,|S|\}}}{\arg \max} \sum_{k=1}^{2^{r}} (-1)^{\beta_{r}^{E}[k]} \alpha_{r_{l}}^{E}[k].$$
(12)

Thus, the bit estimates of an SR node $\hat{\beta}_j^i [1:2^j]$ can be calculated by finding the bit estimates of its source node $\beta_r^E [1:2^r]$ and the repetition sequence using (12), and then combine them as shown in (9).

Algorithm 1: Decoding algorithm of SR node \mathcal{N}_{i}^{i}

Input: $\alpha_{i}^{i} [1:2^{j}], S;$ **Output:** $\hat{\beta}_{j}^{i} [1:2^{j}];$ // Step 1: Soft message computation $\begin{array}{l} \text{for } l \in \{1, \ldots, |\mathbb{S}|\} \text{ do} \\ \text{Calculate } \alpha^E_{r_l} \text{ according to (11).} \end{array}$ end // Step 2: Decoding of source node \mathcal{N}^E_r for $l \in \{1, ..., |S|\}$ do if SNT=Rate-C then Decode source node \mathcal{N}_r^E using $\alpha_{r_l}^E$ to get $\hat{\beta}_{r_l}^E$. else $\hat{\beta}_{r_l}^E[k] = h\left(\alpha_{r_l}^E[k]\right), \ k \in \{1, \dots, 2^r\}.$ if SNT $\neq Rate-1$ then Perform parity check and bit flipping on $\hat{\beta}_{r_l}^E$ using $\alpha_{r_l}^E$. end end end // Step 3: Comparison and selection $\hat{l} = \underset{l \in \{1, \dots, |S|\}}{\operatorname{arg\,max}} \sum_{k=1}^{2^{r}} \left| \alpha_{r_{l}}^{E} \left[k \right] \right|.$ (13)

Return $\hat{\beta}_{j_i}^i$ to parent node according to (9).

The decoding algorithm of an SR node \mathcal{N}_j^i is described in Algorithm 1. The algorithm first calculates $\alpha_{r_l}^E$ to obtain the soft messages that go into the source node for the l^{th} repetition sequence s_l , $l \in \{1, \ldots, |\mathbb{S}|\}$. $\alpha_{r_l}^E$, $\hat{\beta}_{r_l}^E$, and $\hat{\beta}_{j_l}^i$ are the soft and hard messages associated with s_l . Then, the source node is decoded under the rule of the SC decoding. If the source node is a special node, a hard decision is made directly. Parity check and bit flipping will be performed further using Wagner decoding if SNT \neq Rate-1. Finally, the index of the optimal repetition sequence can be selected according to the comparison in (13) and the decoding result is obtained according to (9). Based on Algorithm 1, the SR node-based fast-SSC (SRFSC) decoding algorithm is proposed. It follows the SC decoding algorithm schedule until an SR node is encountered where Algorithm 1 is executed.

IV. ARCHITECTURE OF SRFSC DECODER

The top-level architecture of the proposed SRFSC decoder is shown in Fig. 3. When decoding starts, the instructions for the polar code that is being decoded are fetched by the controller and the channel LLRs are loaded into memory. The controller decodes the instructions to get the node schedule and updates the decoding stage parameters accordingly. The updates in the controller follow the principle of SC decoding until an instruction corresponding to an SR node is reached, where the SR module is activated to process the LLRs. The estimation results from both the SR module and processing module are routed into the partial sum network (PSN) module, from where the estimated codeword is also output when decoding



Fig. 3. Top-level architecture of the proposed SRFSC decoder.

SRstage	SourceStage	FroNum	SeqNum	NodeType
$\log_2\left(1+\log_2 P\right)$	$\log_2(1 + \log_2 P)$	$\log_2\left(\frac{P}{2}\right)$	$\log_2\left(\frac{\log_2 P}{2}\right)$	$\log_2\left(NT(N,\mathbb{A},P)\right)$

Fig. 4. Instruction structure of the proposed SRFSC decoder.

terminates. In the following, the architecture of the various individual modules is discussed in detail.

A. Memory, Processing, and PSN Modules

The architectures of these three modules are identical with those presented in [13] and we thus only describe them on a high level. The memory module stores all soft messages α . The update of hard estimates β is in the partial sum network (PSN) module. A set of P processing elements (PEs) is instantiated in the processing module to process up to 2P LLRs in parallel. A PE implements both the f and the g function using sign-andmagnitude representation and the appropriate output is selected according to the current decoding stage.

B. Controller Module

The operation in the controller module follows the standard SC decoding schedule until an instruction that indicates an SR node is found. When this occurs, the 2*P* LLRs will be routed to the SR module instead of the processing module to perform the decoding of SR node in Algorithm 1. The required number of clock cycles to decode the SR node by the SR module is precalculated and a counter is initialized to this value. All updates in the controller are suspended until the counter reaches zero. Then, the decoding bit index is added the length of the SR node and the updates resume. Although the Rate-0 and Rate-1 nodes can also be represented as special cases of SR nodes, the controller will bypass the SR module and signal the processing module to execute immediate decoding for these two nodes so that there is no additional latency.

The structure of the instructions used in the controller is shown in Fig. 4. The instructions contain all the required information to decode an SR node and they are stored in memory according to the visiting order in the decoding tree. The elements SRstage, SourceStage, FroNum, SeqNum and NodeType in the instruction represent the stage of SR node, the stage of source node, the number of frozen bits in source node, the base 2 logarithm of the number of repetition sequences and the node type, respectively.¹ Moreover, the vector v is replaced with SRstage, SeqNum and NodeType since these three elements can be used directly in the decoder, so that additional calculations (e.g., (10)) can be avoided. NodeType is in fact a pointer to the memory of repetition sequences. As only nodes with SeqNum > 0 have non-zero repetition sequences that need to be stored, NodeType refers to these node types and is used as pointer to find their corresponding repetition sequences in the memory.

The different repetition sequences in the SR node are processed in parallel. Since a maximum of 2P LLRs are input to the SR module each time, we have the constraint

$$2^{\text{SRstage}+\text{SeqNum}} < 2P. \tag{14}$$

All SR nodes that meet this constraint can be handled, while others are divided into smaller nodes. Therefore, SRstage and SourceStage always have values between 0 and $1 + \log_2 P$. FroNum can be calculated according to (7) and thus have values between 0 and P/2. Consider source node with a minimum length of 2. Then, the maximum value of SeqNum is constrained by $2^{1+2\text{SeqNum}} \leq 2P$. Thus, SeqNum has values between 0 and $\frac{1}{2}\log_2 P$. As for NodeType, it has values between 0 and $NT(N, \mathbb{A}, P)$, where NT is a function of N, \mathbb{A} and P, which depends on the polar code being decoded.

As an example, we consider a set of 5G polar codes [14] of length N = 1024 and rates R = 1/2, R = 1/4, and R = 3/4. For a code length of N = 1024, P = 64 is shown to be a reasonable choice [11]. With these parameters, in Fig. 4, SRstage and SourceStage take values in $\{0, 1, \ldots, 7\}$, FroNum takes values in $\{0, 1, \ldots, 3\}$, and SeqNum takes values in $\{0, 1, 2\}$. The three considered codes contain a total of six SR nodes with SeqNum > 0. As such, NodeType takes values in $\{0, 1, \ldots, 6\}$. Specifically, when NodeType = 0, the node only has an all-zero repetition sequence and the remaining values represent the six SR nodes with SeqNum > 0. From the above analysis, the size of each instruction for the considered example is 13 bits.

C. SR Module

The ranges of some elements in the instructions are variable and depend on the set of supported polar codes. Thus, some of the data widths in the SR module are also variable and it is difficult to give a fully generic explanation of our proposed architecture. For this reason, we consider the previous example of N = 1024, $R \in \{1/2, 1/4, 3/4\}$, and P = 64. The architecture of the SR module for this example is shown in Fig. 5. The submodules with red, blue, and green color correspond to the operations in Step 1, Step 2, and Step 3 in

¹Note that we use FroNum instead of SNT because no SR node with a Rate-C node as its source node is found for the code length (N = 1024) and rates (R = 1/2, 1/4, 3/4) that we consider in Section V

Cmd Cmd layer Cmd Repetition 160Sequence Cmd_4 4×7bits $2P \times Q$ 2P7-layer CS SR bits Parity Check $2P \times O$ $Cmd_1 \rightarrow$ generation tree 40 $Cmd_2 \rightarrow$ $Cmd_3 \rightarrow$ 2-layer Repetition $Cmd_4 \rightarrow$ adder tree Sequence Step 1 Step 2 Step 3

Fig. 5. Example of the SR module architecture for N = 1024, $R \in \{1/2, 1/4, 3/4\}$, and P = 64.

Algorithm 1, respectively, and are explained in more detail in the sequel.

Step 1: This part of the SR decoder is used to calculate the input LLRs into the source node if SRstage \neq SourceStage. In the XOR submodule, the first 2^{SRstage} LLRs in the 2P inputs are repeated 2^{SeqNum} times so that the decoding for different repetition sequences can be handled in parallel. The repetition sequences are obtained using NodeType. They will be XORed with the sign bit of the 2^{SeqNum} input repetitions according to (11). The XOR result of different repetition sequences are concatenated and expanded into a vector of length 2P by appending zeros if $2^{\text{SRstage+SeqNum}} < 2P$. Then, the LLR vector enters a $(1 + \log_2 P)$ -layer adder tree that performs the addition of LLRs in (11). The command signal $Cmd_1 = 7$ – (SRstage - SourceStage) is pre-calculated in the control module and it is used in the adder tree to decide the addition result of which layer will be output by a multiplexer. Those outputs from the adder tree are the input LLRs of the source node for different repetition sequences. In the considered example, there exist $2^{\text{SourceStage+SeqNum}} < 16$ for SR node whose SRstage \neq SourceStage. Moreover, all LLRs are quantized using Q bits. Thus, the data width of the adder tree output is 16Q bits.

Step 2: This part of the SR node is used to perform the parity-check and bit-flipping steps for the source node. The LLRs of the source node first enter a $(1 + \log_2 P)$ -layer compare-select (CS) tree. Processing units in the CS tree execute the f function to decode SPC node. There are two cases where more than one SPC nodes will be decoded in parallel in our design: 1) when FroNum = 1 and SeqNum > 0, there are 2^{SeqNum} SPC nodes which correspond to different repetition sequences and are decoded simultaneously, and 2) when FroNum = 2 and FroNum = 3, the decoding of source node can be viewed as a parallel decoding of 2 and 4 SPC nodes, respectively [1]. The length of the SPC node decides the layer from which the index of the least reliable input and the f function result are selected. As the length of the SPC node can be calculated as $2^{\text{SourceStage}+1-\text{FroNum}}$, the output layer selection signal Cmd₄ has the following representation

$$\mathbf{Cmd}_4 = \begin{cases} 7, & \text{FroNum} = 0, \\ 6 - \text{SourceStage} + \text{FroNum}, & \text{otherwise.} \end{cases}$$
(15)

Since the maximum number of parallel SPC nodes in our example is 4, the output indices and LLRs have a data width of 4×7 and 4Q bits, respectively. Note that the output LLRs goes both to the parity check module and a 2-layer adder tree. This is because all SPC nodes have an even parity constraint except when FroNum = 3, where SPC nodes can have an even or odd parity constraint which is calculated according to [1, (16)] and implemented by a 2-layer adder tree.

The parity constraint type, the output indices, and LLRs are then input into the parity-check submodule to do the parity check and bit flipping on these SPC nodes using [1, (13)]. Then, the estimated bits of these SPC nodes are concatenated to form the estimated bits of source node and they are XORed with the repetition sequence to generate the estimated bits of SR node in the SR bits generation submodule according to (9). Finally, the SR bits corresponding to the repetition sequence with the index value from Step 3 are selected as the output.

part This SR decoder Step 3: of the is executed in parallel with Step 2 to evaluate (13) using a $(\texttt{SourceStage} + \texttt{SeqNum})_{\max}$ -layer $SeqNum_{max}$ -layer CS tree, where adder tree and $(\texttt{SourceStage} + \texttt{SeqNum})_{\max}$ is the maximum value of (SourceStage + SeqNum) for all SR nodes with ${
m SeqNum}$ > 0 and ${
m SeqNum}_{
m max}$ denotes the maximum value of SeqNum. As only magnitudes of LLRs are used for addition in (13), all inputs are positive. As a result, the processing unit in the 4-layer adder tree is simpler than that in the 7-layer adder tree in Step 1 because it does not need to compare magnitudes. The output of the adder tree is selected by the output layer selection signal $Cmd_2 = 4 - SourceStage$ and has a bit-width of 4Q as there are at most 4 repetition sequences in the considered example. The four sums are then input into the 2-layer CS tree to find the index of the maximum using selection signal $Cmd_3 = 2 - SeqNum$. Finally, the index is obtained from a





Fig. 6. Floating-point and fixed-point FER and BER performance for SRFSC decoding of 5G polar codes \mathcal{P} (1024, 512) [14].

multiplexer and the value is 0 if SeqNum = 0 and the output from the CS tree otherwise.

V. IMPLEMENTATION RESULTS

The proposed decoder has been implemented using VHDL and targeting an Altera Stratix IV EP4SGX530KH40C2 FPGA device. Channel LLRs are generated by transmitting random codewords through an additive white Gaussian noise (AWGN) channel after binary phase-shift keying (BPSK) modulation. A quantization scheme Q(6, 4, 0) has been used, where $Q(Q_i, Q_c, Q_f)$ are the quantization bit size for internal LLRs, channel LLRs, and fraction bit size for both internal and channel LLRs, respectively. This scheme leads to an errorcorrecting performance that is very close to that of the floatingpoint implementation, as shown in Fig. 6.

Table I compares the proposed decoder with other stateof-the-art works. As can be seen, the proposed SRFSC decoder provides a 17.9% and 31.7% throughput improvement compared to the architectures presented in [11] and [5], respectively. This is mainly due to a 9.9% and 22.4% higher f_{max} with respect to [5] and [11]. The number of CLKs in our work is slightly higher than that in [11]. This is because of the insertion of some registers to decrease certain critical paths and because we have not merged f ad g operations as was done in [11]. In addition, a total of 186, 200 CLKs are required at rates 1/4, 3/4, respectively. In terms of the used LUTs, this work requires an increase of 23.2% and 187.5% compared to [11] and [5], respectively. As far as the memory size is concerned, although our decoder uses fewer RAM bits, the required number of registers is about 8 times higher compared to [5], [11]. The big difference in registers can be mostly attributed to the separate storage of channel and internal LLRs in synthesis. Internal LLRs are stored in RAM and channel LLRs are arranged in registers, while in other works both are stored in RAM.

TABLE I FPGA IMPLEMENTATION RESULTS FOR \mathcal{P} (1024, 512).

	[5]	[11]	This Work
Quantization	Q(6, 4, 1)	Q(6, 4, 1)	Q(6, 4, 0)
P	64	64	64
LUTs	6126	14300	17615
Registers	1223	1216	10505
RAM (bits)	23592	18350	16128
Instruction size	5 bits	6 bits	13 bits
# of Instruction	209	157	41
# of CLKs	266	214	222
$f_{\rm max}$ (MHz)	99.8	89.6	109.6
T/P (Mps)	384	428.6	505.6

VI. CONCLUSION

In this paper, we presented the first FPGA implementation of the SRFSC decoder for polar codes. To this end, we designed a dedicated architecture for the SR node processor. For a 5G polar code with length 1024, code rate 1/2 and P = 64 processing units, we obtained a 17.9% improvement in throughput over the previous work.

REFERENCES

- H. Zheng, S. A. Hashemi, A. Balatsoukas-Stimming, Z. Cao, A. M. J. Koonen, J. Cioffi, and A. Goldsmith, "Threshold-based fast successivecancellation decoding of polar codes," *arXiv*:2005.04394, 2020.
- [2] E. Arıkan, "Channel polarization: A method for constructing capacityachieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.
- [3] T. Koonen, K. A. Mekonnen, F. Huijskens, Q. Pham, Z. Cao, and E. Tangdiongga, "Fully passive user localization for beam-steered highcapacity optical wireless communication system," *J. Lightw. Technol.*, pp. 1–1, Mar. 2020.
- [4] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successivecancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] H. Gamage, V. Ranasinghe, N. Rajatheva, and M. Latva-aho, "Low latency decoder for short blocklength polar codes," *arXiv*:1911.03201, 2019.
- [9] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "A 638 Mbps lowcomplexity rate 1/2 polar decoder on FPGAs," in *IEEE Int. Workshop* on Sig. Proc. Systems (SiPS), Oct. 2015, pp. 1–6.
- [10] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast low-complexity decoders for low-rate polar codes," J. Sign. Process. Syst., vol. 90, no. 5, pp. 675–685, May. 2018.
- [11] F. Ercan, C. Condo, and W. J. Gross, "Reduced-memory high-throughput fast-SSC polar code decoder architecture," in *IEEE Int. Workshop on Sig. Proc. Systems (SiPS)*, Oct. 2017, pp. 1–6.
- [12] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725–728, Apr. 2013.
- [13] 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.
 [14] 3GPP, "3GPP TS RAN 38.212 v1.2.1," Dec. 2017. [Online]. Available:
- [14] 3GPP, "3GPP TS RAN 38.212 v1.2.1," Dec. 2017. [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/38_series/38.212/38212-f30.zip.