A 506 Gbit/s Polar Successive Cancellation List Decoder with CRC

Claus Kestel*, Lucas Johannsen†, Oliver Griebel*, Jhon Jimenez‡,
Timo Vogt†, Timo Lehnigk-Emden‡ and Norbert Wehn*
*University of Kaiserslautern, Kaiserslautern, Germany 67663
Email: {kestel, griebel, wehn}@eit.uni-kl.de
†Koblenz University of Applied Sciences, Koblenz, Germany 56075
Email: {johannsen, vogt}@hs-koblenz.de
‡Creonic GmbH, Kaiserslautern, Germany 67663
Email: {jhon.jimenez, timo.lehnigk-emden}@creonic.com

***Abstract*—Polar codes have recently attracted significant attention due to their excellent error-correction capabilities. However, efficient decoding of Polar codes for high throughput is very challenging. Beyond 5G, data rates towards 1 Tbit/s are expected. Low complexity decoding algorithms like Successive Cancellation (SC) decoding enable such high throughput but suffer on error-correction performance. Polar Successive Cancellation List (SCL) decoders, with and without Cyclic Redundancy Check (CRC), exhibit a much better error-correction but imply higher implementation cost. In this paper we in-depth investigate and quantify various trade-offs of these decoding algorithms with respect to error-correction capability and implementation costs in terms of area, throughput and energy efficiency in a 28 nm CMOS FD-SOI technology. We present a framework that automatically generates decoder architectures for throughputs beyond 100 Gbit/s. This framework includes various architectural optimizations for SCL decoders that go beyond State-of-the-Art. We demonstrate a 506 Gbit/s SCL decoder with CRC that was generated by this framework.**

***Index Terms*—Polar Code, Polar Decoder, High Throughput, 28nm, List Decoding, CRC, Advanced Sorting, Beyond 5G**

I. Introduction

Channel coding is an essential part in baseband processing and enables reliable transmission. It has a long history going back to Shannon's noisy channel coding theorem in 1948 [1]. Advanced channel decoding schemes exploit soft information to improve the error-correction capabilities. The State-of-the-Art (SOA) coding schemes for soft decoding known today are Turbo codes, Low Density Parity Check (LDPC) codes and Polar codes. These codes are adopted in many communications standards like LTE, WiMAX, Wi-Fi, DVB-S2 and Ethernet, to name but a few. Polar codes are relatively new [2]. In 2009, Erdal Arıkan proved that these codes achieve channel capacity for the Binary Symmetric Memoryless Channel (BSMC). Due to their excellent error-correction performance, Polar codes have attracted significant attention and became part of the new 5G standard [3]. Future beyond 5G use cases are expected to require data rates in the Tbit/s range, which is one to two orders of magnitude higher than the throughput of 5G.

We gratefully acknowledge financial support by the EU (project-ID: 760150-EPIC) and the Federal Ministry for Economic Affairs and Energy on the basis of a resolution of the German Bundestag (project-ID: F4013407PR7).

Successive Cancellation (SC) and Successive Cancellation List (SCL) [4] are the most prominent decoding algorithms for Polar codes. SC comes with a low algorithmic complexity but a limited error-correction performance for finite code lengths. SCL applies list decoding on the SC algorithm which significantly improves the error correction at the cost of higher algorithmic complexity. SC and SCL decoding algorithms traverse the Polar Factor Tree (PFT) in a depth-first manner [5] resulting in a sequential decoding procedure. Hence, it is common belief that Polar code decoding with SC and SCL cannot compete with LDPC codes in terms of very high throughput since LDPC codes are decoded with the Belief Propagation (BP) algorithm. The BP has an inherent parallelism, which allows a very high throughput in a natural way. Due to this behavior, BP was also adopted for Polar code decoding [2] to mitigate the problem of the sequential behavior of SC and SCL. However, BP applied to Polar code decoding needs a large number of decoding iterations to approach the error-correction performance of SC [6]. This large number of iterations decreases the throughput and increases the latency. Moreover, even for a very high number of iterations, BP cannot compete with the error-correction performance of SCL decoding [7]. Hence, in this paper, we focus on advanced SCL decoder architectures for throughput towards Tbit/s due to their good error-correction capabilities.

To reach very high throughput without relinquishing error-correction performance, improvements on algorithmic and architectural level were investigated in the past. On algorithmic level, pruning the PFT for SC [5], [8] and SCL [9] can largely reduce the implementation complexity. At the architectural level, an unrolling of the PFT traversal and corresponding pipelining mitigates the sequential data dependencies and, thus, enables very high throughput. The principle of unrolling, applied to LDPC in decoders [10], was later adopted for Polar code decoding with SC [11] and SCL [12]. 644 Gbit/s coded throughput is reported for SC based decoder in a 28 nm technology in [13]. In [14], even 1274 Gbit/s coded throughput is presented for the same technology, but these results are based on synthesis only. After placement and routing, a much lower throughput is expected. As of today, only one publication of an unrolled SCL decoder exists which achieves 12 Gbit/s coded throughput in a 28 nm technology [12].

Concatenating Polar codes with a Cyclic Redundancy Check (CRC) notably improves the error-correction performance [4]. To the best of our knowledge, none of the previously published high-throughput architectures provide CRC support.

This paper makes the following new contributions:

- We in-depth investigate and quantify various trade-offs of the SC and SCL decoding algorithms with respect to error-correction performance and implementation costs in terms of area, throughput and energy efficiency in a 28 nm CMOS FD-SOI technology.
- We present a framework that automatically generates SCL decoder architectures. The framework includes various architectural optimizations for SCL decoders that go beyond State-of-the-Art.
- Finally we present a 506 Gbit/s SCL decoder with CRC that was automatically generated.

The remainder of this paper is structured as follows. We give a brief review of Polar codes and their decoding algorithms, targeting optimized SCL, in Section II. In Section III, we present the architectural concept of our SCL decoders, the framework and the optimized nodes. Section IV gives detailed post-place-and-route results for area, timing and power in 28 nm technology. We discuss the trade-offs between error-correction performance and implementation cost and compare our designs against a SOA SCL decoder implementation. Section V concludes the paper.

II. Background

Polar codes are linear block codes of length $N = 2^n$, K information bits and rate $R = K/N$, denoted by $\mathcal{P}(N, K)$. They belong to the class of multilevel concatenated codes and are related to Reed-Muller-Codes [15]. In contrast to Reed-Muller-Codes, Polar codes use the phenomenon of channel polarization [2] to split bit-channels into reliable and noisy channels. Information bits are sent over reliable channels, specified in the information set $\mathcal{I}$. The unreliable channels, specified in the frozen set $\mathcal{F} = \mathcal{I}^C$ of length $N - K$, are used to include redundancy. The unreliable positions in an input vector $\boldsymbol{u}$ of length N, called frozen bits, are set to zero while the information bits are set in the positions of $\mathcal{I}$. Polar code construction is equivalent to finding the most reliable channels for a given code length and rate, i. e., the set $\mathcal{I}$, and depends on the underlying channel assumptions.

A. SC Decoding

The process of SC decoding [2] for Polar codes of length N can be represented as message passing between nodes of a balanced binary tree [5] with $2N - 1$ nodes, named PFT. The leaf nodes at layer $s = 0$ are the N bits to be estimated by the decoder. The input to the root node with $s = n$ is a vector $\boldsymbol{\alpha}$ of N received channel Log-Likelihood Ratios (LLRs). In layer s of the PFT, a node of size $N_v = 2^s$ receives a vector $\boldsymbol{\alpha}^v$ of N_v LLRs from the parent node and returns a hard decision vector $\boldsymbol{\beta^v}$ of same length. The messages to the left and the right child nodes, $\boldsymbol{\alpha}^l$ and $\boldsymbol{\alpha}^r$, are computed element-wise by

$$\alpha_i^l = f\left(\alpha_i^v, \alpha_{i+N_v/2}^v\right) \tag{1}$$

$$\alpha_i^r = g\left(\alpha_i^v, \alpha_{i+N_v/2}^v, \beta_i^l\right) \tag{2}$$

with the hardware-efficient min-sum formulation of f- and g-functions [16] being

$$f(a, b) = \operatorname{sign}(a) \operatorname{sign}(b) \min(|a|, |b|), \tag{3}$$

$$g(a, b, c) = (1 - 2c) \cdot a + b. \tag{4}$$

Finally, the message to the parent node, the partial sum vector $\boldsymbol{\beta}^v$, is calculated by

$$\beta_i^v = \begin{cases} \beta_i^l \oplus \beta_i^r, & \text{if } i < N_v/2 \\ \beta_{i-N_v/2}^r, & \text{otherwise,} \end{cases} \tag{5}$$

where $\oplus$ denotes the XOR-operation. We call (5) h-function. In this way, the PFT is traversed depth first with priority to the left child, since its result is used in (2) to determine the message to the right child. In the leaf nodes, $\boldsymbol{\beta^v}$ only has one element, the estimated bit $\hat{u}_i$ determined by

$$\hat{u}_i = \begin{cases} 0, & \text{if } i \in \mathcal{F} \text{ or } \alpha_i^v \geq 0 \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

The PFT has N leaf nodes and $N - 1$ non leaf nodes. For the former, in total N bit decisions have to be made while for the latter, the three vector functions (1), (2) and (5) have to be calculated. Nodes whose leafs are all frozen (called Rate-0) or nodes whose leafs are all information bits (called Rate-1 nodes) can be simplified [5]. Alike, PFT tree pruning can be carried out for nodes that represent Repetition code (REP) nodes and Single Parity-Check code (SPC) nodes [8]. This PFT pruning largely reduces the number of nodes of the PFT. In unrolled decoder architectures, each vector function executed by a node, called a computational kernel, corresponds to a pipeline stage. Hence, the PFT pruning largely reduces the number of the pipeline stages and therefore the complexity of such high-throughput decoders.

B. SCL Decoding

SCL decoding has been introduced in [4] to improve the error-correction performance of Polar codes. In the PFT, instead of vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, lists of vectors are passed among the nodes. At the leafs of the PFT both possible values, 0 and 1, for an information bit estimation are considered. The number of codeword candidates (paths) therefore doubles for every bit estimation. The number of paths is limited to the list size L in order to keep decoding complexity practical. This implies that after a bit estimation paths must be discarded if their number exceeds L. The reliability of each path is rated by a Path Metric (PM) which is updated at every bit estimation. The lower the PM of a path, the more likely the associated codeword, thus paths with low PMs are allowed to survive. For an LLR-based SCL [17], the PMs are initialized with 0 and each PM of path l, when estimating bit i, is updated by

$$\mathrm{PM}_{l,i} = \begin{cases} \mathrm{PM}_{l,i-1} + \left|\alpha^v_{l,i}\right|, & \text{if } \beta^v_{l,i} \neq \mathrm{HDD}(\alpha^v_{l,i}) \\ \mathrm{PM}_{l,i-1}, & \text{otherwise,} \end{cases} \tag{7}$$

with $\mathrm{HDD}(\boldsymbol{\alpha}^v)$ being Hard Decision Decoding (HDD) on $\boldsymbol{\alpha}^v$.

The PM can be seen as a cost function, that is increased for the more unlikely bit decision with the absolute value of the dedicated LLR. After the last bit decision, the path with the lowest PM is chosen and output by the decoder. By including a CRC in the input vector $\boldsymbol{u}$ before the encoding, the selection of candidates in SCL decoding and therefore the error-correction performance can be further improved.

Alike to the PFT optimization for the SC algorithm, optimized Rate-0, Rate-1, REP and SPC nodes also exist for the SCL algorithm [9], [18]–[21].

On architectural level, unrolling and pipelining of the PFT traversal for the SCL algorithm can be performed in the same way as for the SC algorithm [12]. The high throughput gained by unrolling and pipelining comes at the cost of flexibility, since resulting decoders are specialized for a specific code and rate.

III. Polar Decoder Framework

In [13], a framework was presented to automatically generate unrolled pipelined high-throughput SC and Soft Cancellation (SCAN) Polar decoders. We extended this framework to also support SCL decoders. The framework written in C++ has the following inputs: the Polar code, selected decoding algorithm, target throughput, quantization and technology information. It outputs a fully synthesizable VHDL decoder model with corresponding test bench and test data. It is guaranteed by construction that the VHDL model and simulation model are bit-level equivalent. The central data structure of the framework is the PFT. The framework performs an automatic optimization and simplification of the PFT as described in the previous section. The framework contains a library of optimized computational kernels for the different PFT nodes. These kernels are implemented in C++ for error-correction performance simulation and as parameterizable VHDL hardware building blocks. In detail, the library contains the following kernels:

- *f-function*, see (1).
- *g-function*, see (2).
- *h-function*, see (5).
- *REP* constrained to $N_v = 4$.
- *SPC* constrained to $N_v = 4$.
- *Rate-0* without any constrains on N_v.
- *Rate-1* with details provided in Section III-A.
- *generic sorter* with details provided in Section III-B.

An SCL decoder architecture is generated by traversing the automatically optimized PFT. Whenever a node is visited a pipeline stage is created by using the building block associated with the corresponding computational kernel of the node. Some additional pointer management is necessary to keep track of the correct decoding paths. These pointers are determined in the leaf nodes, where the bit vectors are estimated from the LLRs, and represent the index of the input candidate each bit vector originates from. The leaf nodes also update the PMs, as described in Section II-B. In the sorter instances after each leaf node, the L best paths are selected and propagated. Parallel to the decoding process, the information bits determined in each leaf node are pushed to update the CRC register pipeline (see Section III-C). To optimize the decoder architecture, the framework inserts and balances registers between the different pipeline stages for a given throughput constraint according to derived timing models (see Section III-D).

A. Rate-1 Node

The Rate-1 node is an important node of SCL decoding which is described in more detail in the following. Depending on list size L and node size N_v, the node input consists of $L \times N_v$ LLRs with corresponding L PMs, while it outputs $L_{out} \times N_v$ bits with corresponding L_{out} PMs and path pointers. It is impractical to consider all $L \cdot 2^{N_v}$ possible codewords, thus different approaches were presented in literature. The architecture of [12] implements the algorithm of [9] where an approximation is introduced to reduce the number of candidate paths. They split the decoding path only for the two least reliable bits in the Rate-1 node. Thus, the error-correction performance sustains a loss. [21] proved that the needed number of paths splits without any error-correction performance degradation is

$$P = min(L-1, N_v). \tag{8}$$

However, the decoder architecture based on this algorithm presented in [20] needs $P+1$ time-steps to process a Rate-1 node. This approach is not suitable for our framework, since the limit for the processing of one stage is one clock-cycle. The same accounts for the procedure presented in [22], where also multiple clock-cycles are needed.

To generate candidates in the Rate-1 nodes, we apply the following approach: The input LLRs are decoded via HDD in a first step. Then, according to (8), the P lowest absolute values of the LLRs of each input path are identified. This is done according to the first two steps of the sorting procedure described in Section III-B. L bit vectors of length N_v save the estimated positions. Each of these vectors is then used to build 2^P bit vectors of length N_v representing split flags, which are XOR-ed with corresponding HDD-estimate of the input. For L inputs of N_v LLRs in this way

$$L_{out} = L \cdot 2^P \tag{9}$$

partial sums $\boldsymbol{\beta}^v$ are generated. The PMs are calculated according to (7) for each candidate. To keep the complexity manageable we introduced a restriction of $N_v = 4$ for $L > 4$.

Decoders without CRC do not need to further explore different candidates in the rightmost Rate-1 nodes of the PFT since there are no more frozen bits which can change the order in the candidate list. The most probable path at this stage will stay the most probable one until the end of the decoding process. Hence, the right-most Rate-1 nodes only have to process the most probable input by HDD. In this way, especially for high code rates, the area and power requirements of the whole decoder can be reduced significantly.

This is, to the best of our knowledge, the first implementation that decodes a complete Rate-1 node in one clock-cycle without any error-correction performance loss.

B. Sorter

The sorter unit is a further essential building block in the SCL decoder. The outputs of optimized leaf nodes with information bits, e. g., Rate-1, are connected to sorters. They reduce the number of L_{in} candidates to the list size L before the next node can start its calculations. Hence, as already mentioned by [17], sorting is a bottleneck for SCL decoders.

[23] proposes an iterative approach to reduce the sorting complexity by comparing the path metrics bit by bit. Depending on the numerical distribution of the PMs, a reduced number of iterations is required. However, a pipelined architecture requires building blocks with fixed latency. Hence, the advantage of such a dynamic iterative sorting approach cannot be exploited in our fully pipelined and unrolled decoder architecture. Other optimized sorting algorithms like pairwise sorting [24], simplified bubble sort or pruned bitonic sort [25] compare pairs of inputs in a list and switch positions, if required. The optimizations in these sorters focus on reducing the number of comparisons to minimizes the pairwise sorting cascades. E. g., in case of bitonic sort, the number of cascaded comparisons, corresponding to the latency, increases with $O(log^2(L_{in}))$. Since our architecture requires the sorting result in one clock cycle to achieve high throughput, such an approach is also unfeasible.

To mitigate these drawbacks, in this work, the sorting is performed in a 4-step approach:

1) *Comparisons*
   In the first stage, all PMs are compared to each other to form a binary matrix $\mathbf{M}$. For $i, j \in [0, \ldots, L_{in}-1]$, the elements $m_{i,j}$ of $\mathbf{M}$ are set according to

$$m_{i,j} = \begin{cases} 1, & \text{if } i < j \text{ and } \mathrm{PM}_i > \mathrm{PM}_j \\ 1, & \text{if } i > j \text{ and } \mathrm{PM}_i \geq \mathrm{PM}_j \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

   $\binom{L_{in}}{2}$ comparisons are needed to fill $\mathbf{M}$, since after comparing PM_i with PM_j, the values for $m_{i,j}$ and $m_{j,i}$ are known. The memory to store $\mathbf{M}$ can be reduced by omitting the entries for $i = j$. Thus, $\mathbf{M}$ is of size $L_{in} \times (L_{in} - 1)$, where the number of ones in row i represents how many PMs are smaller than PM_i.
2) *Rate paths*
   Each row of the matrix $\mathbf{M}$ represents the corresponding decoding path. To rate them, the row weight p_i is calculated by

$$p_i = \sum_{j=0}^{L_{in}-2} m_{i,j} \tag{11}$$

   for all rows $i \in [0, ..., L_{in} - 1]$ of $\mathbf{M}$.
3) *Candidate mapping*
   To realize input to output mapping of the candidates a multiplex-architecture is required. A vector $\boldsymbol{c}$ of length L_{in} holds the indices of each path $i \in [0, ..., L_{in} - 1]$ in sorted order and is assigned according to

$$c_{p_i} = i. \tag{12}$$

   It is guaranteed by the construction of p_i that the assignment from i to p_i is bijective.
4) *Connect sorted candidates*
   In the last step, the best input paths are assigned in sorted order to the output. This mapping has to be done for all PMs, bit vectors and path pointers

$$out(i) = in(c_i) \tag{13}$$

   for all $i \in [0, ..., L - 1]$.

This algorithm enables a single clock cycle implementation of the sorter unit with flexible L_{in} to L configurations instead of restricted $2L$ to L sorting as used in [17]. The flexibility is necessary to sort the different number of candidates generated by the kernels. Up to list sizes of 8 the sorter unit is not the dominant critical path.

C. CRC On-The-Fly

To improve the Frame Error Rate (FER) performance, Tal and Vardy [4] proposed the concatenation of Polar codes with a short CRC to assist the selection of the best candidate at the end of the decoding process. To avoid an increase in the latency due to the additional calculation of the L CRCs, the CRC update has to be carried out in parallel to the decoding. We transformed the CRC update process of [20], which was used in a sequential decoder architecture, to our pipelined decoders. The CRC registers are integrated in and synchronized to the overall memory structure of the decoder. For a given CRC polynomial, the framework automatically generates the corresponding VHDL building blocks. In this paper, we selected a 6-bit CRC defined by the CRC-polynomial $g(x) = x^6 + x^5 + 1$, due to its good performance for the chosen codes. For a fair comparison of FER performance with respect to the code rate, the CRC bits are placed in the positions of the best frozen bit channels. Thus, the code rate for the Polar code increases, but the overall code rate, including the CRC, does not change.

D. Optimized Register Balancing and Power Optimization

Deeply pipelined high-throughput architectures require a huge amount of memory that costs area and more particularly power. To face this issue, in the architecture of [12], registers of the delay line[1] are removed, while the output registers of all

[1]The delay line holds $\boldsymbol{\alpha}_v$ and $\boldsymbol{\beta}_l$ values until they are needed for the calculation of (2) and (5) respectively.

computational elements are retained. Thus, the architecture of [12] is partially-pipelined with an initiation interval of $I = 20$, i. e., their decoder inputs a new block respectively outputs a result every 20 clock cycles.

Our architectures are fully pipelined with an initiation interval $I = 1$. Registers are automatically inserted and balanced in the delay lines and computational units for a given frequency constraint f that is input to the framework. In this way the number of registers is minimized and, hence, the area and power of the generated decoder is optimized for given f. Note that f fixes the throughput since a code block of length N is processed every clock cycle ($I = 1$) yielding a coded throughput of $N \cdot f$. Key to this register optimization is a technology dependant timing characterization of the eight PFT kernels and a timing engine that automatically calculates the critical timing paths in the architecture for a given framework input, i. e., block size, decoding algorithm, list size. The delay of the kernels depends on list size L and/or node size N_v. Thus, we developed parametrizable VHDL building blocks for each kernel and characterized their timing behavior by performing in total 251 synthesis runs in 28 nm technology for various parameter settings. A parameter fitting with minimizing the least-squares was carried out to approximate the delay of the kernels by quadratic delay functions. For all kernels except the sorter, the corresponding delay function t is

$$\begin{aligned} t_{kernel}(L, N_v) = k_0 + k_L \cdot L + k_N \cdot N_v \\ + k_{LN} \cdot L \cdot N_v \\ + k_{LL} \cdot L^2 + k_{NN} \cdot N_v^2 \end{aligned} \tag{14}$$

with k being corresponding weight factors. These factors were determined by the mentioned parameter fitting for each kernel. In the same way the delay of the sorting kernel is approximated by

$$\begin{aligned} t_{sort}(L, L_{in}) = k_0 + k_{L_{in}} \cdot L_{in} + k_L \cdot L \\ + k_{LL_{in}} \cdot L \cdot L_{in} \\ + k_{L_{in}L_{in}} \cdot L_{in}^2 + k_{LL} \cdot L^2. \end{aligned} \tag{15}$$

The timing engine of the framework internally uses these timing models to optimally insert and balance the registers along the pipeline stages in the architecture. To further reduce the power and clock load, the framework also supports clock gating, latch-based design and some further optimizations [13].

IV. Results

This section presents error-correction performance, implementation results and corresponding trade-offs for various SC and SCL decoders. All codes used in the evaluation were constructed according to [26] with a design Signal-to-Noise-Ratio (SNR) of 0 dB for an Additive White Gaussian Noise (AWGN) channel. The architectures of the decoders were automatically generated with the aforementioned framework and implemented in a 28 nm Fully Depleted Silicon on Insulator (FD-SOI) technology under worst case Process, Voltage and Temperature (PVT) conditions (125 °C, 0.9 V for timing, 1.0 V for power). Synthesis is performed with the Design Compiler, Placement & Routing is carried out with the IC-Compiler, both from Synopsys. Power numbers are calculated with back-annotated wiring data. Error-correction performance simulations are carried out with Binary Phase Shift Keying (BPSK) over an AWGN channel. Channel values and internal LLRs are quantized with 6 bit without fractional bits. The PMs for SCL are quantized with 8 bit. This quantization scheme is used for all presented error-correction performance simulations and implementations and matches floating point precision with negligible FER performance impact. Presented throughput numbers are always given as coded throughput. For discussion of FER performance and implementation cost, we select an SNR of 4 dB as reference point.

A. Impact of List Sizes

We investigated the impact of list sizes on the error-correction performance and implementation cost. For this, a Polar code of length $N = 128$ was selected to enable list sizes up to 8 with feasible area consumption. List sizes 2, 4 and 8 with and without CRC were considered.

The maximum achievable frequency for the SCL4-CRC6 decoder was 499 MHz resulting in a throughput of 64 Gbit/s. Hence, we used this throughput as reference and set 64 Gbit/s respectively 499 MHz as target throughput constraint for all other decoders. The corresponding implementation results are listed in Table I.

Fig. 1 shows the corresponding FER performance. We can observe a gain of ~0.2 dB at an FER of 10^{-3} (SNR of 4 dB) for list size 2 compared to SC decoding. This gain diminishes for higher SNR. But it comes with an increase of 2.9× in area and 2.6× in power consumption compared to the SC decoder. List sizes larger than 2 do not provide additional gains for this short code while the implementation costs increase dramatically.

The situation changes when a CRC is used in the SCL decoder. The FER gain with $L = 2$ compared to SC decoding is ~0.3 dB and increases further up to ~0.7 dB for $L = 4$ and ~1.0 dB for $L = 8$. This comes at the cost of 3.3×, 8.0× and

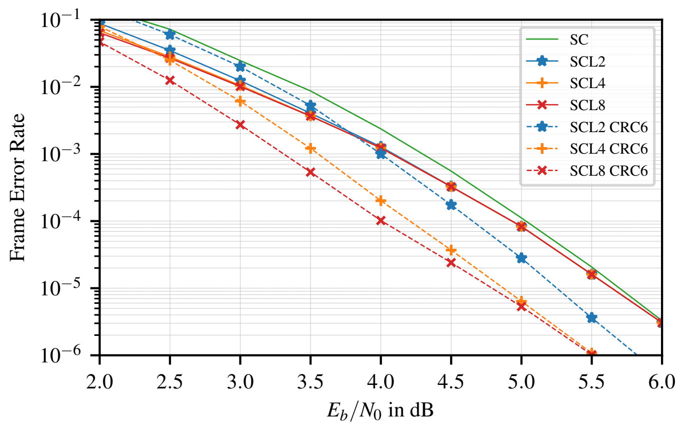

Fig. 1. Error-correction performance of Polar codes N=128, K=64,70

TABLE I
IMPLEMENTATION RESULTS OF DECODERS FOR POLAR CODES WITH N=128, K=64,70

	SC	SCL2	SCL4	SCL8	SCL2-CRC6	SCL4-CRC6	SCL8-CRC6
N	128	128	128	128	128	128	128
K	64	64	64	64	70	70	70
Frequency [MHz]	503	505	491	446	502	499	418
Throughput [Gbit/s]	64	65	63	57	64	64	53
Latency [ns]	17.9	37.6	53.0	94.2	41.8	64.1	141.3
Area [mm^2]	0.07	0.20	0.46	1.65	0.23	0.56	3.15
Area Eff. [Gbit/s/mm^2]	**931**	**322**	**138**	**35**	**283**	**113**	**17**
Power Total [W]	0.08	0.21	0.48	1.69	0.25	0.56	3.34
Energy Eff. [pJ/bit]	**1.28**	**3.21**	**7.67**	**29.61**	**3.87**	**8.75**	**62.46**

TABLE II
IMPLEMENTATION RESULTS OF DECODERS FOR POLAR CODES WITH N=1024, K=512,518

	SC	SCL2	SCL2-CRC6
N	1024	1024	1024
K	512	512	518
Frequency [MHz]	505	503	494
Throughput [Gbit/s]	517	516	506
Latency [ns]	110.9	292.0	309.7
Area [mm^2]	2.05	7.52	7.89
Area Eff. [Gbit/s/mm^2]	**252**	**69**	**64**
Power Total [W]	1.65	4.53	4.74
Energy Eff. [pJ/bit]	**3.20**	**8.79**	**9.37**

45.0× larger area for list sizes $L \in [2, 4, 8]$ compared to the SC decoder. The power consumption of decoders with CRC and $L \in [2, 4, 8]$ are 3.1×, 7.0× and 41.8× higher.

It is important to mention that, for this short code, list sizes larger than 2 without CRC do not provide much error-correction benefit. In contrast when using CRC, the FER performance can be further improved with increasing list sizes. List sizes larger than 4, however, only provide little benefit in error correction that is not in relation with the large increase in implementation costs.

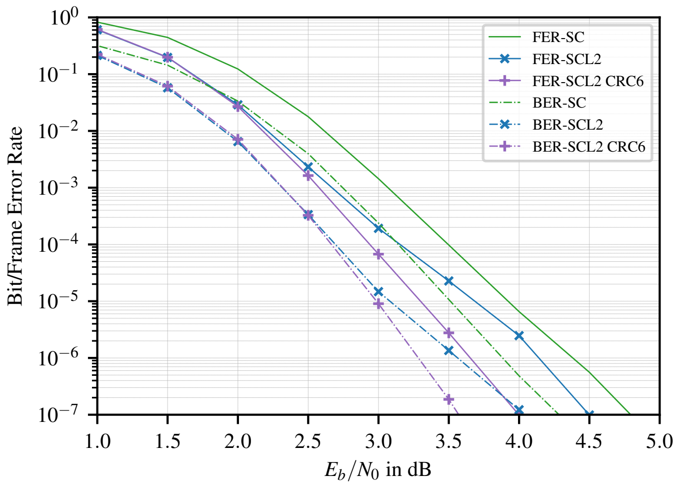

Fig. 2. Error-correction performance of Polar codes N=1024, K=512,518

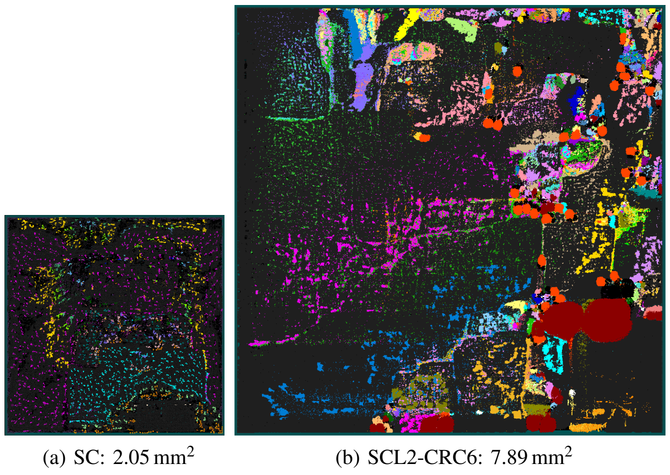

Fig. 3. Layouts of SC and SCL decoder, N=1024, K=512,518 with throughput of 517 Gbit/s (SC) and 506 Gbit/s (SCL2-CRC6) respectively; pipeline stages are represented by a different colors, memory is highlighted in dark-gray; Rate-1 nodes, sorter units and CRC components are colored in dark-red, orange-red and white for SCL2-CRC6 respectively; both layouts have the same scaling.

B. *SC/SCL Comparison*

Here we compare SC decoder with various SCL decoders with list size 2, with and without CRC. Code block size is 1024 and code rate 1/2 since this Polar code is commonly used in literature. Corresponding FER and Bit Error Rate (BER) error correction performance and implementation results are shown in Fig. 2 and Table II, respectively. Our SCL decoder with list size 2 achieves a throughput of 516 Gbit/s that is nearly the same throughput as the SC decoder. The SCL decoder with CRC achieves 506 Gbit/s. This is, to the best of our knowledge, the fastest Polar SCL decoder implementation.

Fig. 2 shows the gain in error-correction performance of an SCL decoder with list size 2 compared to an SC decoder. At an FER of 10^{-5} (SNR of 4 dB) we observe a gain of ~0.3 dB. This gain must be paid with an increased implementation complexity, i.e., the area increases by 3.7× and the power consumption by 2.7×. The SCL decoder with CRC further improves the FER by ~0.7 dB compared to the SC decoder at the cost of an 3.8× larger area and 2.9× in power. Corresponding layouts are shown in Fig. 3. The additional implementation costs for the CRC in the SCL decoder are

TABLE III
IMPLEMENTATION RESULTS OF SCL DECODERS WITH L=2 FOR POLAR CODE N=512, K=427

	Optimized for Area	Optimized for Frequ.	[12]
Area [mm^2]	*0.84*	1.63	*0.87*
Frequency [MHz]	214	*477*	*468*
Throughput [Gbit/s]	109	244	12
Latency [ns]	84.2	104.7	540.0
Area Eff. [Gbit/s/mm^2]	**130**	**150**	**14**
Power Total [W]	0.57	1.16	0.09
Energy Eff. [pJ/bit]	**5.25**	**4.74**	**7.25**

relatively small compared to the additional gain in FER performance (~0.4 dB). To be more precise, the increase in implementation costs do not stem from the CRC itself, but are caused by the need of exploring multiple paths in all Rate-1 nodes (see Section III-A). All Rate-1 nodes together occupy a cell area of 0.18 mm^2 which is 3.2 % of the total decoder cell area and are highlighted in dark-red in Fig 3b. In contrast, the CRC units only take 0.05 %. Furthermore, the PFT changes when using CRC, since the CRC bits replace frozen bits and thus change the rate of the Polar code (see Section III-C).

C. Comparison with State-of-the-Art

To the best of our knowledge there is only one known high-throughput implementation of an SCL decoder [12] with list size 2, also implemented in 28 nm FD-SOI technology. A fair comparison is challenging, since [12] was designed for a systematic Polar code, which provides a slightly better BER compared to a non-systematic code. In this paper, we considered non-systematic Polar code to enable the CRC on-the-fly calculation. To make the comparison as fair as possible, we used the same code size, code rate and internal quantization as the authors of [12]. Note, that the constraints on the node sizes given in Section III differ from the ones of the reference but it has no impact on the error-correction performance. Since this decoder was implemented in the same technology we generated two decoders with our framework. In one we constrained the area to the area of [12]. In the other we constrained the frequency to the frequency of [12]. Implementation results of the two decoders are shown in Table III. According to our comparison methodology our decoders outperform the SOA in throughput and area efficiency while providing even a better energy efficiency.

V. CONCLUSION

In this paper we presented a detailed analysis on the trade-offs between error-correction performance versus implementation costs for SC and SCL decoders. We studied the impact of list sizes and showed the advantage of CRC supported SCL decoding. This analysis is based on 10 advanced Polar SCL decoder implementations for very high throughput whose architectures were automatically generated by a framework. The generated decoders can achieve throughput up to 516 Gbit/s in 28 nm CMOS FD-SOI technology under worst case PVT conditions.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[2] 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, 2009.

[3] 3GPP, "NR; Multiplexing and channel coding," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, Apr. 2018.

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

[5] 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.

[6] S. M. Abbas, Y. Fan, J. Chen, and C. Y. Tsui, "High-throughput and energy-efficient belief propagation polar code decoder," *Trans. VLSI Syst.*, vol. 25, no. 3, pp. 1098–1111, Mar. 2017.

[7] A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Belief propagation decoding of polar codes on permuted factor graphs," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 1–6.

[8] 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.

[9] ——, "Fast list decoders for polar codes," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 318–328, Feb. 2016.

[10] P. Schläfer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *SiPS 2013 Proceedings*, Oct. 2013, pp. 153–158.

[11] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electron. Lett.*, vol. 51, no. 10, pp. 762–763, 2015.

[12] P. Giard, A. Balatsoukas-Stimming, T. C. Müller, A. Burg, C. Thibeault, and W. J. Gross, "A multi-Gbps unrolled hardware list decoder for a systematic polar code," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov. 2016, pp. 1194–1198.

[13] T. Lehnigk-Emden, M. Alles, C. Kestel, and N. Wehn, "Polar code decoder framework," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2019, p. 2.

[14] P. Giard, C. Thibeault, and W. J. Gross, *High-Speed Decoders for Polar Codes*, 1st ed. Springer International Publishing, 2017.

[15] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 447–449, Jun. 2008.

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

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

[19] ——, "Simplified successive-cancellation list decoding of polar codes," in *2016 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016, pp. 815–819.

[20] ——, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5756–5769, Nov. 2017.

[21] ——, "Fast simplified successive-cancellation list decoding of polar codes," *2017 IEEE Wirel. Commun. Netw. Conf. Workshop WCNCW*, pp. 1–6, Mar. 2017.

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

[23] S. Shi, B. Han, J. L. Gao, and Y. J. Wang, "Enhanced successive cancellation list decoding of polar codes," *IEEE Commun. Lett.*, vol. 21, no. 6, pp. 1233–1236, Jun. 2017.

[24] H. Li, "Enhanced metric sorting for successive cancellation list decoding of polar codes," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 664–667, Apr. 2018.

[25] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "On metric sorting for successive cancellation list decoding of polar codes," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1993–1996.

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