Scalable Successive-Cancellation Hardware Decoder for Polar Codes

Alexandre J. Raymond and Warren J. Gross
Department of Electrical and Computer Engineering
McGill University
Montréal, Québec, Canada
alexandre.raymond@mail.mcgill.ca, warren.gross@mcgill.ca

Abstract—Polar codes, discovered by Arıkan, are the first error-correcting codes with an explicit construction to provably achieve channel capacity, asymptotically. However, their error-correction performance at finite lengths tends to be lower than existing capacity-approaching schemes. Using the successive-cancellation algorithm, polar decoders can be designed for very long codes, with low hardware complexity, leveraging the regular structure of such codes. We present an architecture and an implementation of a scalable hardware decoder based on this algorithm. This design is shown to scale to code lengths of up to \( N = 2^{20} \) on an Altera Stratix IV FPGA, limited almost exclusively by the amount of available SRAM.

Index Terms—Error-correcting codes, polar codes, successive-cancellation decoding, hardware implementation.

I. INTRODUCTION

Since their introduction in 2008, polar codes [1] have attracted a lot of attention from the information theory community, as they are the first codes to provably achieve channel capacity, asymptotically in code length.

Although initially only defined for the binary erasure channel (BEC), they were later extended to other models, such as the additive white Gaussian noise (AWGN) channel [2].

Their recursive construction was shown to support low-complexity implementations of the successive-cancellation (SC) algorithm in hardware [3][4]. Those low-complexity decoders can in turn be used as components in more complex schemes, such as list decoding [5][6] and concatenated coding [7], which improve the error-correction performance of polar codes at finite lengths.

The remainder of this paper is structured as follows. Section II provides background information on polar codes and SC decoding. Then, Section III details the proposed architecture. Section IV analyzes FPGA implementation results, while Section V concludes this work.

II. BACKGROUND

Polar codes are a class of linear block codes based on a recursive definition. They are constructed using a generator matrix \( G \), obtained from the base matrix \( F_2 \equiv \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \), using

\[
G = F_N \equiv (F_2)^{\otimes n},
\]

where \( N = 2^n \) is the code length, and \( \otimes \) represents the Kronecker product. In this paper, we use \( u \) to denote an information vector, \( x \) for a codeword, \( y \) for a received vector, and \( \hat{u} \) for the information vector estimated by the decoder.

These codes can be decoded using a recursive, multi-stage structure featuring \( n \) stages of \( N/2 \) nodes, yielding a complexity \( O(N \log N) \) [1].

To simplify their implementation in hardware, decoding can be carried out in the log-likelihood-ratio (LLR) domain, where the SC equations become the standard sum-product algorithm (SPA) equations, which can be approximated using the well-known min-sum algorithm (MSA) [8]:

\[
\lambda_f(\lambda_a, \lambda_b) \approx \text{sign}(\lambda_a)\text{sign}(\lambda_b) \min(|\lambda_a|, |\lambda_b|), \tag{1}
\]

\[
\lambda_g(\hat{s}, \lambda_a, \lambda_b) = \lambda_a(-1)^{\hat{s}} + \lambda_b, \tag{2}
\]

where \( \hat{s} \) designates a partial sum. This approximation yields a performance degradation of \( \sim 0.1\text{dB} \) over SPA, as illustrated in Figure 1, although this gap tends to shrink for higher-rate (\( R = k/N \)) codes.
III. ARCHITECTURE

The architecture presented in this paper is based on the semi-parallel decoder of [3], and introduces modifications aiming to improve its scalability with respect to code length. This decoder uses a fixed datapath, and operates under resource constraints, where only $P \ll N/2$ processing elements (PE) are implemented. This limitation, however, only impacts throughput minimally [8].

Figure 2 provides a top-level overview of the redesigned decoder architecture, while its various changes are discussed in the following sections.

A. Memory Improvements

Unlike [3], which makes use of a single SRAM to store all LLRs, this improved architecture relies on two separate types of memories: channel and internal. This separation allows full-throughput operation of the decoder by supporting the loading of a subsequent frame into the channel memory, without write contention, while the previous one is still being processed. This is made possible by the fact that, per the structure of the decoding graph, channel LLRs are not directly required in the second half of the decoding process, i.e. after bit $i = N/2$ and stage $l = (n - 1)$.

Furthermore, the improved design does away with asymmetric read/write ports in its SRAMs. Those memories are replaced by pairs of $P$-LLR wide SRAMs, whose outputs are concatenated into $2P$-LLR words consumed by the processing elements, whose own $P$-LLR outputs are written to each SRAM in sequence. Note that the $\&$ operator used in Figure 2 symbolizes concatenation, with sign extension if needed.

B. Quantization

The separation of the channel and internal memories, described in Section III-A, also makes it possible to use distinct quantization levels for each memory. This enhancement is suggested by a characteristic of the successive-cancellation algorithm, namely that (2) affects the range of the computations in each successive stage, while their precision remains unchanged by both operations. It follows that the values processed by lower-indexed stages require more range than those in the higher ones.

Since the decoder must retain an entire $N$-LLR frame in memory, the channel SRAMs account for nearly half of the decoder’s soft information storage requirements [3]. A lower quantization for this memory therefore reduces the decoder area significantly.

Quantization is denoted using shorthand $(Q_i, Q_e, Q_f)$, which indicates the number of integer bits for internal LLRs, integer bits for channel LLRs, and fractional bits for both types, respectively; $Q = Q_i + Q_f$ and $Q_e = Q_i + Q_f$ are also used to refer to the total number of quantization bits in each case.

Simulations showed that full-range quantization does not benefit error-correction performance; much lower levels can match a floating-point implementation. Specifically, we carried out those simulations for codes of length $N = 2^{15}$, with $R \in \{0.25, 0.50, 0.75, 0.90\}$; results are summarized in Table I. We found that, for those codes, 6–8 bits of quantization suffice for good error-correction performance (within $\sim 0.1$dB of floating point MSA), depending on their rate, as shown in Figure 1. We also noticed that higher-rate codes tend to require fewer bits of fractional precision and integer range for internal LLRs, but more bits of integer range for channel ones.
C. Chained PE

This architecture makes use of a chained PE in stage 0, carrying out functions $\lambda_f$ and $\lambda_g$ in a single clock cycle (CC). The concept behind this improvement was introduced in [9], while the restricted implementation used in this paper, targeting only stage 0, was independently proposed in [4].

This chained PE relies on the specific schedule of the polar decoding graph, in which stage 0 is always activated twice in a row, using the same operands: first for function $\lambda_f$, and then for function $\lambda_g$, using the result of $\lambda_f$. By chaining both operations in a special PE, we can output two decoded bits $\hat{u}_{(i,i+1)}$ at once, yielding a $(N/2)$-CC reduction in decoding latency.

This behavior is illustrated in Table III, specifically in clock cycles $\{3, 6, 13, 16\}$. In those cases, the computations of functions $\lambda_f$ and $\lambda_g$ are performed in the same clock cycle, yielding two decoded bits simultaneously.

The chained PE does not incur any overhead over the regular PE. The data dependency present in-between functions $\lambda_f$ and $\lambda_g$, satisfied by the sign of $\lambda_f$, occurs late in the processing of $\lambda_g$, and can be computed very rapidly.

D. Semi-Parallel Partial-Sum Encoder

The main factor limiting the scalability of [3] is the growing complexity of its partial-sum update logic. In this paper, we introduce an encoder-based alternative inspired by the design of [9], which proposed a fully-parallel partial-sum computation module. Our implementation extends this encoder, adapting it to a novel semi-parallel architecture. This architecture operates over multiple clock cycles and uses a fixed datapath, removing it from the decoder's critical path altogether.

This encoder is triggered after decoding-stage 0, and processes two decoded bits at a time. Figure 3 illustrates its structure, a mirrored version of the decoding graph, in which the $f$ nodes are defined as binary additions (XOR), and the $g$ nodes, as pass-through connections:

$$\hat{f}(\hat{u}_a, \hat{u}_b) = \hat{u}_a \oplus \hat{u}_b,$$

$$\hat{g}(\hat{u}_a) = \hat{u}_a.$$  (3)  (4)

As in the decoding graph, the nodes are associated into $N/2$ pairs per stage. Those pairs are processed by the $P/2$ encoding PEs.

In order to make the design scalable, a semi-parallel architecture was chosen for the encoder. Since the encoding graph mirrors the decoding graph, their schedules are very similar. The encoding schedule is illustrated in Table III, where $e$ denotes the activation of encoding stage $\ell_{enc}$. Due to the semi-parallel nature of the encoder, stages which are handled in multiple clock cycles are denoted using a subscript, e.g. $e_0$.

In Figure 3, the subgraph highlighted in bold illustrates the nodes activated to calculate partial sums $s_{i,0}$ and $s_{j,2}$. Those two values are subsequently used to evaluate $\lambda_g$ nodes in stage $l = 1$ of the decoding graph.

The partial-sum encoder follows a schedule similar to that of the decoding, although with half as many processing elements; those processing elements produce two values instead of one, since they are not restricted by a data dependency as the decoding PEs are. The encoder thus increases latency by $(\frac{N}{P}(P - 1) + \frac{N}{P} \log_2(\frac{N}{P}) - \log_2 P + 2)$ CC, or ~67% for $P = 64$, but allows higher operating frequencies, for a net throughput gain.

Using $P/2$ encoding PEs, the encoder can make use of $P$-bit wide words in the $s$ SRAMs, allowing the decoder to retrieve $P$ partial sums simultaneously during decoding, in a single clock cycle. Furthermore, because of the specific structure of the encoding graph, the values stored in memory are properly aligned for direct consumption by the decoding PEs, via a fixed datapath.

Note that the internal partial sums $s_{0,j}$ correspond to $\hat{u}_i$, where $i$ is the bit-reversed [1] value of $j$. Furthermore, $\hat{s}_{n,j}$ yields an estimation of codeword value $\hat{x}_j$, where $i$ is again bit-reversed $j$. As part of the encoding process resulting in this estimated codeword $\hat{x}$, the encoder creates internal estimations $\hat{s}_{l,j}$, which are required by $\lambda_g$ during the decoding process.

In a non-systematic polar decoder, it is not necessary to evaluate $\hat{x}$ completely, which saves a final encoding stage after $\hat{u}_{N-2}$ and $\hat{u}_{N-1}$ are decoded. However, in a systematic decoder [10], those extra steps could be carried out to obtain $\hat{x}$, which is required to retrieve the original information vector, while avoiding the need for extra hardware to perform the additional encoding step.

IV. EXPERIMENTAL RESULTS

The various characteristics of this architecture, explored in Section III, are summarized in Table IV. In this table, latency

---

**TABLE I**

| $R$ | $Q_{1}$ | $Q_{2}$ | $Q_{3}$ |
|-----|---------|---------|---------|
| 0.25 | (6, 3, 2) | (6, 3, 2) | (6, 4, 1) |
| 0.50 | (6, 3, 2) | (6, 4, 1) | (6, 4, 0) |

---

**TABLE II**

| $N$ | $R$ | $P$ | Q(e) | LUT | FF | SRAM (bits) | $f_{\text{max}}$ (MHz) | $T_{\text{FP}}$ (Mqps) |
|-----|-----|-----|------|-----|----|-------------|-----------------------|---------------------|
| 2^{15} | 0.25 | 64 | (6, 3, 2) | 4,161 | 1,629 | 510,464 | 156 | 15 |
| 2^{15} | 0.50 | 64 | (6, 3, 2) | 4,161 | 1,629 | 510,464 | 156 | 29 |
| 2^{15} | 0.75 | 64 | (6, 4, 1) | 3,731 | 1,496 | 477,440 | 155 | 43 |
| 2^{15} | 0.90 | 64 | (6, 4, 0) | 3,263 | 1,304 | 411,648 | 157 | 56 |
| 2^{16} | 0.25 | 64 | (6, 4, 0) | 3,414 | 1,316 | 821,248 | 157 | 57R |
| 2^{16} | 0.50 | 64 | (6, 4, 0) | 3,548 | 1,349 | 3,278,848 | 140 | 51R |
| 2^{16} | 0.75 | 64 | (7, 4, 0) | 3,956 | 1,866 | 13,109,248 | 102 | 38R |
| 2^{16} | 0.90 | 64 | (7, 4, 0) | 3,927 | 1,427 | 444,672 | 153 | 57R |
| 2^{17} | 0.25 | 64 | (7, 3, 0) | 3,725 | 1,965 | 411,904 | 157 | 57R |
| 2^{17} | 0.50 | 64 | (7, 3, 0) | 3,927 | 1,427 | 444,672 | 153 | 57R |
| 2^{17} | 0.75 | 64 | (7, 5, 0) | 3,731 | 1,496 | 477,440 | 155 | 57R |
| 2^{17} | 0.90 | 64 | (7, 5, 0) | 3,548 | 1,866 | 411,648 | 157 | 56 |

Decoder from [3]

| $N$ | $R$ | $P$ | $\hat{s}$ | $\hat{x}$ | $\hat{s}_{\text{min}}$ | $\hat{s}_{\text{max}}$ |
|-----|-----|-----|----------|----------|-------------------|-------------------|
| 2^{18} | 0.25 | 64 | 58,480 | 33,451 | 364,288 | 66 | 31R |
| 2^{18} | 0.50 | 64 | 221,471 | 131,764 | 1,445,632 | 10 | 6R |
At \( N = 2^{17} \), the largest code length supported by our previous-generation decoder, this improved architecture uses 81 times less look-up tables (LUT), 104 times fewer flip-flops (FF), has a maximum operating frequency 16 times higher, and a throughput 11 times greater, using the same parameters.

V. CONCLUSION

In this paper, we presented a scalable architecture for SC decoding of polar codes. This decoder features a semi-parallel, encoder-based partial-sum update module. This module utilizes SRAM for storage, and makes use of a fixed datapath. Additionally, this architecture leverages a multi-level quantization scheme for LLRs, decreasing memory use and decoder area. This state-of-the-art decoder was synthesized for an Altera Stratix IV FPGA targeting up to \( N = 2^{20} \), limited almost exclusively by the amount of available SRAM.

ACKNOWLEDGEMENT

The authors would like to thank Gabi Sarkis and Pascal Giard, of McGill University, for helpful discussions.

REFERENCES

[1] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), 2008, pp. 1173–1177.
[2] I. Tal and A. Vardy, “How to construct polar codes,” arXiv/CoRR, vol. abs/1105.6164, 2011.
[3] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” Signal Processing, IEEE Trans. on, vol. 61, no. 2, pp. 289–299, 2013.
[4] A. Mishra, A. J. Raymond, L. G. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, “A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS,” in Proc. IEEE Asian Solid-State Circuits Conf (A-SSCC), 2012, to appear.
[5] I. Tal and A. Vardy, “List decoding of polar codes,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), 2011, pp. 1–5.
[6] A. Balatsoukas-Stimming and A. Burg, “Tree search architecture for list SC decoding of polar codes,” arXiv/CoRR, vol. abs/1303.7127, 2013.
[7] H. Mahdavifar, M. El-Khamy, Jungwon Lee, and I. Kang, “On the construction and decoding of concatenated polar codes,” arXiv/CoRR, vol. abs/1301.7491, 2013.
[8] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” in Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), 2011, pp. 1665–1668.
[9] Chuan Zhang, Bo Yuan, and K. K. Parhi, “Reduced-latency SC polar decoder architectures,” in Proc. IEEE Int. Conf. on Commun. (ICC), 2012, pp. 3471–3475.
[10] E. Arikan, “Systematic polar coding,” IEEE Commun. Lett., vol. 15, no. 8, pp. 860–862, 2011.