doi: `http://dx.doi.org/10.1109/IPEC.2012.6522661`

A Successive Cancellation Decoder ASIC for a 1024-bit Polar Code in 180nm CMOS

A. Mishra*, A. J. Raymond†, L. G. Amaru*, G. Sarkis†, C. Leroux‡, P. Meinerzhagen*, A. Burg*, and W. J. Gross†
*Telecommunications Circuits Laboratory, EPFL, Lausanne, Switzerland
†Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada
‡IMS Laboratory, Institut Polytechnique de Bordeaux, Bordeaux, France

***Abstract*—This paper presents the first ASIC implementation of a successive cancellation (SC) decoder for polar codes. The implemented ASIC relies on a semi-parallel architecture where processing resources are reused to achieve good hardware efficiency. A speculative decoding technique is employed to increase the throughput by 25% at the cost of very limited added complexity. The resulting architecture is implemented in a 180nm technology. The fabricated chip can be clocked at 150 MHz and uses 183k gates. It was verified using an FPGA testing setup and provides reference for the true silicon complexity of SC decoders for polar codes.**

I. Introduction

Polar codes, introduced by Arıkan in [1], are a new class of error-correcting codes that provably achieve the capacity of symmetric binary-input discrete memoryless channels. Recently, polar codes have also been extended to the additive white Gaussian noise channel [2] and to other relevant channels [3]. An important property of polar codes is that the complexity of the encoder and of a successive cancellation (SC) decoder scales only as $O(N\cdot\log N)$ [1], where $N = 2^n$ is the code length. However, the scaling behavior usually provides only little information on the true silicon complexity of a corresponding circuit and the availability of hardware efficient VLSI architectures for a new type of code is key to demonstrate the practical relevance of these new codes.

A first step toward this objective has been made in a number of recent publications that propose different implementation strategies for the decoder. The first high-level considerations of a potential SC decoder implementation were already described in [1]. Later, more hardware efficient sequential architectures were described in [4], [5] and [6]. A comparison that shows the superiority of SC decoding over the alternative belief propagation algorithm at comparable complexity was provided in [7], together with a semi-parallel architecture that further reduces complexity compared to [4]. Unfortunately, all of the above publications consider only the register-transfer level and provide no ASIC implementation.

Outline and Contributions: In this paper, we describe the first ASIC implementation of a decoder for polar codes. To this end, we first summarize the successive cancellation decoding algorithm and highlight the scheduling of computations in the decoding process. We also describe the semi-parallel decoder architecture in Section II. In Section III, we propose an architectural modification of the semi-parallel decoder that improves throughput by 25% without noticeable overhead and describe the circuit-level details of the processing elements. Finally, ASIC implementation and measurement results are summarized in Section IV.

II. Decoding Algorithm and Architecture

In this section, we first review the SC decoding algorithm, then the semi-parallel SC decoder architecture of [7].

A. Algorithm

Let the polar code under consideration have length N and code rate k/N. Denote the binary input vector $(u_0, u_1, .., u_{N-1})$ by u_0^{N-1}. Only k elements of u_0^{N-1} carry information: the remaining $N-k$ bits are fixed to 0 and are called the *frozen bits*. The *frozen bit* indices are determined by the communication channel type and conditions [8]. Both information and *frozen* bits are encoded into the codeword $x_0^{N-1} = u_0^{N-1}G$, where G is the polar code generator matrix. The codeword x_0^{N-1} is sent over the communication channel. At the output of the channel, y_0^{N-1} is received and the corresponding log-likelihood ratios (LLRs) L_0^{N-1} are calculated. An SC decoding algorithm produces an estimate $\hat{u}_i$ of a bit u_i given L_0^{N-1}, the previously decoded bits $\hat{u}_0^{i-1}$, and the *frozen bit* set. If u_i is in the *frozen bit* set, then $\hat{u}_i = 0$; otherwise, $\hat{u}_i$ is decoded based on the LLR of $\hat{u}_i$ ($L_{\hat{u}_i}$). As shown in (1), $L_{\hat{u}_i}$ is a function of L_0^{N-1} and $\hat{u}_0^{i-1}$. The dependency of $L_{\hat{u}_i}$ on $\hat{u}_0^{i-1}$ imposes the sequential decoding order $\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{N-1}$.

$$\hat{u}_i(L_0^{N-1}, \hat{u}_0^{i-1}) = \begin{cases} 0, & \text{if } L_{\hat{u}_i}(L_0^{N-1}, \hat{u}_0^{i-1}) \geq 0 \\ 1, & \text{otherwise.} \end{cases} \tag{1}$$

B. Algorithm to Architecture Mapping

The SC decoding algorithm leads to a regular data dependency graph (DDG) [1]. In Fig. 1, the DDG for $N = 8$ is depicted. Each computational node in the graph can operate either on likelihood ratios, as proposed in [1], or directly on LLRs, as proposed in [4]. The latter yields a considerable reduction in complexity at the cost of only a minor degradation in error-rate performance. There are two different types of nodes: f and g. The f-type node receives as input two LLRs (L_a, L_b) and calculates an output LLR as

$$L_f(L_a, L_b) = \text{sign}(L_a)\cdot \text{sign}(L_b)\cdot \min(|L_a|, |L_b|). \tag{2}$$

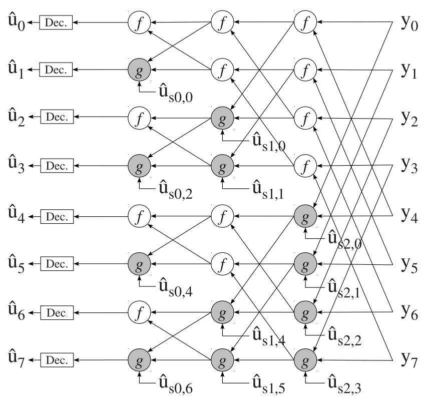

Fig. 1. DDG of a SC polar decoder for a code of length $N = 8$.

The g-type node also receives as input two LLRs (L_a, L_b), in addition to a partial modulo-2 sum $(\oplus)$ of previously estimated bits $(\hat{u}_s)$. In this case, the output LLR is

$$L_g(\hat{u}_s, L_a, L_b) = L_a \cdot (-1)^{\hat{u}_s} + L_b. \tag{3}$$

The computation of $L_{\hat{u}_i}$ is equivalent to a topological-order traversal of the graph starting from L_0^{N-1} on the right-hand side. Note that the maximum-length path in the topological-order traversal of the graph is not $\log N$, but $2N-2$ due the data dependency of the g-nodes on previously decoded bits $\hat{u}_s$.

Directly mapping the computational nodes in the DDG to $N \cdot \log N$ hardware processing elements (PEs) leads to an isomorphic implementation with a critical path that spans $2N-2$ PEs which is impractical and hardware inefficient. The line architecture [4] [5] reduces the number of PEs to $N/2$ by means of iterative decomposition at the cost of $2N$ memory elements (MEs) while maintaining approximately the same decoding latency. However, the regular structure of the DDG permits further sharing of PEs, increasing the utilization rate of PEs with minor latency penalties. To exploit this opportunity, the semi-parallel SC decoder [7] provides a tunable number $(P < N/2)$ of PEs. The time scheduling for a semi-parallel SC decoder with $N = 8$ and $P = 4$ is denoted by CC_{orig} in Fig. 3. The decoding latency for the semi-parallel SC decoder was $2N + \frac{N}{P} \cdot \log(\frac{N}{4P})$ and the memory requirements remained the same as that of the line architecture.

C. Architecture

The architecture described in [7] divides the semi-parallel SC decoder into four major units: an array of P PEs, the LLR memory, the partial-sum update logic and the associated storage registers, and the controller, as shown in Fig. 2. Additionally, a read-only memory (ROM), implemented as a lookup-table (LUT), is used to store the indices of the frozen bits for a given channel signal-to-noise ratio, chosen as per [8].

Processing Elements: Without resource sharing (i.e., $P = N/2$ PEs), the employed SC decoder architecture estimates the bits $\hat{u}_0^{N-1}$ sequentially, in $2N-2$ clock cycles. The corresponding decoding schedule is identified by CC_{orig} in Fig. 3. However, since it was shown in [7] that a small P is sufficient to achieve 90% of the throughput of a decoder with $N/2$ PEs we instantiate only $P = 64$ PEs for the chosen code block size of $N = 2^{10}$). Each PE implements both the f and g functions in the LLR domain (2) and (3), using the sign-and-magnitude representation.

LLR Memory: The LLR memory allows the reuse of intermediate results from PE calculations during the decoding process. Unlike [7], our LLR memory is implemented using registers, which are connected to the PEs using a multiplexer network. Since each PE uses two Q-bit values to calculate one Q-bit value, the registers are of size Q.

Partial Sums: Throughout the decoding process, the processing elements need partial sums $\hat{u}'_{s,z}$ of the estimated bits $\hat{u}_i$ to compute L_g. These partial sums are calculated through continuous updates of the stored $\hat{u}'_{s,z}$ during the decoding process to yield

$$\hat{u}'_{s,z} = \bigoplus_{j=0}^{N-1} \hat{u}_j \cdot (\beta(j,s) \cdot \delta(j,s,z)) \tag{4}$$

$$\beta(j,s) = \overline{B(s,j)}$$

$$\delta(j,s,z) = \prod_{v=0}^{s-1} (\overline{\overline{B(s-v-1,z)} + B(v,j)}),$$

where $B(s,j) = \frac{j}{2^s} \bmod 2$, and where $\beta(j,s)$ and $\delta(j,s,z)$ are masks that indicate if $\hat{u}_j$ contributes to $\hat{u}'_{s,z}$ or not. The operator $\prod$ denotes the binary product, and $\delta(\cdot) = 1$ when $s = 0$. Note that this expression already takes into account the time multiplexing introduced in [7], which reduces the amount of memory needed for the partial sums from $O(N \cdot \log N)$ to $O(N)$.

The hardware for the update corresponds to $(N-1)$ registers and a small amount of combinational logic in front of each partial sum storage element, as illustrated in Fig. 2. This logic computes $\beta(\cdot)\delta(\cdot)$, with s and z being the hardwired indices of $\hat{u}'_{s,z}$. If any of those blocks yields true for a given decoded bit, $\hat{u}_i$ is added $(\oplus)$ to the current content of the corresponding storage element.

Controller: The controller coordinates the decoding process with three state counters: the currently decoded bit index i, the current stage index s, and the portion of a stage p_s being processed, $0 \le p_s < \lceil \frac{2^s}{P} \rceil$. The bit index i is implemented using a sequential counter incremented whenever s reaches 0. The stage number s is a function of both i and p_s: when $p_s \ge \lceil \frac{2^s}{P} \rceil$, i and s are updated. Because some temporary computations are still available in the LLR memory, not all stages have to be evaluated when decoding a bit i. Specifically, s is set to the index of the first bit equal to 1 in the binary representation of the updated i, or to $n-1$ if i has wrapped around $N-1$. We notice that stage s requires 2^s PE computations. Due to the time sharing of only P PEs this requires $\lceil \frac{2^s}{P} \rceil$ clock cycles that are counted by p_s.

In addition, the controller is in charge of selecting the appropriate inputs to the PEs, selecting the right function to perform, and selecting the proper memory locations (LLR and

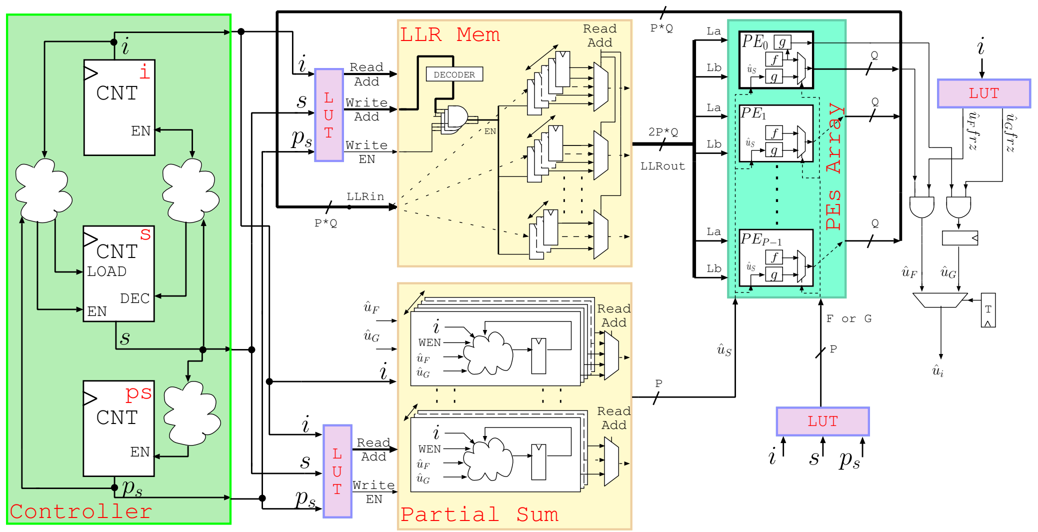

Fig. 2. Polar decoder high-level architecture

partial sums) to update. These signals are calculated from i, s, and p_s using small LUTs, as indicated in Fig. 2.

III. Architectural Improvements

A. Concurrent Decoding in Stage 0

In order to improve the throughput of the decoder, the architecture is modified to reduce the number of cycles required to decode one codeword by $N/2$. The improvement is achieved by decoding two bits at a time whenever the decoder is in the last stage ($s = 0$), which is possible by exploiting the fact that two subsequent $\hat{u}_i$ are obtained from f and g nodes that take the same input LLRs. Unfortunately, the g-node needs the output of the preceding f node as its $\hat{u}_{s0,z}$ input. Therefore the g-node output cannot be computed before the f node ouput is available. The semi-parallel architecture in [7] computes the g-node in the clock cycle after the f-node. In our architecture, it is possible to compute both node outputs in the same cycle. To this end, the two possible g-node outputs are calculated speculatively while the output of f is calculated. The correct g-node output is then selected with a negligible additional combinational delay. Fig. 3 shows an example of the new, shortened schedule of f- and g-nodes for concurrent decoding of two bits in case of $N = 8$ and $P = 4$, denoted CC_{conc}. Overall the number of cycles for decoding is reduced by $N/2$.

In terms of area, it should be noted that only one of the P processing elements must be able to perform this concurrent decoding. Hence, the increase in silicon area is hardly noticeable. However, since two bits are decoded concurrently, both have to be considered also in the $\hat{u}_s$ memory update logic. This extra logic renders the corresponding hardware slightly more complex: the increase in total area due to this change was only 1.62%. We note that this speculative approach is similar to the one presented in [6]; however, it is applied to a very different baseline architecture and is much simpler since it applies only to one stage in the decoder. Furthermore, it entails almost no additional hardware.

B. Optimized PE Implementation

The proposed architecture, used by all PEs, improves upon the PE architecture in [7], which merges both functions f and g in a single PE and thus shares a comparator and an XOR gate between the two functions. The LLRs are stored in sign-and-magnitude form. The value of $\text{sign}(L_f)$ is given by $\text{sign}(L_a) \oplus \text{sign}(L_b)$, whereas $|L_f|$ is $\min(|L_a|, |L_b|)$, as shown in (2) and (3).

The PE architecture in [7] calculates $|L_g|$ by converting $\min(|L_a|, |L_b|)$ to two's complement representation, and adding it to $\max(|L_a|, |L_b|)$. The hardware for this operation has a long carry path through the magnitude comparator, the two's complement conversion block (an adder), and the adder. We note that $|L_g|$ has three possible magnitudes, namely $(|L_a|-|L_b|)$, $(|L_b|-|L_a|)$, and $(|L_a|+|L_b|)$. The improvement in the proposed architecture comes from calculating all the possible values of $|L_g|$ simultaneously in sign-and-magnitude form; selecting the correct output based on $\hat{u}_s$, $\text{sign}(L_a)$, and $\text{sign}(L_b)$; and finally saturating as required. The change is marked as *PE Mod* in Fig. 4. The value of $\text{sign}(L_g)$ is given by $\hat{u}_s \oplus \text{sign}(L_a)$ when $|L_a| > |L_b|$, and $\text{sign}(L_b)$ otherwise. Compared to the proposal in [7], this optimized architecture results in 50% reduction of the delay through the PE with an increase of 30% in its area. However, since the PEs only make up 8.5% of the total core area (for $P = 64$), the overall impact is small and the area-delay product of the circuit improves significantly.

Furthermore, a special PE—named PE_0 in Fig. 2—is introduced, which computes two decoded bits at a time as described in Section III-A. PE_0 has an additional g node output for concurrent decoding, which is used in stage 0, as shown in Fig. 3. We note that PE_0 does not replicate a full g node but shares the speculative computations of the normal PE. The implementation uses 8 additional 2-input MUXs as compared to normal PE. PE_0 also functions as a normal PE when used in stages 1 and 2. As a result, PE_0 is 44% larger than the

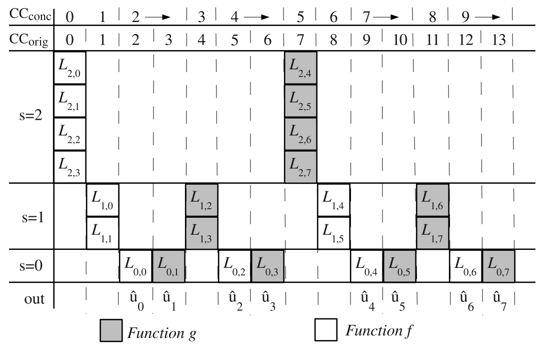

Fig. 3. Schedule for original and improved semi-parallel SC decoder.

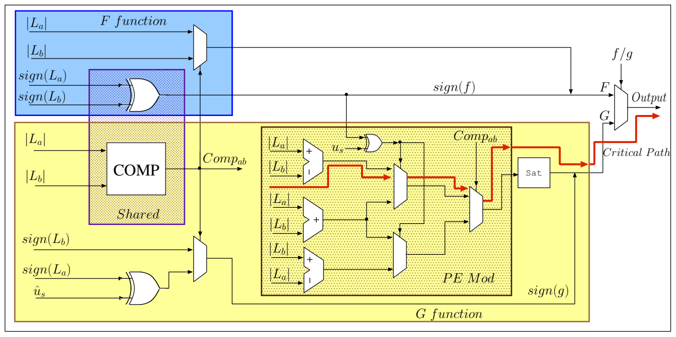

Fig. 4. RTL architecture of a standard PE

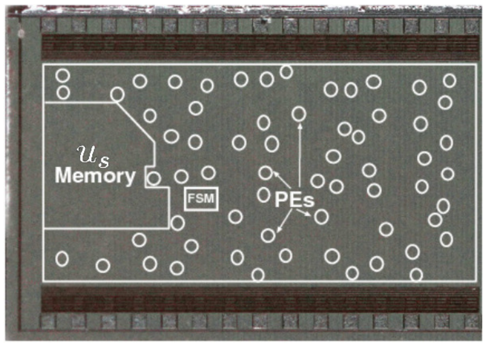

Fig. 5. Microphotograph of the SC Polar Decoder chip.

TABLE I
SUMMARY OF MEASURED RESULTS FOR SC POLAR DECODER WITH $N = 1024$, $k = 512$, $P = 64$, AND $Q = 5$

Technology	180 nm
Core area	1.71 mm^2
Chip area	1.72 mm^2
Gate count	183,637
Frequency	150 MHz
Throughput	49 Mbps
Voltage	1.3 V
Power	67 mW
Energy efficiency	1.37 nJ/bit

other PEs, but since this change only affects a single PE in the entire design, the impact on total area is very small. The delay through PE$_0$ is virtually the same as that of the other PEs.

IV. Implementation Results and Measurements

This ASIC, implemented in the 180nm process, was built with $N = 1024$, $P = 64$, and a code rate of $1/2$. Each LLR value is stored using $Q = 5$ bits in sign-and-magnitude form. Synopsys Design Compiler and Cadence SoC Encounter were used for synthesis and layout, respectively. The critical path of the design originates from the controller, goes through the LLR memory, a PE and back to the LLR and partial sum memories. The layout is simple, with power rails on the periphery and four additional pairs of vertical stripes to distribute the core power without much drop. Functional testing of the chip was done by connecting the chip to an FPGA board which supplied the clock and the stimuli for each cycle, and recorded the responses. The maximum achieved frequency is 150 MHz in this setup. The measurement results are shown in Table I. A micrograph of the chip is shown in Fig. 5, in which the PEs, controller and partial sum memory are marked; while the LLR memory accounts for the remaining, unmarked area.

V. Conclusion

Polar codes are an interesting candidate for error correction in future telecommunication systems. The silicon complexity of a successive cancellation (SC) decoder chip for a code block size of 1024 bits is 183k gates, resulting in 1.72 mm^2 in a 180 nm technology. The corresponding throughput is 49 Mbps. The key concepts for the hardware efficient implementation of a SC decoder for polar codes are the semi-parallel architecture proposed in [7], which makes the silicon complexity feasible, and a concurrent decoding technique to increase the throughput by 25% with negligible additional complexity. On top of that, an optimized implementation of the processing elements (PE) is important to achieve a high operating frequency, as the critical paths pass through the PEs.

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, July 2009.

[2] E. Abbe and A. Barron, “Polar coding schemes for the AWGN channel,” in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 31 2011-aug. 5 2011, pp. 194 –198.

[3] E. Sasoglu, E. Telatar, and E. Arikan, “Polarization for arbitrary discrete memoryless channels,” in *Information Theory Workshop, 2009. ITW 2009. IEEE*, Oct. 2009, pp. 144–148.

[4] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” in *Proc. IEEE Int Acoustics, Speech and Signal Processing (ICASSP) Conf*, 2011, pp. 1665–1668.

[5] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J Gross, “Hardware implementation of successive cancellation decoders for polar codes,” *Journal of Signal Processing Systems*, vol. 69, no. 3, pp. 305–315, December 2012.

[6] Chuan Zhang, Bo Yuan, and Keshab K. Parhi, “Reduced-latency SC polar decoder architectures,” in *Communications, 2012. ICC ’12. IEEE International Conference on*, 2012, pp. 3520–3524.

[7] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” *To appear in IEEE Transactions on Signal Processing*.

[8] I. Tal and A. Vardy, “How to construct polar codes,” *submitted to IEEE Trans. Inf. Theory, available online as* `arXiv:1105.6164v2`, 2011.