Energy-adaptive Polar Coding: Trading Off Reliability and Decoding Energy with Adaptive Polar Coding Circuits

Haewon Jeong, Christopher G. Blake, and Pulkit Grover
haewon@cmu.edu, christopher.blake@mail.utoronto.ca, pulkit@cmu.edu

***Abstract*—To be considered for the 2017 IEEE Jack Keil Wolf ISIT Student Paper Award. Recent works have found that using a long and complicated error correcting code (ECC) designed for the worst-case error probability requirement wastes excessive total system energy (transmit + circuit energy) when the error probability requirement is much higher than the worst case. We propose a novel adaptive polar coding strategy that adjusts the decoder circuit to consume minimal decoding circuit energy at each given target error requirement. By combining Thompson's VLSI theory and scaling analysis of polar codes, we provide upper bounds on energy, area, and time complexity of polar decoding circuits in terms of target block error probability. Comparison of the upper bounds for non-adaptive polar coding and our proposed adaptive polar coding showed that the adaptive coding strategy has a scaling-sense gain in decoding energy with little circuit area overhead when there is a large gap between the worst-case and typical target error rate requirements.**

I. Introduction

Recent work has shown that codes that approach Shannon capacity are necessarily sub-optimal when the minimization of the total communication energy (*i.e.,* encoding/decoding circuit energy + transmit energy) is taken into account instead of only transmit energy [13]. The lower bounds proved in [5], [12], [13] state that total energy cost must grow to infinity as target error probability P_e approaches 0. This suggests that to be energy-optimal, we should use different coding schemes for different target error probabilities [10].

However, designing different codes for each different target P_e requires high circuit area cost, and also high design cost. To save energy consumed when using ECC with minimal circuit area and design overhead, *"energy-adaptive coding"* was proposed in our previous work [18]. The idea of energy-adaptive coding, which builds on rate-adaptive codes, is to use one code which can adjust its decoding complexity so that it can adapt energy consumption as target error rate changes. It was shown using simulations that an adaptive code design based on low-density-parity-check codes saves up to 2x energy at a lower precision requirement.

In this work, we extend the simulation-based analysis in [18] to theoretical analysis to examine whether adaptive coding strategies can save decoding energy *in scaling sense* over non-adaptive coding strategies. Our goal is to give asymptotic scaling of decoding energy required using adaptive coding strategies in terms of block error probabilities. For this goal, we propose another energy-adaptive coding scheme based on polar codes [3] since they have a hierarchical structure that can be conveniently utilized for adaptive coding strategies[1]. Polar codes are suitable for theoretical analysis of adaptive coding strategy as they not only achieve Shannon capacity but also their finite-length scaling rules are rigorously studied [14], [16], [24].

For our analysis, we consider *a varying-target-error-rate scenario* in which the lowest P_e requirement (worst-case) approaches 0 while more frequent P_e (typical) requirement remains constant. Under this scenario, we derive upper bounds on energy, area, and time complexity of the proposed energy-adaptive polar coding strategy and compare them against non-adaptive polar coding. The comparison shows that integrative implementation of adaptive polar coding saves decoding energy approximately by $\log \frac{1}{P_{e,worst}}$ factor at typical target P_e, while having no scaling-sense circuit area overhead.

The proposed adaptive coding strategy can be useful for devices that run applications with varying target error rate requirements such as cell phones that receive packets for video/audio streaming (high P_e tolerance) and also packets for program instructions (very low P_e tolerance). It can be beneficial for wireless sensor networks as well since wireless sensor nodes often have limited or unreliable energy sources and hence might have to adapt their coding schemes based on their energy level by sacrificing data reliability [17], [27].

Using ECC in an adaptive fashion is not a new idea. Rate-adaptive (rate-compatible) codes that use a single encoder/decoder pair for codes with different rates were proposed to maximize communication rate when channel is time varying [1], [8], [15], [23], [26]. Adaptive successive cancellation list decoding for polar codes was studied to improve the minimum distance of polar codes with lower decoding complexity [21]. Rateless polar codes introduced in [22] were designed to adapt to unknown channels by incrementally freezing more bits. Our approach is similar to previous adaptive coding ideas in the sense that we reduce circuit area by building an ECC solution that uses one encoder/decoder pair for many different situations. However, our goal of adapting a code is not to maximize the data transmission rate, but to minimize

[1]We thank Rüdiger Urbanke for suggesting polar codes for an energy-adaptive coding strategy at Allerton, 2015.

decoding energy for a fixed channel and varying target error requirements. Also, in contrast to previous works that deal with a varying channel, we consider a scenario where a channel is fixed but target error rate at the receiver varies.

Area/energy/time analyses given here are all upper bounds obtained from a "mesh-network" based decoder construction [6]. We chose mesh network structure since it was shown that the mesh network implementations achieve the energy lower bound for polar decoders within poly-logarithmic factors [6]. The regular structure of mesh networks is easy to analyze and yet it is an efficient layout for planar circuits. Ideally, we should compare an energy *lower* bound of non-adaptive strategy with our upper bound of adaptive strategy (rather than comparing two upper bounds) in order to show fundamental energy savings with the adaptive strategy. However, assuming that the relationship between P_e and N given in Theorem 1 is tight, our comparison *does* conclusively show that adaptive strategy outperforms the non-adaptive strategy in scaling sense. This is because the non-adaptive strategy's bound is tight within poly-logarithmic factors [6]. Finally, we want to note that our analysis is limited to decoding energy only. Including encoding and transmit energy would be our future work.

Our key contributions are summarized as follows:

- Showing that the proposed adaptive coding strategy has scaling-sense gain in decoding energy with little circuit area overhead in comparison to non-adaptive coding when there is a wide range of target error rate requirements.
- Proposing a way of using polar decoders flexibly depending on how much communication reliability we can sacrifice in exchange for energy savings.

In Section II, we introduce a few definitions and channel and circuit model that we use in the paper. In Section III, we provide our construction of energy-adaptive polar coding strategy. Then we state our main result on area, time, and energy upper bounds of adaptive polar codes in Section IV. The advantage of using adaptive polar codes in terms of time and energy is quantified using this main result. Proof overview of the main result is given in Section V.

II. Model, definitions, preliminaries

A. Channel Model

A transmitter encodes k bits of message $\boldsymbol{u}$ into an n-bit codeword $\boldsymbol{x}$ and sends $\boldsymbol{x}$ over a channel W ($n > k$). We will denote this coding scheme as (n, k) code and the rate of the coding scheme is $R = \frac{k}{n}$ (bits/channel use). Throughout this work, we will assume that the channel W is a binary symmetric channel (BSC) with flip probability p_{ch}, denoted by BSC(p_{ch}). This channel model is equivalent to hard-decision decoding under additive white Gaussian noise (AWGN) channel with binary phase shift keying (BPSK). Also, we denote the target block error probability by P_e, and we assume that there are different target error probability requirements for different applications at the receiver end. We denote a scenario where the worst-case target block error probability is $P_{e,worst}$ and typical target block error probability is $P_{e,typ}$ as *a* $(P_{e,typ}, P_{e,worst})$ *scenario*. By $P_{e,typ}$, we mean informally a more frequently required error probability at the receiver. For instance, if a device requires error rate 10^{-3} for 70% of the time, and 10^{-6}, 10^{-9}, 10^{-12} for 10% of the time each, $P_{e,typ} = 10^{-3}$ and $P_{e,worst} = 10^{-12}$.

B. Coding and Decoding Schemes

Definition 1 ($(n, R, \mathcal{E}, \mathcal{D})$ Coding Scheme)**.** *An $(n, R, \mathcal{E}, \mathcal{D})$ coding scheme consists of a code of rate R, an encoding function $\mathcal{E} : 2^{nR} \to 2^n$ and a decoding function $\mathcal{D} : \mathcal{Y}^n \to 2^{nR}$, where $\mathcal{Y}$ is the set of channel output alphabet, n is the* block length *and R is* rate *of the code.*

Definition 2 ((N, R, W) Polar Coding)**.** *An (N, R, W) polar code is a polar code with length N and rate R which is designed for a channel W.*

We will not discuss the design of polar codes in this paper for which we refer the readers to [3], [25]. We will use successive cancellation (SC) decoding for polar codes as defined by Arıkan in [3]. The following theorem showed the gap-to-capacity performance of SC decoders [14, Theorem 3].

Theorem 1. *There is an absolute constant $\mu < \infty$ such that the following holds. Let W be a binary-input memoryless output-symmetric channel with capacity $I(W)$. Then there exists $a_W < \infty$ such that for all $\epsilon > 0$ and all powers of two $N \geq a_W(1/\epsilon)^\mu$, a polar code with block length N and rate $I(W) - \epsilon$ achieves a block error probability of at most $2^{-N^{0.49}}$ for communication over W by SC decoding algorithm.*

C. Circuit Model

Our circuit model follows the model in [13] that adapts Thompson's VLSI model [7]. We assume that a circuit Ckt is made of nodes and wires. Nodes can be input, output or computational nodes. Ckt carries out a computation Comp by communicating bits over the wires. We use a metric called *bit-meters* as an estimate of circuit energy. It was introduced in [13] as a good approximation of energy expended for communicating bits over a medium, especially metal wires. Also, experimental work has shown that energy spent on wires is a major energy cost on modern VLSI circuits [10].

Definition 3 (bit-meters of a wire and of a circuit)**.** *Bit-meters cost of a wire for a computation Comp is*

$$\text{bit-meters}_{\text{Comp}}(\textit{wire}) = B \cdot d \tag{1}$$

where B is the number of bits moved through wire during the computation Comp and d is the Euclidean distance of wire. Bit-meters cost of a circuit for a computation Comp is

$$\text{bit-meters}_{\text{Comp}}(Ckt) = \sum_{\textit{wire in Ckt}} \text{bit-meters}_{\text{Comp}}(\textit{wire}). \tag{2}$$

Circuit energy for Ckt that implements Comp can be estimated using the following equation.

$$E(Ckt) = \mu \cdot \text{bit-meters}_{\text{Comp}}(Ckt) \tag{3}$$

where μ is the coefficient of information friction.

For estimating circuit area upper bounds, we follow the assumptions given in [7, Assumption U1-U2]. We assume that a node has at most $O(1)$ input wires and $O(1)$ output wires, and each of its wires is $O(1)$ units long. Total circuit area is upper bounded using the smallest bounding rectangle that covers all nodes and wires.

D. Mesh Network Circuit Implementation

Definition 4 (Processing Cells and N-cell Mesh Network)**.** *A processing cell (in short, a cell) is a two-dimensional array of computational nodes that takes a certain set of inputs and produces a set of outputs. A cell can exchange messages with its adjacent cells. An N-cell mesh network is a $\sqrt{N}$-by-$\sqrt{N}$ square grid of processing cells in which cell i's $(i = 1, \cdots, N)$ are placed equal distance apart. Cells in a mesh network have interconnections with all its nearest neighboring cells.*

Definition 5 (M-cell sub-mesh)**.** *An M-cell sub-mesh of an N-cell mesh network is a rectangular grid of M cells ($M < N$) in the N-cell mesh.*

E. SC Decoding on Mesh Network

We now briefly discuss how SC decoding of an (N, R, W) polar code can be implemented using an N-cell mesh network. More detailed explanations are given in Appendix A. Cell i in the mesh will take the i-th channel output y_i, and produce the i-th decoded bit $\hat{u}_i$. To complete SC decoding of an N-length polar code, we have to compute $N \log N$ likelihood ratio (LR) values [3] and $N(\log N - 1)$ combiner bits (CBs), also known as partial sums [4], [9]. Each cell will be responsible for computing $\log N$ LRs and $(\log N - 1)$ CBs. The computation of LR/CB values is carried out in a recursive fashion. More precisely, a new LR value is computed from two previously computed LR values (and the same is done for CB computations). A cell will acquire two previously computed values by sending a request message on the mesh, and the message will be routed until it reaches the cell that contains the requested value. After getting replies with the requested values, a cell will execute simple arithmetic operations on them to produce the new LR/CB value.

F. Problem Statement

In this paper, we consider a problem of constructing an adaptive polar coding strategy for a $(P_{e,typ}, P_{e,worst})$ scenario, a fixed channel BSC(p_{ch}), and a fixed rate R that adapts its encoding and decoding functions depending on the given target block error probability P_e. We investigate required decoding circuit energy, decoder circuit area, and clock cycles to achieve typical block error probability $P_{e,typ}$ of an adaptive polar coding strategy designed for a $(P_{e,typ}, P_{e,worst})$ scenario.

III. Energy-adaptive Polar Coding

We first define a general L-level adaptive coding strategy.

Definition 6 (L-level Adaptive Coding Strategy)**.** *An L-level adaptive coding strategy consists of L different coding schemes, $\mathcal{C}_1 = (n_1, R_1, \mathcal{E}_1, \mathcal{D}_1)$, $\cdots$, $\mathcal{C}_L = (n_L, R_L, \mathcal{E}_L, \mathcal{D}_L)$, and a level selection rule $\mathcal{L} : E \to \{\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_L\}$ where E is a set of environment parameters e.g., target error probability P_e or channel SNR. We will call $\mathcal{C}_l$ as the l-th level code of (N, R, W, L) adaptive coding strategy.*

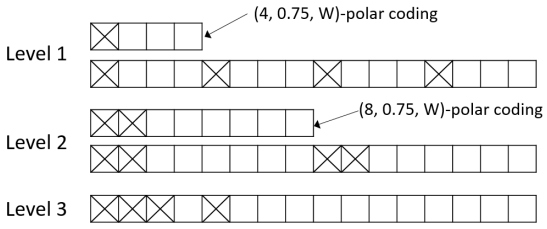

Fig. 1. An example of $(16, 0.75, W, 3)$ adaptive polar coding strategy where W is a BSC with flip probability $p_{ch} = 0.05$. The 1-st level coding scheme consists of four blocks of $(4, 0.75, W)$ polar codes, and the second level consists of two blocks of $(8, 0.75, W)$ polar codes. Finally, the 3-rd level scheme is equal to the $(16, 0.75, W)$ polar code.

Example 1 (Example of 2-level Adaptive Coding Scheme)**.** *Let the channel be binary erasure channel (BEC) with erasure probability p_ϵ. Let $\mathcal{C}_1 = (6, 1/2, \mathcal{E}_1, \mathcal{D}_1)$ where $\mathcal{E}_1$ is encoding a repetition code with rate 1/2, i.e.,*

$$\mathcal{E}_1 : (u_1, u_2, u_3) \to (u_1, u_1, u_2, u_2, u_3, u_3).$$

$\mathcal{D}_1 : \{0, 1, E\}^6 \to \{0, 1\}^3$ decodes $(\hat{u}_1, \hat{u}_2, \hat{u}_3)$ by any of bits that are not erased and if both of the repeated bits are erased guess the bit to 0. Let $\mathcal{C}_2 = (6, 1/3, \mathcal{E}_2, \mathcal{D}_2)$ where $\mathcal{E}_2$ and $\mathcal{D}_2$ an encoding and a decoding function of a repetition code with 3 repetitions. Let the level selection rule as follows:

$$\mathcal{L}(p_\epsilon) = \begin{cases} \mathcal{C}_1, \text{ if } p_\epsilon > 0.3 \\ \mathcal{C}_2, \text{ otherwise} \end{cases}$$

This is a 2-level adaptive code that switches between two different rates, $1/2$ and $1/3$, depending on the channel condition.

Now let us define ***L-level energy-adaptive polar coding***. We call this construction *energy-adaptive polar coding* because block length and rate are fixed and the only aspect changing over different levels is computational energy for encoding and decoding. The underlying idea in the design of adaptive polar coding is to divide a long polar code into smaller sub-blocks and encode/decode them in parallel as separate smaller polar codes. For an L-level energy-adaptive polar code, $\mathcal{C}_L$ would be a polar code with length N. Then the $(L-1)$-th level coding scheme divides the code into two $N/2$ sub-blocks and uses the blocks as $N/2$-length polar codes. At the $(L-2)$-th level the sub-blocks will be divided into two again. The construction is formally defined in the following definition.

Construction 1 (L-level Energy-Adaptive Polar Coding)**.** *An (N, R, W, L) energy-adaptive polar coding is an L-level adaptive coding strategy with $\mathcal{C}_1 = (N, R, \mathcal{E}_1, \mathcal{D}_1)$, $\cdots$, $\mathcal{C}_L = (N, R, \mathcal{E}_L, \mathcal{D}_L)$. The l-th level code is divided into sub-*

blocks of length $n_l = N/2^{L-l}$ and n_1 satisfies the following condition:

$$n_1 \geq a_W(1/(I(W) - R))^{\mu} \tag{4}$$

where a_W and μ are constants from Theorem 1. We denote $\mathcal{E}_l'$ and $\mathcal{D}_l'$ as the encoding and decoding functions for an $(n_l, R,\ W)$ polar code. Then, $\mathcal{E}_l$ and $\mathcal{D}_l$ are written as follows:

$$\mathcal{E}_l(u_1^{NR}) = \mathcal{E}_l'(u_1^{n_l R}) \circ \mathcal{E}_l'(u_{n_l R+1}^{2n_l R}) \circ \cdots \circ \mathcal{E}_l'(u_{NR-n_l R+1}^{NR})$$

$$\mathcal{D}_l(y_1^N) = \mathcal{D}_l'(y_1^{n_l}) \circ \mathcal{D}_l'(y_{n_l+1}^{2n_l}) \circ \cdots \circ \mathcal{D}_l'(y_{N-n_l+1}^{N})$$

where $\circ$ denotes the vector concatenation and v_i^j denotes $(v_i, \cdots, v_j)$ of a vector $\mathbf{v}$. Level selection rule $\mathcal{L} : P_e \to \{\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_L\}$ is given as

$$\mathcal{L}(P_e) = C_{l^*} \text{ where } l^* = \min\{l \,|\, 2^{L-l-n_l^{0.49}} \leq P_e\}.$$

where P_e is target block error probability.

This level selection rule $\mathcal{L}(P_e)$ ensures that block error probability of the chosen level is smaller than P_e. However, this is a conservative selection rule based on the theoretical guarantee of SC decoding in Theorem 1 and the union bound. For a better understanding, we provide an example of 3-level energy-adaptive polar coding strategy in Fig. 1.

We discuss two different ways to implement adaptive coding strategies. We call an implementation of an L-level adaptive coding as *"naive"* if it consists of L separate sub-circuits dedicated to each different level. In other words, Subckt$_l$ implements an encoding and a decoding function for level l, $\mathcal{E}_l$ and $\mathcal{D}_l$. When the l-th level coding scheme $\mathcal{C}_l$ is used, only Subckt$_l$ will be activated. Any non-naive implementation is called *"integrative"* implementation. Unlike the naive implementation, an integrative implementation can reuse the same circuit for encoding and decoding of different levels. This can reduce the total area occupied by the circuit in comparison to naive implementation. At the same time, because integrative implementations do not have specialized sub-circuits for each adaptive level, the computation can suffer from more delay or require more energy. We will denote naive implementation with the subscript $_{adap-naive}$, integrative implementation with $_{adap-int}$, and non-adaptive implementation with $_{non-adapt}$.

IV. Main Results

In our main theorem, we compare decoding circuit energy, circuit area, and computation time of non-adaptive polar coding and adaptive polar coding implementations under a $(P_{e,typ}, P_{e,worst})$ scenario.

Theorem 2. *Under a $(P_{e,typ}, P_{e,worst})$ scenario and δ being 0.0205, decoding circuit energy required to achieve $P_{e,typ}$ is upper bounded by*

$$E_{non-adapt} = O\left(\log^{3+3\delta} \frac{1}{P_{e,worst}} \left(\log\log \frac{1}{P_{e,worst}} \right)^4 \right)$$

$$E_{adap-naive} = O\Bigg(\log^{2+2\delta} \frac{1}{P_{e,worst}} \Bigg(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \Bigg)^{1+\delta} \left(\log \left(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \right) \right)^4 \Bigg)$$

$$E_{adap-int} = O\Bigg(\log^{2+2\delta} \frac{1}{P_{e,worst}} \left(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \right)^{1+\delta} \left(\log\log \frac{1}{P_{e,worst}} \right)^2 \left(\log \left(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \right) \right)^2 \Bigg).$$

Circuit area of the decoders is upper bounded by

$$A_{non-adapt} = O\left(\log^{2+2\delta} \frac{1}{P_{e,worst}} \left(\log\log \frac{1}{P_{e,worst}} \right)^2 \right)$$

$$A_{adap-naive} = O\left(\log^{2+2\delta} \frac{1}{P_{e,worst}} \left(\log\log \frac{1}{P_{e,worst}} \right)^3 \right),$$

and $A_{adap-int}$ has same scaling-sense upper bound as $A_{non-adapt}$.

Number of clock cycles required to achieve $P_{e,typ}$ is upper bounded by

$$T_{non-adapt} = O\left(\log^{3+3\delta} \frac{1}{P_{e,worst}} \left(\log\log \frac{1}{P_{e,worst}} \right)^2 \right)$$

$$T_{adap-naive} = O\Bigg(\left(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \right)^{3+3\delta} \left(\log \left(\log\log \frac{1}{P_{e,worst}} + \log \frac{1}{P_{e,typ}} \right)^2 \right) \Bigg),$$

and $T_{adap-int}$ has same scaling-sense upper bound as $T_{adap-naive}$.

The theorem shows the following relations:

- $E_{non-adapt} > E_{adap-int} > E_{adap-naive}$.
- $A_{adap-naive} > A_{non-adapt} \approx A_{adap-int}$.
- $T_{non-adap} > T_{adap-naive} \approx T_{adap-int}$.

Naive-adaptive implementation, which simply instantiates a different circuit for each level, is most energy-efficient at the cost of the larger overall area. Integrative implementation of adaptive coding saves energy by approximately $\log \frac{1}{P_{e,worst}}$ factor compared to $E_{non-adapt}$, but still requires more energy than $E_{adap-naive}$. However, it does not have any additional circuit area cost in order sense. Furthermore, computation time for achieving $P_{e,typ}$ with integrative-adaptive implementation is smaller than non-adaptive coding.

V. Proof Overview

The main idea of this proof is combining the results of Guruswami et al. [14] on the scaling exponent of polar codes and the results of Blake et al. [6] which analyzed the energy complexity of SC polar decoders on a mesh network. Lemma 3-5 present upper bounds on time, area, and energy complexity of energy-adaptive polar decoding by giving a concrete SC decoder construction based on a mesh network. Finally, the last lemma connects the selection rule of adaptive level with a $(P_{e,typ}, P_{e,worst})$ scenario.

Lemma 3. *Circuit area of a non-adaptive polar decoder of block length N is upper bounded by*

$$A_{non-adapt} = O(N \log^2 N), \tag{5}$$

and circuit area for an (N, R, W, L) adaptive polar coding

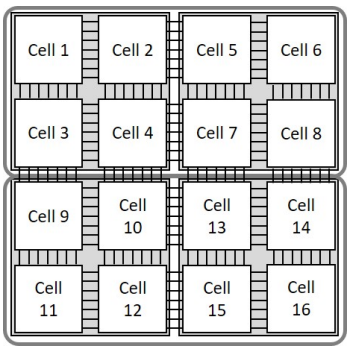

Fig. 2. An example mesh-network decoder implementation for a 3-level energy-adaptive polar code with $N = 16$. At level 1, cells only have to communicate within 4-cell sub-meshes which are marked with the gray-shaded rectangles. At level 2, cells need to communicate within 8-cell sub-meshes which are the rectangles with thick gray boundaries. Finally at level 3, cells have to communicate globally all over the mesh.

decoder is upper bounded by

$$A_{adap-naive} = O(LN \log^2 N) \tag{6}$$

$$A_{adap-int} = O(N \log^2 N). \tag{7}$$

Proof Overview. The upper bound on $A_{non-adapt}$ was briefly discussed in [6]. We provide a more detailed derivation of the upper bound in this work. Due to the similarity between FFT and polar decoding, we can employ an argument similar to that used to obtain an area upper bound on the mesh network implementation of FFT algorithm in Thompsons's original work [7]. The crux of proving the upper bound is to show that each cell in the mesh network can be implemented in an $O(\log N)$-by-$O(\log N)$ rectangle. Then the total area of N-cell mesh becomes $O(N \cdot \log^2 N)$. A rigorous analysis given in Appendix A.

Similarly, we can show that a decoder circuit for the l-th level code of (N, R, W, L) adaptive polar coding can be implemented on an N-cell mesh network where each cell takes $O(\log^2 n_l)$ area. Hence $A_{adap-naive} = O(\sum_{l=1}^{L} N \log^2 n_l) = O(L \cdot N \log^2 N)$.

To prove the upper bound on $A_{adap-int}$, we now quantify area required by additional circuit elements for converting a non-adaptive N-cell mesh network decoder into an adaptive decoder. A cell needs an additional input for level-selection bits, additional memory for frozen bit information at each level, and additional instructions on when to start/end the cell's computation at each level. Level-selection bits are $\log L$ bits long, which is upper bounded by $O(\log \log N)$ since $L \leq \log N$. Frozen bit information requires L-bit memory in each cell which is upper bounded by $O(\log N)$. Finally, additional instructions require $C \cdot L$ area which is $O(\log N)$. In all, the additional circuit elements to turn the non-adaptive decoding circuit into the adaptive circuit only requires $O(\log N)$ area in each cell. Hence a cell for the adaptive polar coding circuit can still be implemented in $O(\log N)$-by-$O(\log N)$ rectangle. The area upper bound for $A_{adap-int}$ is thus $O(N \log^2 N)$. □

Lemma 4. *Number of clock cycles for SC decoding an N-length non-adaptive polar code is upper bounded by*

$$T_{non-adapt} = O(N^{1.5} \log^2 N). \tag{8}$$

Number of clock cycles for decoding the l-th level coding scheme of an (N, R, W, L) adaptive polar coding is upper bounded by

$$T^{(l)}_{adap-naive} = O(n_l^{1.5} \log^2 n_l) \tag{9}$$

$$T^{(l)}_{adap-int} = O(n_l^{1.5} \log^2 n_l). \tag{10}$$

Proof. Refer to Appendix B. □

Lemma 5. *Bit-meters energy estimate of SC decoding for a N-length non-adaptive polar code is upper bounded by*

$$E_{non-adapt} = O(N^{1.5} \log^4 N) \tag{11}$$

and bit-meters energy estimate of decoding the l-th level coding scheme of an (N, R, W, L) polar coding is upper bounded by

$$E^{(l)}_{adap-naive} = O(N n_l^{0.5} \log^4 n_l) \tag{12}$$

$$E^{(l)}_{adap-int} = O(N n_l^{0.5} \log^2 N \log^2 n_l). \tag{13}$$

Proof. During one clock cycle, $O(\log N)$ bits need to move at most $O(\log N)$ distance within one cell. Since only one cell is active at a time in non-adaptive polar decoding, bit-meters during one clock cycle is upper bounded by $O(\log^2 N)$. Combining this with Lemma 4 gives the upper bound (11).

Upper bounds on $E^{(l)}_{adap-naive}$ and $E^{(l)}_{adap-int}$ can be proved similarly. The only differences are that N/n_l cells are active during one clock cycle in adaptive circuits and that bit-meters during one clock cycle are bounded by $O(\log^2 n_l)$ in naive implementation and by $O(\log^2 N)$ in integrative implementation. □

Lemma 6. *Let us consider using (N, R, W, L) adaptive polar coding under a $(P_{e,worst}, P_{e,typ})$ scenario. Then the following holds with $\delta = 0.0205$.*

1) Any N that satisfies

$$N \geq \left(\log \frac{1}{P_{e,worst}}\right)^{2+2\delta},$$

meets $P_{e,worst}$ condition.

2) Any n_l that satisfies

$$n_l \geq \left(\log N + \log \frac{1}{P_{e,worst}}\right)^{2+2\delta}$$

meets $P_{e,typ}$ condition.

Proof. Refer to Appendix C. □

We want to mention that δ here is 0.0205 since we directly apply Theorem 1. We can make δ to be arbitrarily small, but then the condition on n_1 given in (4) have to be much larger.

Theorem 2 can be obtained by substituting N and n_l in Lemma 3-5 with the result given in Lemma 6.

VI. Future Work

Extending our work on SC decoding to other decoding algorithms, e.g., belief-propagation decoding [2] or linear-program decoding [11], is an interesting direction of future study. A particular challenge we may find in their analysis is that their finite-length analysis may be difficult. However, in practice, their performance may be comparable, or even better, than the theoretically analyzable construction herein.

Appendix A
Proof of Lemma 3

In this proof, we will focus on how a processing cell can be implemented in $O(\log N)$-by-$O(\log N)$ area.

Before delving into analyzing the structure of a cell, we first briefly recap SC decoding process. Let us first introduce some notations. $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$ denotes the likelihood ratio of the i-th bit of N-length polar code given the channel output y_1^N and previously decoded bits u_1^{i-1}:

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) = \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|\hat{u}_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|\hat{u}_i = 1)}$$

where W denotes channel transition probability. We will use $L_N^{(i)}$ as an abbreviation of $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$. Also, we will use $v_{1,e}^j$ and $v_{1,o}^j$ to denote even-indexed terms and odd-indexed terms in the vector v_1^j, respectively. $v^{(i)}$ denotes the i-th term in the vector $\boldsymbol{v}$. Now The following $L_N^{(i)}$'s can be computed recursively using the following relations [3]:

$$\begin{aligned} &L_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) \\ &= \frac{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})}{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})} \end{aligned} \tag{14}$$

$$\begin{aligned} L_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) = \left[L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})\right]^{1-2\hat{u}_{2i-1}} \\ \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}), \end{aligned} \tag{15}$$

with the base case

$$L_1^{(1)}(y_i) = \frac{W(y_i|\hat{u}_i = 0)}{W(y_i|\hat{u}_i = 1)}. \tag{16}$$

We will divide the computation into two parts, likelihood ratio (LR) computation and combiner bit (CB) computation. Combiner bits were referred to as *partial sums* in the previous literature [4], [9], [19], [20].

It is hard to see CB computation in (14-15). We will generalize the notations to make the CB computation clearly seen in recursive equations. We will use $\boldsymbol{y}$ to denote a vector of length n which can be any sub-vector of the channel output y_1^N and n can be any number among $N, N/2, \cdots, 1$. Most importantly, we introduce a new vector, $\boldsymbol{b}_n(\boldsymbol{y})$, which we call *a combiner bit vector for the vector* $\boldsymbol{y}$, where $\boldsymbol{y}$ is a length-n sub-vector of the channel output. The combiner bit vector $\boldsymbol{b}_n(\boldsymbol{y})$ can be thought of as the second term in the $L_N^{(i)}$ computation and it also has length n. Using these notations, we rewrite (15) as follows:

$$\begin{aligned} &L_n^{(2i)}(\boldsymbol{y}, (\boldsymbol{b}_n(\boldsymbol{y}))_1^{2i-1}) \\ &= \left[L_{n/2}^{(i)}(\boldsymbol{y}_1^{n/2}, (\boldsymbol{b}_n(\boldsymbol{y}))_{1,o}^{2i-2} \oplus (\boldsymbol{b}_n(\boldsymbol{y}))_{1,e}^{2i-2}\right]^{1-2\boldsymbol{b}_n^{(2i-1)}(\boldsymbol{y})} \\ &\cdot L_{n/2}^{(i)}(\boldsymbol{y}_{n/2+1}^n, (\boldsymbol{b}_n(\boldsymbol{y}))_{1,e}^{2i-2}) \\ &= \left[L_{n/2}^{(i)}(\boldsymbol{y}_1^{n/2}, (\boldsymbol{b}_{n/2}(\boldsymbol{y}_1^{n/2}))_1^{i-1})\right]^{1-2\boldsymbol{b}_n^{(2i-1)}(\boldsymbol{y})} \\ &\cdot L_{n/2}^{(i)}(\boldsymbol{y}_{n/2+1}^n, (\boldsymbol{b}_{n/2}(\boldsymbol{y}_{n/2+1}^n))_1^{i-1}) \end{aligned} \tag{17}$$

From (17), we can derive the recursive relation between combiner bit vectors:

$$\begin{aligned} b_{n/2}^{(i)}(\boldsymbol{y}_1^{n/2}) &= b_n^{(2i-1)}(\boldsymbol{y}) \oplus b_n^{(2i)}(\boldsymbol{y}) \\ b_{n/2}^{(i)}(\boldsymbol{y}_{n/2+1}^n) &= b_n^{(2i)}(\boldsymbol{y}). \end{aligned} \tag{18}$$

Combiner bits must be computed separately through this recursive relation in order to carry out LR computation as they are necessary to combine two $L_{n/2}^{(i)}$'s in (17). The recursive equations for combiner bits can be computed in the opposite direction of likelihood ratios, $L_n^{(i)}(\boldsymbol{y}, \boldsymbol{b})$'s. Computing $L_n^{(i)}(\boldsymbol{y}, \boldsymbol{b})$'s starts from the base case $L_1^{(i)}(\boldsymbol{y})$'s then $L_2^{(i)}$'s and so on. On the other hand, $b_n^{(i)}(\boldsymbol{y})$'s are computed from the base case $b_N(i)(y_1^N) = \hat{u}_i$ then $b_{N/2}^{(i)}$'s and so on. As we can describe the recursive formula for likelihood ratios on a butterfly-like network, we can do the same for the combiner bit formula (see Fig. 3)

We now describe the job of each cell on a mesh network for SC decoding process. The i-th cell (Cell i) takes one input, y_i, and outputs a decoded bit, $\hat{u}_i$. To decode the i-th bit, Cell i has to compute $L_N^{(i)}$. During the recursive computation process to compute $L_N^{(i)}$, it computes $\log N$ LR values and $(\log N - 1)$ CB values that are on the i-th row of the butterfly-like network graphs. By this choice, one of two values required at each recursion stage is obtained in the same cell and hence a cell has to communicate with only one other cell which greatly reduces the communication cost.

In our implementation, we adopt *lazy evaluation*, which means we start the computation from the final values of the recursion and intermediate values are computed only when they are requested (call-by-need). It was explained as *left-to-right* implementation in [3]. Also, we assume that LR values are M-bit floating numbers.

As briefly explained in Section II-E, a mesh network executes SC decoding by exchanging messages. The format of messages is explained in Table I. After each cell received the corresponding channel output, the decoding process starts by the request message for `(LR, 1, log N)`. Then a cell searches its instruction set to find required values to compute `(LR, 1, log N)`. It then sends out request messages to retrieve those values. After it receives the value return messages for all required values, it computes `(LR, 1, log N)`, then decode the first bit $\hat{u}_1$. It ends its role by sending out the request message for the next cell, `(LR, 2, log N)`. When

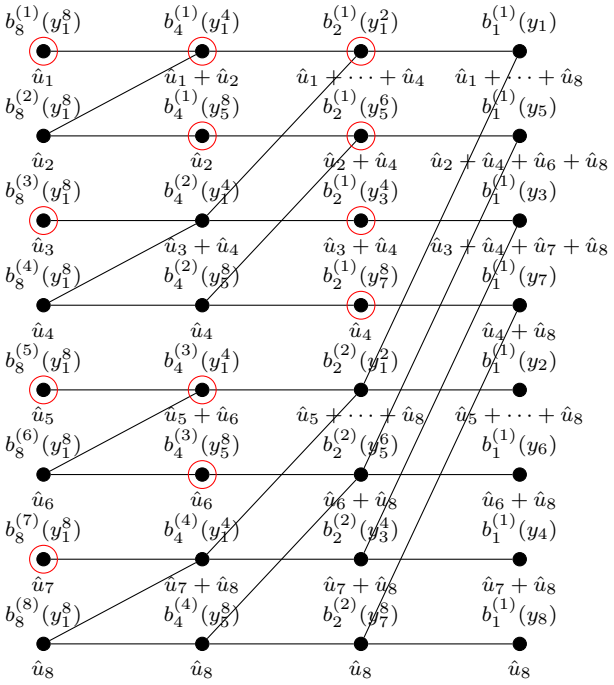

Fig. 3. This is the graph representation of recursive calculation of combiner bits when code length is 8. Each node represents one value of combiner bits and each column is one stage of recursion. The labels above the nodes are our notation of combiner bits and the labels below the nodes are actual values of the combiner bit. The leftmost nodes are the values in the first recursion stage which are simply decoded bits $\hat{u}_1$ to $\hat{u}_8$, and all the other nodes are XOR sums of the bits connected from their left edges. Combiner bits that are required during likelihood ratio computation are marked with a red circle. The other bits are only intermediate values to compute red-marked bits. We can see that the bottom half nodes at the third recursion stage and all the nodes in the last recursion stage do not have to be computed since they are not red-marked nor used to compute any red-marked nodes.

TABLE I
THE FORMAT OF MESSAGES

Request Message Format
`[REQ, input sel, dest addr, src addr]`
Value Return Message Format
`[VAL, input sel, dest addr, value]`
Destination/Source Address Format
`[LR/CB, cell addr, reg addr]`

Remark: Both types start with one-bit information `REQ/VAL` denoting whether the message is a request or a value return followed by three-bit information `input sel` that indicates which input the message is for among α_1, α_2, γ, α_3, and α_4. `dest addr` that follows is $\log N$-bit destination address. The only difference between request and value return messages is the last element in a message. A request message has a $\log N$-bit source address, `src addr`, since the destination cell has to reply back to the source cell.A value return message has `value` as the last component in the message. `value` is either a M-bit floating-point LR value or an one-bit CB value.

a cell receives a request message directed to it, it first searches its registers to check whether the requested value is already computed. If it is, it will reply a value return message to the source cell that sent the request message. Otherwise, it will

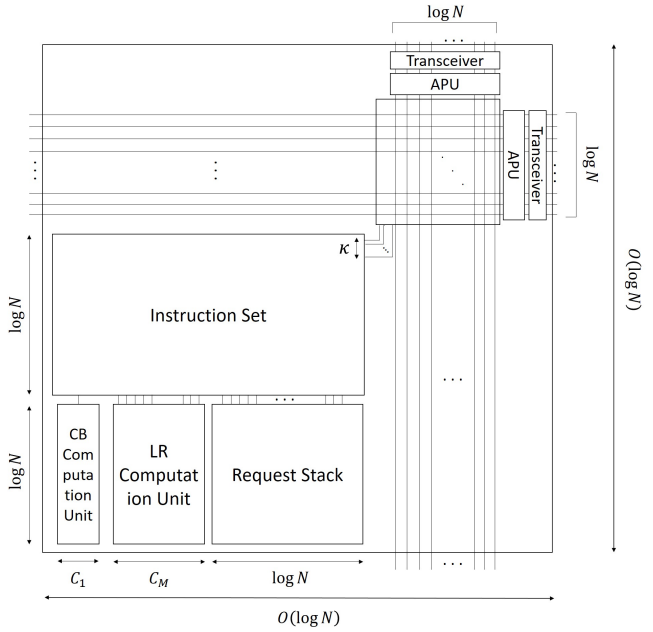

Fig. 4. A diagram of a cell showing the interconnections between the units in the cell.

start computing the request message. When the computation is complete, it saves the computed value to the corresponding register and sends the value return message to the source cell. A tricky part of this implementation is that a cell always has to send a request message to itself while computing a value. Hence it has to process a new request message, before completing the ongoing computation. We implement a request stack to track the request messages it has received so that it can come back to the original process before receiving a new request message.

A cell consists of 6 different units:

- Addressing processing unit (APU)
  APU processes incoming/outgoing messages and decide which direction to route the message to.
- Transceiver
  A transceiver exchanges messages of length $O(\log N)$ bits with the adjacent nodes in up/down/left/right directions.
- LR computation unit
  An LR computation unit has an arithmetic logic and registers to store previously computed LR values at the cell. An LR arithmetic logic takes two M-bit inputs α_1 and α_2, and one-bit input γ (when i even) and outputs M-bit β_1:
  $$\beta_1 = \frac{\alpha_1 \cdot \alpha_2}{\alpha_1 + \alpha_2} \tag{19}$$
  when i is odd and it computes
  $$\beta_1 = \alpha_1^{(1-2\gamma)}\alpha_2 \tag{20}$$
  when i is even.
- CB computation unit

An CB computation unit also has an arithmetic logic and registers to store previously computed CB values at the cell. A CB arithmetic logic takes two one-bit inputs α_3 and α_4 and outputs β_2. It computes

$$\beta_2 = \alpha_3 \oplus \alpha_4 \tag{21}$$

when i is odd and it simply sets

$$\beta_2 = \alpha_3 \tag{22}$$

It also has registers to store previously computed CB values at the cell.

- Instruction set
  For the computation of an LR value, we need two or three inputs, and for the CB computation, we need one or two inputs. An instruction set has information on which values to request at each stage. It stores messages to send out to request a value.
- Request stack
  Finally, we need a request stack due to the lazy evaluation strategy. If a cell is sending a request to itself, it has to halt the ongoing computation and start the requested computation. In this case, it saves the previously running computation in the request stack.

A diagram of a cell with all its units and the interconnections between them is shown in Fig. 4.

Proof. Let us examine how much area each element requires. It was shown that transceiver and APU for routing on an N-cell mesh can be implemented in $O(\log N)$ area [7]. An LR computation unit requires $O(\log N \cdot M)$ area for registers. The LR arithmetic logic involves M-bit multiplication, M-bit addition, and M-bit division. Hence it requires C_M area where C_M is constant that depends only on M. Similarly, a CB computation unit requires $O(\log N)$ area for registers. The CB arithmetic logic only contains bit XORs, so it requires a constant area of C_1. An instruction set stores one to three request messages for LR/CB computations and there are $(2\log N - 1)$ LR/CB values to compute. Since each message is $O(\log N)$ bits long, instruction set will consume $O(\log^2 N)$ area. The format of messages is explained in Table I and control programs are given in Program 1- 3. Finally a request stack requires $O(\log N)$ space for each request and we need to store up to $\log N$ requests as recursion depth is $\log N$ (although request stack will not be full except for the first cell). Laying out these units as depicted in Fig. 4 gives the total area $O(\log^2 N)$. □

APPENDIX B
PROOF OF LEMMA 4

Proof. When a node computes an LR value, it follows three steps:

1) Receiving a request to compute the value
2) Sending requests for the inputs α_1, α_2 and γ, and waiting for the value return messages
3) Computing β_1 from $\alpha_1, \alpha_2, \gamma$ and store it on the LR register

Program 1 Control program for a REQ message

`[REQ, input sel, dest addr, src addr]` was received.
if `dest addr` $\neq$ `(LR, i, log N)` **then**
  Save `(input sel, src addr)` to the request stack register.
end if
Check whether `dest addr` register is empty.
if `dest addr` register is empty **then**
  Send a message for `(REQ, dest addr)`.
else
  Send `(VAL, input sel, src addr, val(dest addr))`.
  Remove `(input sel, src addr)` from the top of the request stack.
end if

Program 2 Control program for a CB VAL message

`[VAL, input sel, dest addr, value]` was received.
if `input sel` = α_3 **then**
  Save `value` to the α_3 register.
  Send a message for `(`α_3`, dest addr)`.
else if `input sel` = α_4 **then**
  Set α_4 = `value`.
  Execute the CB computation logic: $\beta_2 = \alpha_3 \oplus \alpha_4$.
  Save the computed β_2 value to `dest addr` register.
  Send `(VAL, input sel, src addr, val(dest addr))` and remove the top element from the request stack.
end if

At step 1, it takes $\Theta(\log N)$ time at APU. A routing algorithm for APU is given in Algorithm 4.

At step 2, finding a request message to send out takes $O(\log \log N)$ clock cycles and sending the message to transceiver takes $O(\log N)$ clock cycles (due to serial communication sending κ bits in one clock cycle). Waiting time for the value return message to come back is $O(\sqrt{N} \log N)$ clock cycles, because at each hop, it takes $O(\log N)$ clock cycles at the APU and there are at most $2(\sqrt{N} - 1)$ hops in the mesh ($4(\sqrt{N} - 1)$ hops round-trip). Computation time at the destination cell that received the request message is not counted here because that will be double counting.

At step 3, number of clock cycles required for computing β_1 is constant, t_M, that depends only on M.

Since step 2 dominates the time required, total time is upper bounded by $O(\sqrt{N} \log N)$. CB computation follows the same steps, but only simpler at step 3. Hence number of clock cycles for CB computation is also upper bounded by $O(\sqrt{N} \log N)$. □

Program 3 Control program for a LR VAL message

`[VAL, input sel, dest addr, value]` was received.
if `input sel` = α_1 **then**
  Save `value` to the α_1 register.
  Send a message for (α_1, `dest addr`).
else if `input sel` = α_2 **then**
  Save `value` to the α_2 register.
  if γ is needed **then**
    Send a message for (α_2, `dest addr`).
  else
    goto *final*.
  end if
else if `input sel` = γ **then**
  goto *final*.
end if
final:
Execute the LR computation logic.
Save the computed β_1 value to `dest addr` register.
if request stack is not empty **then**
  Send (VAL, α_1/α_2, `src addr, val(dest addr)`) and remove the top element from the request stack.
else
  Run the decoding logic with the computed β_1.
  Save the decoded bit to CB(i, $\log N$) register.
  Send a trigger to the next cell.
end if

Algorithm 4 Routing algorithm for an APU at Cell i

Routing Rule: Routing will first done vertically and then horizontally. Address of Cell i is represented with $\log N$ which is the binary representation of $(i-1)$. We denote the address of Cell $i = [a_1 a_2 \cdots a_{\log N}]$ and `dest addr` $= [b_1 b_2 \cdots b_{\log N}]$.

for $j = 1, \cdots, \log N$ **do**
  if $a_j \neq b_j$ **then**
    if j even **then**
      if $a_j > b_j$ **then**
        Route to the *top*
      else
        Route to the *bottom*
      end if
      if $a_j > b_j$ **then**
        route to the *left*
      else
        route to the *right*
      end if
    end if
    break
  end if
  if $j = \log N$ **then**
    Route a message to the instruction set in the same cell
  end if
end for

APPENDIX C
PROOF OF LEMMA 6

Proof. From Theorem 1 and the condition (4), for any $N \geq \left(\log \frac{1}{P_{e,worst}}\right)^{1/0.49}$, block error probability after SC decoding is less than $P_{e,worst}$.

Now let us prove the second part of the lemma.

$$P_{blk} \leq (\text{\# sub-blocks}) \cdot \Pr(\text{error in each sub-block}) \quad (23)$$
$$\leq (\text{\# sub-blocks}) \cdot 2^{-n_l^{0.49}} \quad (24)$$
$$= \frac{N}{n_l} 2^{-n_l^{0.49}} \quad (25)$$

(24) follows from union bound and (25) follows from Theorem 1. For any n_l that satisfies

$$\frac{N}{n_l} 2^{-n_l^{0.49}} \leq P_{e,typ} \quad (26)$$

meets the $P_{e,typ}$ requirement. The condition given in (26) is equivalent to

$$\log n_l + n_l^{0.49} \geq \log \frac{1}{P_{e,typ}} + \log N. \quad (27)$$

We will lower bound n_l with even stronger inequality

$$n_l^{0.49} \geq \log \frac{1}{P_{e,typ}} + \log N. \quad (28)$$

This gives the lower bound

$$n_l \geq \left(\log N + \log \frac{1}{P_{e,typ}}\right)^{1/0.49} \quad (29)$$

If we let $\delta = 0.0205$, we obtain the lemma. □

REFERENCES

[1] Rate-compatible puncturing of low-density parity-check codes. *IEEE Transactions on Information Theory*, 50(11):2824–2836, 2004.
[2] E. Arıkan. A performance comparison of polar codes and reed-muller codes. *IEEE Communications Letters*, 12(6):447–449, June 2008.
[3] E. Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, July 2009.
[4] G. Berhault, C. Leroux, C. Jego, and D. Dallet. Partial sums generation architecture for successive cancellation decoding of polar codes. In *SiPS 2013 Proceedings*, pages 407–412, Oct 2013.
[5] C. G. Blake and F. R. Kschischang. Energy consumption of vlsi decoders. *IEEE Transactions on Information Theory*, 61(6):3185–3198, June 2015.
[6] Christopher G. Blake and Frank R. Kschischang. Energy complexity of polar codes. *IEEE International Symposium on Information Theory - Proceedings*, 2016-Augus:810–814, 2016.
[7] C.D.Thompson. *Communication in networks for coordinating behavior*. PhD thesis, Carnegie Mellon University, 1980.
[8] A. Eslami and H. Pishro-Nik. A practical approach to polar codes. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 16–20, July 2011.
[9] Y. Fan and C. y. Tsui. An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation. *IEEE Transactions on Signal Processing*, 62(12):3165–3179, June 2014.

[10] K. Ganesan, P. Grover, J. Rabaey, and A. Goldsmith. On the total power capacity of regular-ldpc codes with iterative message-passing decoders. *IEEE Journal on Selected Areas in Communications*, 34(2):375–396, Feb 2016.

[11] N. Goela, S. B. Korada, and M. Gastpar. On lp decoding of polar codes. In *2010 IEEE Information Theory Workshop*, pages 1–5, Aug 2010.

[12] P. Grover, A. Goldsmith, and A. Sahai. Fundamental limits on the power consumption of encoding and decoding. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 2716–2720, July 2012.

[13] Pulkit Grover. "Information-friction" and its impact on minimum energy per communicated bit. *IEEE International Symposium on Information Theory (ISIT)*, 2013.

[14] Venkatesan Guruswami and Patrick Xia. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Transactions on Information Theory*, 61(1):3–16, 2015.

[15] Joachim Hagenauer. Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications. *IEEE Transactions on Communications*, 36(4):389–400, 1988.

[16] S. H. Hassani, K. Alishahi, and R. L. Urbanke. Finite-length scaling for polar codes. *IEEE Transactions on Information Theory*, 60(10):5875–5898, Oct 2014.

[17] Sheryl L. Howard, Christian Schlegel, and Kris Iniewski. Error control coding in low-power wireless sensor networks: When is ECC energy-efficient? *Eurasip Journal on Wireless Communications and Networking*, 2006(2):1–14, 2006.

[18] H. Jeong and P. Grover. Energy-adaptive codes. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 132–139, Sept 2015.

[19] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross. A semi-parallel successive-cancellation decoder for polar codes. *IEEE Transactions on Signal Processing*, 61(2):289–299, Jan 2013.

[20] C. Leroux, I. Tal, A. Vardy, and W. J. Gross. Hardware architectures for successive cancellation decoding of polar codes. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1665–1668, May 2011.

[21] Bin Li, Hui Shen, and David Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters*, 16(12):2044–2047, 2012.

[22] Bin Li, David Tse, Kai Chen, and Hui Shen. Capacity-achieving rateless polar codes. *IEEE International Symposium on Information Theory - Proceedings*, 2016-Augus:46–50, 2016.

[23] Jing Li and Krishna R Narayanan. Rate-compatible low density parity check codes for capacity-approaching arq schemes in packet data communications. In *Communications, Internet, and Information Technology*, pages 201–206, 2002.

[24] M. Mondelli, S. H. Hassani, and R. L. Urbanke. Unified scaling of polar codes: Error exponent, scaling exponent, moderate deviations, and error floors. *IEEE Transactions on Information Theory*, 62(12):6698–6712, Dec 2016.

[25] Eren Sasoglu. Polarization and polar codes. *Foundations and Trends in Communications and Information Theory*, 8(4):259–381, 2012.

[26] Branka Vucetic. An Adaptive Coding Scheme for Time-Varying Channels. *IEEE Transactions on Communications*, 39(5):653–663, 1991.

[27] Dongfeng Yuan Zhen Tian, Quanquan Liang, and Zhen Tian. Energy efficiency analysis of adaptive error corrections in wireless sensor networks. *Wireless Communications and ...*, 9(60672036):401–405, 2008.