Efficient Algorithms for Systematic Polar Encoding

Harish Vangala, Yi Hong, and Emanuele Viterbo

Abstract—Arıkan has laid the foundations of systematic polar codes and has also indicated that the computational complexity order of the systematic polar encoder (SPE) can be the same of a non-systematic polar encoder (NSPE) i.e., $\Theta(N \log N)$. In this paper, we propose three efficient encoders along with their full pseudocode implementations, all with $\Theta(N \log N)$ complexity. These encoders work for any arbitrary choice of frozen bit indices, and they allow a trade-off between the number of XOR computations and the number of bits of memory required by the encoder. We show that our best encoder requires exactly the same number of XORs as that of NSPE.

Index Terms—Encoder, low complexity, systematic polar codes

I. INTRODUCTION

Systematic polar codes (SPC) were proposed by Arıkan and are known for their improved bit error rate (BER) performance compared to the original non-systematic polar codes [1], [2]. In [1], a systematic polar encoder (SPE) was proposed based on a successive cancellation decoder (SCD) over a binary erasure channel (BEC). In [3], another encoder was proposed using a cascade of two non-systematic polar encoder (NSPE) circuits. It was recently proved in [4] that such encoder works only under certain conditions. They also prove that if a given polar code does not satisfy these conditions they can transform it into a new polar code that performs at least as well as the original one. In [5, Section 3], following Arıkan’s recursion, a systematic encoder is proposed for general $\ell \times \ell$ triangular kernels with a complexity $\mathcal{O}(N \log N)$ and a memory of $\mathcal{O}(N)$ bits. We note that all the above SPEs have complexity, memory, and latency higher than an NSPE.

In this paper, we measure the efficiency of an encoder as the exact number of XOR operations and the exact memory required in number of bits for the computations (excluding the input/output). In particular,

1) we present three efficient systematic encoders for SPC, which work for codes with any choice of frozen bit indices.

2) we demonstrate a trade-off between the number of XOR operations and the number of bits required among our three encoders. The least number of XORs required by our non-recursive SPE is exactly equal to that of an NSPE.

3) we revisit [1] and present explicit and efficient recursive encoders with exact memory and XOR counts.

An open access simulation platform, including the encoders in this paper is available in [6].

II. SYSTEM MODEL

Let $\mathcal{N} \triangleq \{1, 2, \ldots, N\}$. Boldface lower and upper case letters represent vectors, and matrices respectively. All the vectors in this paper are row-vectors. Given a vector $z = [z_1, z_2, \ldots]$ and a subset of indices $\mathcal{J}$, $z_\mathcal{J}$ denotes the sub-vector formed by the elements with the indices in $\mathcal{J}$. Similarly, given a matrix $A$, $A_\mathcal{J}$ denotes the sub-matrix formed by the rows at the indices $\mathcal{J}$. Let $F \triangleq \begin{bmatrix} 1 & 0 \end{bmatrix}$ be the standard kernel used to define polar codes and $F^{\otimes n}$ denote the $n$-fold Kronecker product of $n$ copies of $F$.

A. The Non-Systematic Polar Encoder (NSPE)

Binary polar codes are completely specified by a triple $(N, K, I)$, where $N$ is the code length, $K$ is the length of the message, and $I \subseteq \mathcal{N}$; $|I| = K$ is the set of indices known as information bit indices. The remaining $N - K$ indices are called as frozen bit indices. Here, $N$ is a power of 2 and we define $n \triangleq \log_2(N)$.

For a $(N, K, I)$ polar code, the generator matrix is $G = (F^{\otimes n})_I$. Therefore, given a message vector $u$ of $K$ information bits, a codeword $x$ is generated as:

$$x = u \cdot G = d \cdot F^{\otimes n}$$

(1)

where $d$ is a vector of $N$ bits including information bits such that $d_I = u$, $d_{I^c} = 0$, and $I^c \triangleq \mathcal{N} \setminus I$. The bits $d_{I^c}$ are called as frozen bits, and are set to zero.

Due to the recursive construction of the matrix $F^{\otimes n}$, we can perform this matrix-vector multiplication in $\Theta(N \log N)$ only. This NSPE is illustrated in Fig. 1, and it requires exactly $(\frac{N}{2} \log_2 N)$ XORs and $2N$ bits (i.e. for the extreme nodes). We will use these values to benchmark our new encoders.

![Fig. 1. The non-systematic polar encoder (NSPE) with an exact complexity of $(\frac{N}{2} \log_2 N)$ XORs, when $(N, K, I) = (8, 5, \{2, 4, 6, 7, 8\})$](image)

B. The Systematic Polar Encoding

A systematic polar code may be described as equivalent to the original polar code in (1), except that the message vectors are mapped to codewords, such that the message-bits are explicitly visible [1]. Consider the indices of $K$ bits in a codeword $x$, where the message bits appear explicitly. It was

\[d_1 = 0 \quad d_2 = u_1 \quad d_3 = 0 \quad d_4 = u_2 \quad d_5 = 0 \quad d_6 = u_3 \quad d_7 = u_4 \quad d_8 = u_5\]
shown in [1] that this set can be chosen equal to the set of information bit indices \( I \). Note that, it slightly differs from what is usually considered as a systematic linear block code, where message bits appear as first \( K \) bits. This motivates our interpretation of the SPE below.

The SPE of an information vector \( u \) of \( K \) bits, is the solution of:

\[
x = y \cdot F^\otimes n
\]  

(2)

where, \( y_x \) and \( x_{z} \) are the unknowns. Within the systematic codeword \( x \), \( x_{z} = u \) are information bits and \( x_{z} \) are parity bits. Frozen bits are given by \( y_{z} = 0 \). For example, if \((N, K, I) = (4, 2, \{1, 3\}) \) and \( u = [1 \ 0] \), then (2) becomes:

\[
[1, x_2, 0, x_4] = [y_1, 0, y_3, 0]
\]

where \( y_1, y_3 \) and \( x_2, x_4 \) are the unknowns. Note that there are exactly \( N \) unknowns, shared between \( x \) and \( y \). It is easy to see that (2) is a system of linear equations, up to a re-arrangement.

Since \( F^\otimes n \) is an upper triangular matrix, a straightforward Gaussian elimination (GE) could be used to solve the equations. Then the number of XORs required to perform GE would be approximately equal to the number of ones in the matrix: \( 3 \log_2 N = N \log_2 3 \approx N^{1.585} \). This is much higher than the achievable complexity of \( \Theta(N \log N) \) XORs, as indicated by Arıkan. In fact, devising efficient encoders achieving this complexity bound forms the core objective of this paper.

III. NEW SYSTEMATIC POLAR ENCODERS

In this section, we propose three novel systematic polar encoders based on efficient methods of solving (2). Our first SPE requires the least number of XORs, exactly equal to that of an NSPE. Later, using the recursion in [1] we show two explicit encoders based on efficient methods of solving (2). Our first SPE requires the least number of XORs, exactly equal to that of an NSPE, and \( N(1 + \log_2 N) \) bits (see the pseudocode of EncoderA\( (1) \) and Table I).

B. Recursive SPEs based on Arıkan’s recursion[1]

We revisit Arıkan’s recursive idea based on a divide-and-conquer iteration, which splits the set of equations (2) into two at each step. This enables us to design efficient encoding algorithms that trade the number of XORs with memory.

Consider a vector \( z = [z_1, \ldots, z_N] \) of \( N \) elements. We define a level-\( l \) partition of \( z \) as,

\[
z = [z_1^{(l)}, \ldots, z_l^{(l)}, z_{l+1}^{(l)}], \quad 0 \leq l \leq n,
\]

(3)

where, \( z_i^{(l)} \) denotes a sub-vector of \( z \) of length \( 2^{n-1} \), which can be expanded as \( z_i^{(l)} = [z_{a_i}, \ldots, z_{a_i+2^{n-1}-1}] \), where \( a_i = (2^l - i) \cdot 2^n - l + 1, 1 \leq i \leq 2^l \). In other words, \( z_i^{(l)} \) is simply a pointer to the above sub-array of \( z \) (as in a programming language like C). The partition indices are given in reverse order so that they follow the order of evaluation in our encoders. Such indices also simplify the traversal of a tree structure that we use later. Finally, note that \( z_i^{(n)} = z_{N-i+1} \) and \( z_1^{(0)} = z \).

Let \( \{r_{l,i} : 0 \leq l \leq n, 1 \leq i \leq 2^l \} \) denote a collection of \( 2N - 1 \) different length binary vectors corresponding to the \( 2N - 1 \) nodes of a binary tree shown in Fig. 4. Let \( l \) be the level in the tree (or recursion index) and \( i \) be the sub-vector index, for the nodes at the same level in the tree. All the \( 2^l \) vectors at a given level \( l \) have the same length \( 2^{n-1} \) and can be concatenated as a single length \( N \) vector. Although all these vectors add up to a total of \( N(1 + \log_2 N) \) bits, we will later show how to significantly reduce such memory requirement.

Recursion at level \( l = 1 \) —- Consider the level-1 partitioning:

\[
x = [x_1^{(1)}, x_2^{(1)}] \quad \text{and} \quad y = [y_1^{(1)}, y_2^{(1)}].
\]

Using the block

![Diagram](image-url)
decomposition $\mathbf{F}^\otimes n$, (2) can be rewritten as:
\[
x^{(1)}_1 = y^{(1)}_1 \mathbf{F}^\otimes n-1 \quad \text{and} \quad x^{(2)}_1 = y^{(2)}_1 \mathbf{F}^\otimes n-1 + x^{(1)}_1
\] (4)
\[
x^{(1)}_2 = y^{(1)}_2 \mathbf{F}^\otimes n-1 + x^{(1)}_1
\] (5)

Clearly, the set of equations in (4) can be independently solved (to obtain $x^{(1)}_1$) and substituted in (5) to save one sub-matrix multiplication. Note that solving (5) is different from solving (4) due to the presence of a binary offset vector $x^{(1)}_1$. We initialize these offset vectors as: $r_{1,1} = 0$ and $r_{1,2} = x^{(1)}_1$.

Recursion at levels $2 \leq l \leq n$ — In general, at level $(l - 1)$, we find $2^{l-1}$ equations, relating the level $l - 1$ partition of vectors of $x$ and $y$ as below:
\[
x^{(l-1)}_i = y^{(l-1)}_i \mathbf{F}^\otimes n-l + r_{l-1,i} \quad 1 \leq i \leq 2^{l-1}.
\] (6)

Then, at level $2 \leq l \leq n$, each of the above equations splits similarly to (4) and (5) as follows
\[
x^{(l)}_{2i-1} = y^{(l)}_{2i-1} \mathbf{F}^\otimes n-l + r_{l,2i-1} \quad \text{and} \quad (7)
\]
\[
x^{(l)}_{2i} = y^{(l)}_{2i} \mathbf{F}^\otimes n-l + r_{l,2i} \quad \text{and} \quad (8)
\]
where the new binary offsets $\{r_{l,2i-1}, r_{l,2i}\}$ at level $l$ are computed as
\[
r_{l,2i-1} := (r_{l-1,i})^{(1)} \quad (9)
\]
\[
r_{l,2i} := (r_{l-1,i})^{(1)} + (r_{l-1,i})^{(2)} + x^{(l)}_{2i-1},
\] (10)
where the two halves of $r_{l-1,i}$ are denoted by $(r_{l-1,i})^{(1)}$ and $(r_{l-1,i})^{(2)}$, using a similar notation to (3). Note that (10) can be applied only after solving (7), since the solution $x^{(l)}_{2i-1}$ is required. We will next see a more convenient visualization of these recursive computations. Much different from [5, Section 3], our recursion operates on the binary offset vectors and does not involve any bit-reversal permutation.

Binary tree representation — Consider a perfect binary tree with $n + 1$ levels $l = 0, \ldots, n$, with root node at level $0$. The $2^{l-1}$ nodes at a level $l - 1$ are associated with the $2^{l-1}$ set of $2^{n-l+1}$ equations from (6). Each node at level $l - 1$ splits into two at level $l$ as shown in Fig. 3. The solution of (2) is obtained by an in-order traversal of the tree. Each time we visit a leaf node (i.e., at $l = n$), we obtain a simple equation of the form: $x^{(n)}_i = y^{(n)}_i + r_{n,i}$, where either $x^{(n)}_i$ or $y^{(n)}_i$ is the unknown to solve. Computing $r_{n,i}$ using the successive updates of (9) and (10), is therefore the critical objective of this traversal. The binary offsets $r_{l,i}$ are stored in a tree structure in Fig. 4 as explained before.

As a part of the in-order traversal of the tree in Fig. 3 (or Fig. 4), we visit each node three times, except for the leaf nodes which are visited only once.

Let us consider the sequence of visits and respective operations for a node at level $l - 1$ in the tree as shown in Figs. 3 and 4.

1) First Visit — The required offsets $r_{l,2i-1}$ are simply copied using (9). Then a recursive call down the left branch is made which, upon return, gives the solution (7).

2) Second Visit — Returning from the left sub-tree we have full knowledge of $x^{(l)}_{2i-1}$, so that $r_{l,2i}$ can be updated by using (10). Then a recursive call down the right branch is made which, upon return, gives the solution of (8).

3) Third Visit — By now we have visited the entire sub-tree and we have all the solutions of (6). Then we return to the parent node.

The exact XOR count — Let $f(N)$ denote the number of XORs required to solve (2). From the recursive equations (7)–(10) we have:

\[
f(N) = 2 \cdot f\left(\frac{N}{2}\right) + \beta \frac{N}{2} \quad \text{and} \quad f(1) = 1.
\] (11)

The first term on the right hand side accounts for solving recursively (7) and (8), while the second term $\beta \frac{N}{2}$ accounts for the computations required by (9) and (10). In the case where we have allocated memory for each node of the tree in Fig. 4 (namely, EncoderB), we have $\beta = 2$. A different implementation (namely, EncoderC), saving on the memory to store the intermediate values of the $r_{l,i}$ offset vectors will have $\beta = 4$. The closed form solution of (11) yields $f(N) = N + \frac{\beta N}{2} \log_2 N$.

Note that we ignore all the index computations to address the subvectors and any operation to identify frozen bits that are implementation dependent and can be avoided by hardwiring. 

EncoderB requires exactly $N(1 + \log_2 N)$ XORs, which is approximately the same number of XORs required by the encoder proposed in [3]. However, our EncoderB works without any restrictions on code rate and frozen bit indices. Our pseudocode EncoderB() implements the storage of offset vectors as a local variable $v$ allocated upon each recursive call and released on return. At any point in the tree traversal, we have a chain of at most $n + 1$ recursive function calls being active (a path from the root node to a leaf node in Fig. 4). Therefore, the maximum size of allocated memory by these
function calls is exactly \( N + \frac{N}{2} + \ldots + 1 = 2N - 1 \) bits. Note that the same memory allocation/deallocation process can be implemented by simply addressing an array of \( 2N - 1 \) bits.

**EncoderC** provides a memory-efficient implementation by using a single \( N \)-bit vector, since at any level \( l \) of Fig. 4, we need to store \( N \) bits of binary offsets \( r_l \). However, this slightly increases the number of XORs due to the following reason. We update the offsets every time we visit a non-leaf node for the second time. Therefore when we visit it for the third time, we should restore the original offset to enable computations at the parent node (see Fig. 4). This requires using twice equation \((10)\). Hence doubling the number of XORs yields \( \beta = 4 \), and leads to an overall \( N(1 + 2 \log_2 N) \) XOR count, with only \( N \) bits memory (exclusive IO). **EncoderC** provides its pseudocode.

### Table I
**Summary of Systematic Polar Encoders**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recursion</th>
<th># bits (excl. IO)</th>
<th># XORs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EncoderA</td>
<td>No</td>
<td>( N(1 + \log_2 N) )</td>
<td>( \frac{N}{2} \log_2 N )</td>
</tr>
<tr>
<td>EncoderB</td>
<td>Yes</td>
<td>( 2N - 1 )</td>
<td>( N(1 + \log_2 N) )</td>
</tr>
<tr>
<td>EncoderC</td>
<td>Yes</td>
<td>( N )</td>
<td>( N(1 + 2 \log_2 N) )</td>
</tr>
<tr>
<td>NSPE</td>
<td>Yes/No</td>
<td>( 2N )</td>
<td>( N \log_2 N )</td>
</tr>
</tbody>
</table>

### IV. Conclusions
We have designed three efficient, low complexity algorithms to perform systematic polar encoding, at the same complexity order of \( \Theta(N \log N) \). All the three algorithms work for any arbitrary choice of frozen bit indices (not necessarily polar). They further illustrate a trade-off between the required number of XORs and the memory required for an SPE. We benchmark our encoders with a standard NSPE as detailed in Table I. Our first encoder is a non-recursive encoder that requires the least number of XORs, exactly equal to that of an NSPE. The remaining two encoders are recursive, that require approximately twice and four times the number of XORs as an NSPE, respectively. Further improvements to the encoders such as higher parallelization and lower memory are relegated to our future work.

### References