

A Low Complexity Encoding Algorithm for Systematic Polar Codes

Guo Tai Chen, Zhaoyang Zhang, Caijun Zhong, and Liang Zhang

Abstract—Arıkan has shown that systematic polar codes (SPC) outperform nonsystematic polar codes (NSPC). However, the performance gain comes at the price of elevated encoding complexity, i.e., compared to NSPC, the available encoding methods for SPC require higher memory and computation. In this letter, we propose an efficient encoding algorithm requiring only N bits of memory and having $\frac{N}{2} \log_2 N$ XOR operations. Moreover, the auxiliary variables in the algorithm can share the memory to reduce extra memory requirement. Furthermore, a parallel 2-bit encoding algorithm is also presented to improve the encoding throughput. Remarkably, we show that parallel encoding can be implemented with the same number of XOR operations and memory bits. Finally, the proposed encoding algorithm can be directly used for NSPC with the same complexity.

Index Terms—Polar codes, systematic polar codes, encoding algorithm, parallel encoding.

I. INTRODUCTION

Polar codes, originally proposed by Arıkan in [1], have gained enormous interests due to a number of distinctive features. For instance, polar codes have explicit coding structure and can achieve the capacity of symmetric binary memoryless channels (S-BMC). Moreover, polar codes with finite length yield competitive performance when compared to LDPC [2] and Turbo codes [3] in addition to having low encoding and decoding complexity.

The standard polar codes are in nonsystematic form where both frozen bits and information bits (also referred to as user bits) are placed on the polarized bit-channels of the polarization structure and the user bits do not appear in the polar codeword. However, information bits as part of the codeword are required in some scenarios, such as the famous Turbo codes [4] whose component codes are systematic codes that can exchange information between modules in turbo decoding. To construct systematic polar codes (SPC), Arıkan proposed the idea of shifting the user bits from polarized bit-channels to unpolarized bit-channels [5], which makes the frozen and user bits lie on two different extremes of

G. T. Chen (email: chenguot@163.com) is with Fuqing Branch of Fujian Normal University, China, and he is also a domestic visitor at Zhejiang University from 2015 to 2016. Z. Zhang (Corresponding Author, email: ning_ming@zju.edu.cn), C. Zhong (email: caijunzhong@zju.edu.cn) and L. Zhang are with the College of Information Science and Electronic Engineering, Zhejiang University, China.

This work was supported in part by the National Key Basic Research Program of China under Grant 2012CB316104, the National Natural Science Foundation of China under Grants 61371094 and 61401099, the Huawei HIRP Flagship Projects under Grants YB2015040053 and YB2013120029, Zhejiang Provincial Natural Science Foundation of China under Grant LR15F010001, the Fundamental Research Funds for the Central Universities under Grant 2016QNA5004, the Research Funds of the Education Department of Fujian Province under Grants JA12350 and JA14339.

TABLE I
SUMMARY OF SYSTEMATIC POLAR ENCODERS

Algorithm	Recursion	# bits(excl. I/O)	# XORs
EncoderA [9]	No	$N(1 + \log_2 N)$	$\frac{N}{2} \log_2 N$
EncoderB [9]	Yes	$2N - 1$	$N(1 + \log_2 N)$
EncoderC [9]	Yes	N	$N(1 + 2 \log_2 N)$
NSPC [9]	Yes/No	$2N$	$\frac{N}{2} \log_2 N$
Proposed SPC	No	N	$\frac{N}{2} \log_2 N$

polarization structure. Arıkan showed that systematic polar codes outperform nonsystematic polar codes (NSPC) in terms of bit error ratio (BER) and the performance have also been investigated in [6].

Recently, SPC as component codes of concatenated codes have been investigated in [7] and [8]. Compared to the NSPC with the same polarization structure, SPC is inherently more complex. Hence, to facilitate the application of SPC, the key challenge is to find an efficient encoding method. In [5], Arıkan presented a recursive method for SPC encoding with $\alpha \frac{N}{2} \log_2 N$ ($\alpha > 1$) XOR operations, where Arıkan also suggested using successive cancellation (SC) decoder as an encoder for SPC. Following this suggestion, an SPC encoder which facilitates easy parallelization was proposed in [10], [11], with the limitation of executing SC algorithm twice and constrained frozen bits. Another SPC encoding algorithm in the recursive implementation with elimination method was presented in [12]. Most recently, the authors of [9] proposed three encoding algorithms for SPC with memories of $N(1 + \log_2 N)$, $2N - 1$ and N bits and XOR operations of $\frac{N}{2} \log_2 N$, $N(1 + \log_2 N)$ and $N(1 + 2 \log_2 N)$, respectively. However, the major drawback of the above discussed encoding methods is the high requirements on memory or computation, which may not be suitable for devices with small size and limited power. Motivated by this, in this letter, we propose a new efficient encoding algorithm for SPC requiring only N bits of memory (excluding the input/output) and $\frac{N}{2} \log_2 N$ XOR operations. To the best of the authors' knowledge, the proposed algorithm requires the minimum memory as well as XOR operations compared to the known encoding methods, as illustrated in Table I. In addition, to further improve the encoding throughput, a parallel 2-bit encoding algorithm is also discussed, which shows that the parallel encoding can be accomplished without incurring additional cost in terms of XOR operation and memory bit.

II. NONSYSTEMATIC AND SYSTEMATIC POLAR CODES

For polar codes with codeword length $N(= 2^n, n \geq 1)$ and kernel matrix $F = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, the polarization transformation

matrix G can be written as $G = F^{\otimes n}$ where \otimes denotes the Kronecker power operation. Let $\mathbf{u} = (u_0, u_1, \dots, u_{N-1})$ and $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$ be the bit vectors on the left and right side of the encoder shown in Fig.1, respectively, then we have

$$\mathbf{x} = \mathbf{u}G. \quad (1)$$

For NSPC, \mathbf{u} is the only input of the encoder, i.e., \mathbf{u} includes both the frozen and user bits, and the encoding is performed from left to right according to the coding structure shown in Fig.1, where \mathcal{A} denotes the index set of the user bits.

However, for SPC, both \mathbf{u} and \mathbf{x} are inputs of the encoder. To construct systematic polar codes, Arikan proposed to place the user bits on the right side of the encoder and keep the same indices for the bits as illustrated in Fig.1 with $N = 2^3$ and $\mathcal{A} = \{1, 3, 5, 6, 7\}$, where the left extreme node with hollow arrow denotes the frozen bit while the right extreme node with solid arrow denotes the user bit.

The index set for the frozen bits is the complementary set of \mathcal{A} , i.e., $\mathcal{A}^c = \{0, 1, \dots, N-1\} - \mathcal{A}$. Now, denote $\mathbf{u}_{\mathcal{A}}$ and $\mathbf{u}_{\mathcal{A}^c}$ as the bit vector with elements u_i , $i \in \mathcal{A}$ and $i \in \mathcal{A}^c$, respectively, and the similar denotation is also for $\mathbf{x}_{\mathcal{A}}$ and $\mathbf{x}_{\mathcal{A}^c}$, then, Equation (1) can be rewritten as

$$(\mathbf{x}_{\mathcal{A}} \ \mathbf{x}_{\mathcal{A}^c}) = (\mathbf{u}_{\mathcal{A}} \ \mathbf{u}_{\mathcal{A}^c}) \begin{pmatrix} G_{\mathcal{A}\mathcal{A}} & G_{\mathcal{A}\mathcal{A}^c} \\ G_{\mathcal{A}^c\mathcal{A}} & G_{\mathcal{A}^c\mathcal{A}^c} \end{pmatrix}, \quad (2)$$

where $G_{\mathcal{A}\mathcal{A}^c}$ is a sub-matrix of G with elements $G_{i,j}$, $i \in \mathcal{A}$ and $j \in \mathcal{A}^c$, and $G_{\mathcal{A}\mathcal{A}}$, $G_{\mathcal{A}^c\mathcal{A}}$ and $G_{\mathcal{A}^c\mathcal{A}^c}$ are defined in the same fashion. The objective of SPC encoding is to obtain $\mathbf{x}_{\mathcal{A}^c}$ given the inputs $\mathbf{u}_{\mathcal{A}^c}$ and $\mathbf{x}_{\mathcal{A}}$.

Since matrix $G_{\mathcal{A}\mathcal{A}}$ is invertible, $\mathbf{x}_{\mathcal{A}^c}$ can be computed by [5]:

$$\mathbf{x}_{\mathcal{A}^c} = (\mathbf{x}_{\mathcal{A}} + \mathbf{u}_{\mathcal{A}^c} G_{\mathcal{A}^c\mathcal{A}}) G_{\mathcal{A}\mathcal{A}}^{-1} G_{\mathcal{A}\mathcal{A}^c} + \mathbf{u}_{\mathcal{A}^c} G_{\mathcal{A}^c\mathcal{A}^c}. \quad (3)$$

As discussed in the Introduction section, the known methods in literature to compute $\mathbf{x}_{\mathcal{A}^c}$ requires relatively large memory and heavy computation. Motivated by this, the main objective of this letter is to find an efficient algorithm to compute $\mathbf{x}_{\mathcal{A}^c}$.

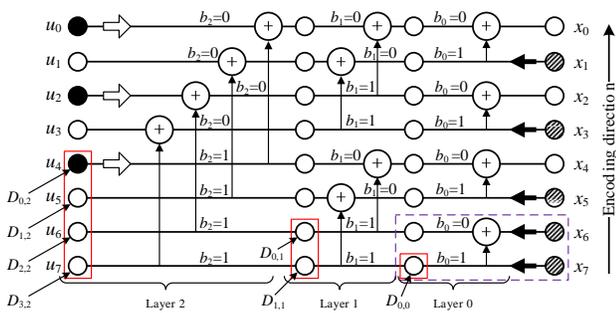


Fig. 1. SPC encoding diagram with $N = 2^3$ and $\mathcal{A} = \{1, 3, 5, 6, 7\}$.

III. EFFICIENT ENCODING ALGORITHM FOR SPC

The proposed encoding algorithm is similar to the algorithm **EncoderA** in [9] where the encoding is implemented from the bottom horizontal connection to the top horizontal connection and the calculation for each horizontal connection starts from known node (i.e., one of elements of $\mathbf{x}_{\mathcal{A}}$ or $\mathbf{u}_{\mathcal{A}^c}$) and moves

from one node to the next and to the other side of the polarization structure, as shown in Fig.1. Due to the fact that each node is allocated with one bit memory in **EncoderA**, the total memory requirement of **EncoderA** is $N(1 + \log_2 N)$ bits. The key feature of the proposed encoding algorithm is to reduce the memory requirement from $N(1 + \log_2 N)$ bits to N bits while maintaining the same computation load, i.e., $\frac{N}{2} \log_2 N$ XOR operations.

A. Encoding algorithm

To exploit the recursive nature of polar codes, the polarization structure with $N = 2^n$ is divided into n layers labeled by $0, 1, \dots, (n-1)$ from right to left as shown in Fig.1. And for layer λ ($\lambda = 0, 1, \dots, n-1$), the N nodes (includes the corresponding operations) are separated into $2^{(n-1)-\lambda}$ blocks from top to bottom and each block contains $2^{\lambda+1}$ elements. As an illustration, the dashed box in the right-bottom corner of Fig.1 represents one of the first-layer blocks.

To analyze the memory requirement of the encoding algorithm, let us first consider the blocks in the same layer. The key observation is that the blocks in same layer are independent and there is no information exchange between the blocks in the same layer, which indicates that the memory used for one block can be recycled for the other blocks in the same layer when the encoding proceeds from the bottom up. In addition, a close inspection reveals that, in each block, only the lower half of the elements need to be stored in memory for the encoding process, and the outcomes of the XOR operations in the upper half can be stored in the associated lower half. Then, it is easy to show that only 2^λ bits of memory are required for the encoding process in layer λ . Hence, the total required memory of all layers is $2^n - 1 = N - 1$ bits.

To corroborate the above argument, let us consider the following illustrative example. For notational convenience, we define $D_{a_\lambda, \lambda}$ as the memory address used for layer λ , where a_λ ($a_\lambda = 0, 1, \dots, 2^\lambda - 1$) is the index of bit memory. We focus on the bottom block in layer 0 highlighted by the dashed box in Fig. 1. In this block, x_6 and x_7 are the known bits and x_7 will be first processed. The first step is to copy x_7 to $D_{0,0}$, and then to $D_{1,1}$ and $D_{3,2}$. Since the stored value in $D_{0,0}$ is the same as x_7 , the XOR operation between x_6 and x_7 can be replaced with x_6 and $D_{0,0}$ and the outcome of the XOR operation can be stored in $D_{0,0}$, since the previous value in $D_{0,0}$ is obsolete. The next step is to copy the value of $D_{0,0}$ to $D_{0,1}$. Once done, $D_{0,0}$ is released since its value is no longer required in the remaining process, hence can be recycled for use in the next block in layer 0. As such, only $1 = (2^0)$ bit memory is required for the encoding process in layer 0. Finally, the above process is extended to other layers.

In the encoding process of SPC, there exist two different operations, namely, directly copying and XOR operation. Hence, it is of significant interest to obtain a fast method to determine the proper operation. Here, we present a simple method to address this issue. Let ϕ ($\phi = 0, 1, \dots, N-1$) denote the index of current horizontal connection for top to bottom and denote the binary expression of ϕ as $b_{n-1} \dots b_0$. Then, directly copying is performed in layer λ when $b_\lambda = 1$

and XOR operation occurs when $b_\lambda = 0$, as illustrated in Fig. 1.

For the propagation from left to right, the case for $b_\lambda = 0$ is more complex. Let us take the information propagation of u_0 for example. Since both u_0 and $D_{0,2}$ are needed for the XOR operation $u_0 \oplus D_{0,2}$ at the same time, we can not copy u_0 to $D_{0,2}$. To circumvent this problem, we introduce a temporary variable t , and set $t = u_0$. Hence, the XOR operation $x_0 \oplus D_{0,2}$ can be replaced with $t \oplus D_{0,2}$, and the corresponding outcome can be assigned to t as well, i.e., $t = t \oplus D_{0,2}$. When $b_\lambda = 1$, the directly copying operation is to copy t into $D_{a_\lambda, \lambda}$, which implies that t and $D_{a_\lambda, \lambda}$ share the same value after the directly copying, hence t can then be used for the following operations instead of $D_{a_\lambda, \lambda}$. Due to the introduction of a temporary variable, the total required memory bits for the proposed encoding process is N . It is also worth emphasizing that the number of XOR operations in the proposed encoding process is only $\frac{N}{2} \log_2 N$.

The pseudocodes of the proposed encoding of SPC is listed in **Algorithm 1** where \leftarrow is the assignment operator. Lines 6-15 are for the propagation from right to left, and lines 17-27 are for the propagation from left to right. After each process of propagation, a_λ will be updated from the next propagation, which is shown in lines 28-31.

Comparing the pseudocodes between **EncoderA** in [9] and **Algorithm 1**, it can be found that the encoding processes of both algorithms work in the same serial fashion and are implemented from horizontal connection $(N-1)$ to horizontal connection 0 one by one, which indicates that both algorithms have the same number of XOR operations and directly copying. However, our proposed algorithm repeatedly utilizes the N -bit memory while **EncoderA** requires $N(1 + \log_2 N)$ bits of memory. Moreover, the XOR operation in the proposed algorithm only requires two operands and is performed in place while the XOR operation in **EncoderA** has three operands including one destination and two sources, which may incur extra computation. And we will show in the next subsection that the updating of b_λ and a_λ does not need extra computation. Therefore, the efficiency of the proposed encoding algorithm has not degraded in comparison to **EncoderA** in [9].

It is also worth pointing out that the proposed algorithm can also be used as an encoding algorithm for NSPC where the encoding only has the propagation from left to right based on the polarization structure similar to Fig.1. This indicates that the minimum requirement of NSPC encoding is also $\frac{N}{2} \log_2 N$ XOR operations and N bits of memory.

B. Simplification for SPC encoder

One might think that we need extra bit memory for b_λ and a_λ and extra computation for the updating of a_λ . In fact, ϕ , b_λ and a_λ can share the same memory and only updating ϕ is enough for all updating. We will show this in the following.

In hardware implementation, ϕ is expressed in binary as $(b_{n-1} \cdots b_1 b_0)_2$, which is shown in Fig.2. When ϕ is updated, the operations in current horizontal connection are decided by the values of b_{n-1}, \dots, b_1 and b_0 . $(b_{n-1} \cdots b_1 b_0)$ will

Algorithm 1: Proposed Encoding algorithm for SPC

Input: \mathbf{u} and \mathbf{x} with unfilled bits (variables of (1));
Output: Full codeword \mathbf{u} and \mathbf{x} ;

```

1 for  $\lambda = 0 : (n-1)$  do //initialization
2    $a_\lambda \leftarrow 2^\lambda - 1$ ;
3 for  $\phi = (N-1) : -1 : 0$  do
4   Store binary expression of  $\phi$  into  $b_{n-1} \cdots b_0$ ;
5   if  $\phi \in \mathcal{A}$  then //information bits
6     if  $b_0 = 0$  then
7        $D_{0,0} \leftarrow D_{0,0} \oplus x_\phi$ ;
8     else
9        $D_{0,0} \leftarrow x_\phi$ ;
10    for  $\lambda = 1 : (n-1)$  do
11      if  $b_\lambda = 0$  then
12         $D_{a_\lambda, \lambda} \leftarrow D_{a_\lambda, \lambda} \oplus D_{a_{\lambda-1}, \lambda-1}$ ;
13      else
14         $D_{a_\lambda, \lambda} \leftarrow D_{a_{\lambda-1}, \lambda-1}$ ;
15     $u_\phi \leftarrow D_{a_{n-1}, n-1}$ ;
16  else //frozen bits
17     $t \leftarrow u_\phi$ ;
18    for  $\lambda = (n-1) : -1 : 1$  do
19      if  $b_\lambda = 0$  then
20         $t \leftarrow t \oplus D_{a_\lambda, \lambda}$ ;
21      else
22         $D_{a_\lambda, \lambda} \leftarrow t$ ;
23    if  $b_0 = 0$  then
24       $x_\phi \leftarrow x_\phi \oplus t$ ;
25    else
26       $x_\phi \leftarrow t$ ;
27       $D_{0,0} \leftarrow t$ ;
28  for  $\lambda = 1 : (n-1)$  do
29    if  $a_\lambda = 0$  then
30       $a_\lambda \leftarrow 2^\lambda$ ;
31     $a_\lambda \leftarrow -$ ;

```

be bitwise visited as one switch to select the corresponding operation for the propagation from right to left (or vice versa).

In layer λ , 2^λ bits memory are required, which means a_λ must be a number of λ bits. Note that $(b_{\lambda-1} \cdots b_0)_2$ has the same value (in decimal) as a_λ ($\lambda > 0$). Thus, a_λ can be obtained by selecting $b_{\lambda-1} \cdots b_0$ from ϕ without extra memory as shown in Fig.2. In layer 0, a_0 is fixed to be 0 due to that only one bit is required.

In **Algorithm 1**, if a full word \mathbf{u} is not required, line 15 can be deleted.

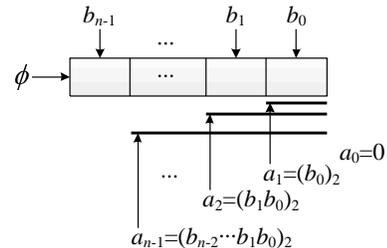


Fig. 2. The bits memory for ϕ , b_λ and b_λ

IV. DISCUSSION ON PARALLEL ENCODING

The encoding algorithm described in **Algorithm 1** works in a serial fashion. To further improve the encoding throughput, we discuss the implementation of a parallel encoding with 2 bits at a time in this section.

Similar to the previous algorithm, the encoding is processed from bottom to top as shown in Fig.1. Let 2ψ and $2\psi + 1$ denote the indices of the two horizontal connections being processed at a time, respectively. Here $\psi \in \{0, 1, \dots, \frac{N}{2} - 1\}$. From Fig.1, it can be found that there are four different cases of information propagation for two known bits in the encoding process as depicted in Fig.3. However, it turns out that case (d) never happens for polar codes constructed on symmetric memoryless channels (S-BMC).

Proposition 1. For $N = 2^n$ ($n \geq 1$) polar codes with coding structure similar to Fig.1, the (2ψ) -th bit channel, $W_{2\psi}$, must be frozen if the $(2\psi + 1)$ -th bit channel, $W_{2\psi+1}$, is frozen on S-BMC.

Proof: The result can be obtained by invoking Lemma 5 in [13]. ■

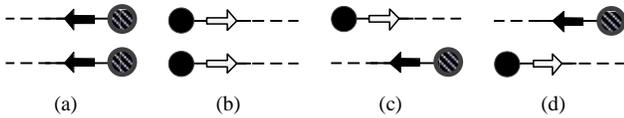


Fig. 3. The four cases with top being the (2ψ) -th bit and bottom being the $(2\psi + 1)$ -th bit: (a) both are user bits, (b) both are frozen bits, (c) frozen bit on top and user bit at the bottom, (d) user bit on top and frozen bit at the bottom.

We now elaborate on the parallel encoding algorithm and the corresponding memory requirements, where $\phi = 2\psi$ and ψ decreases from $(\frac{N}{2} - 1)$ to 0. a_λ ($\lambda > 0$) and b_λ are obtained in the same way as in Subsection III.B and a_0 is updated to be the same as a_1 . In the following, we use $\mathbf{D}_{a_\lambda, \lambda}^{+1}$, \mathbf{u}_ϕ^{+1} and \mathbf{x}_ϕ^{+1} to denote the two-bit vectors $(D_{a_\lambda, \lambda}, D_{a_\lambda+1, \lambda})$, $(u_\phi, u_{\phi+1})$ and $(x_\phi, x_{\phi+1})$, respectively.

As shown in Fig.1, in layer 0 of case (a), with parallel processing, the direct copying operation and XOR operation are performed simultaneously. As such, one additional memory bit, denoted as $D_{1,0}$, is required and then the above operations are done as $\mathbf{D}_{0,0}^{+1} \leftarrow (x_\phi \oplus x_{\phi+1}, x_{\phi+1})$. In layer $\lambda (> 0)$, it can be noticed that the operation types of the ϕ -th and $(\phi + 1)$ -th user bits are the same and will be decided according to b_λ , where the operations of $\mathbf{D}_{a_\lambda, \lambda}^{+1} \leftarrow \mathbf{D}_{a_\lambda-1, \lambda-1}^{+1}$ are implemented when $b_\lambda = 1$ while the operations of $\mathbf{D}_{a_\lambda, \lambda}^{+1} \leftarrow \mathbf{D}_{a_\lambda, \lambda}^{+1} \oplus \mathbf{D}_{a_\lambda-1, \lambda-1}^{+1}$ for $b_\lambda = 0$. And the operations of $\mathbf{u}_\phi^{+1} \leftarrow \mathbf{D}_{a_{n-1}, n-1}^{+1}$ will be done at the last of case (a).

Note that $\mathbf{D}_{0,0}^{+1}$ are idle when the process is done from layer $n - 1$ to 1 and we use them as temporary variables in case (b). Then, the process of case (b) is the same as lines 17-22 in **Algorithm 1** but 2-bits per computation cycle. At the last, \mathbf{u}_ϕ^{+1} is updated with $(D_{0,0} \oplus D_{1,0}, D_{1,0})$ where the new values of $\mathbf{D}_{0,0}^{+1}$ after layer 1 are also expected values in the encoding.

Case (c) has two opposite information propagations and still works in the serial mode as **Algorithm 1** in which $D_{0,0}$ and

$D_{1,0}$ are used to replace t in the frozen bit processing part and $D_{0,0}$ in the user bit processing part, respectively.

Consider polar codes with $N = 1024$ and code rate 1/2 constructed at signal-to-noise ratio 2dB under additive white Gaussian noisy channel as an illustration, the number of case (a), (b) and (c) are 135, 135 and 242 respectively, that is, 754 horizontal propagations will be implemented with our proposed parallel encoding, which can obtain about 36% gain in throughput with comparison to **Algorithm 1**.

From the above description, it can be noted that a new bit memory, i.e. $D_{1,0}$, is introduced in the parallel 2-bit encoding algorithm while the temporary variable t in **Algorithm 1** is no longer required. Hence, the requirement of bit memory for the parallel 2-bit encoding algorithm is the same as **Algorithm 1**. Also, it can be easily verified that the number of XOR operations remains unchanged.

Limited to the two opposite information propagations of frozen bits and user bits in the SPC encoding, parallel multiple-bit encoding for SPC is more complex and the storage memory and computation will also increase, which is beyond the discussion of this paper and will be left for future work.

V. CONCLUSIONS

We have presented an efficient encoding algorithm for SPC which need N bits memory and $\frac{N}{2} \log_2 N$ XOR operation, a minimum requirement for the available encoding methods. By sharing memory, our analysis shows the algorithm can be further simplified. To improve encoding throughput, a parallel 2-bit encoding algorithm has also been discussed, which shows the parallel encoding algorithm can be achieved with the same memory bits and XOR operations.

REFERENCES

- [1] E. Arıkan, "Channel polarization: A method for constructing capacity achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051-3073, Jul. 2009.
- [2] I. Tal and V. Alexander, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213-2226, May 2015.
- [3] K. Niu and K. Chen, "CRC-Aided decoding of polar codes," *IEEE communications letters*, vol. 16, no. 10, pp. 1668-1671, Oct. 2012.
- [4] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.
- [5] E. Arıkan, "Systematic polar coding," *IEEE communications letters*, vol. 15, no. 8, pp. 860-862, Aug. 2011.
- [6] L. Li, W. Zhang and Y. Hu, "On the error performance of systematic polar codes," *arXiv preprint*, 2015, arXiv:1504.04133.
- [7] D. Wu, A. Liu, Y. Zhang and Q. Zhang, "Parallel concatenated systematic polar codes," *Electronics Letters*, vol. 52, no. 1, pp. 43-45, Jan. 2016.
- [8] Q. Zhang, A. Liu, Y. Zhang and X. Liang, "Practical Design and Decoding of Parallel Concatenated Structure for Systematic Polar Codes," *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 456-466, Feb. 2016.
- [9] H. Vangala, Y. Hong and E. Viterbo, "Efficient algorithms for systematic polar encoding," *IEEE communications letters*, vol. 20, no. 1, pp. 17-20, Jan. 2016.
- [10] G. Sarkis, P. Giard, A. Vardy, C. Thibault and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946-957, May 2014.
- [11] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibault and W. J. Gross, "Flexible and low-complexity encoding and decoding of systematic polar codes," *arXiv:1507.03614 [cs.IT]*, pp. 1-9, 2015.
- [12] N. Presman and S. Litsyn, "Recursive descriptions of polar codes," *arXiv:1209.4818v3 [cs.IT]*, pp. 1-60, 2012.
- [13] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562-6582, Oct. 2013.