MULTI-RATE POLAR CODES FOR SOLID STATE DRIVES

Yi Zhong, Chun Zhang,[*]
Institute of Microelectronics
Tsinghua University, Beijing China

Chenrong Xiong, Zhiyuan Yan[†]
Department of ECE
Lehigh University, Bethlehem, PA USA

ABSTRACT

As solid state drives (SSDs) are gradually replacing hard disk drives, error correction is critical to SSDs since NAND flash has deteriorating reliability over their life span. Existing error correction codes suffer from limited error correction capability or error floor issues. Polar codes are promising for SSDs since they are theoretically proven optimal codes and have good error floor behavior. In this paper, we first design multi-rate polar codes for SSDs, and then implement encoder and decoder that simultaneously support multiple rates in FPGA. Multi-rate polar codes provide a good tradeoff between reliability and efficiency. Finally, we use an FPGA emulation platform to evaluate the error performance of our polar codes, and examine their error floor behavior.

Index Terms— Solid state drive, error correction codes, polar codes, FPGA emulator

1. INTRODUCTION

Although NAND flash memory has high capacity and fast cell access, its storage reliability is an important problem that needs to be alleviated. When the number of program-erase (P/E) cycles increase, the Inter-poly oxide layer in memory will be gradually destructed so that it becomes increasingly hard to trap electrons, leading to growing crossover probability. Various error correction codes (ECCs) have been considered for this problem, but existing ECCs either have limited error correction capability or suffer from error floor problems [1].

Recently proposed polar codes [2] are good candidates as error correction codes for SSDs for two reasons. First, polar codes were proved as capacity-achieving codes. Also, polar codes do no suffer from error floor problems [3].

In this paper, we first design multi-rate polar codes for SSDs, and then implement encoder and decoder that simultaneously support multiple rates in FPGA. Multi-rate polar codes are suitable for SSDs, because NAND flash has deteriorating reliability over their life span. Using multi-rate polar codes provides a good tradeoff between reliability and efficiency (redundancy). In early stages, the crossover probability is lower, and hence polar codes with higher code rates can meet the error correction requirements. When the number of P/E cycles increases, correction ability of high rate codes is no longer enough, and instead lower rate codes are necessary in late stages.

We also use an FPGA emulation platform to evaluate the error performance of our polar codes, and examine their error floor behavior. Our results show that our polar codes provide the desired tradeoff between reliability and redundancy, and that they do not have error floor when bit error rates go down to 10^{-12}. The FPGA platform also sets our work apart from prior works in this area (see, for example, [4]). Prior works such as [4] rely on numerical simulations for error performance evaluation and cannot examine the error performance for very low error rates. Our FPGA platform allows us to evaluate the error performance of polar codes at much lower bit rate than [4].

The remainder of this paper is organized as follows. In Section 2, the encoder and decoders of polar codes are briefly reviewed. In Section 3, we present the design details of multi-rate polar codes. In Section 4, we present our hardware implementation of the encoder and decoder, describe our FPGA emulation platform, and present the error performance of our polar codes. In Section 5, we make some conclusions and discuss some future work.

2. REVIEW

We now briefly review the encoding and decoding operations of polar codes, as they will be discussed later.

For an N-bit ($N = 2^n$) binary dataword $(u_0, u_1, \cdots, u_{N-1})$, denoted as u_0^{N-1}, $N - K$ bits are zeros (called frozen bits) and K bits are information bits. The codeword corresponding to u_0^{N-1}, denoted as x_0^{N-1} is computed as

$$x_0^{N-1} = u_0^{N-1} B_N F^{\otimes n}, \tag{1}$$

where B_N is the $N \times N$ bit-reversal permutation matrix, $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, and $F^{\otimes n}$ is the n-th Kronecker power of F. The (N, K) polar code is said to have a rate of K/N.

The successive cancelation (SC) decoding algorithm [2] decodes the dataword u_0^{N-1} one by one from u_0 to u^{N-1}. If

[*]This project is partly sponsored by the National High Technology Research and Development Program of China (863 Program, No. 2015AA016701).

[†]This work was supported in part by the National Science Foundation under Grant ECCS-1509674.

an encoding bit is a frozen bit, it is set to zero. Otherwise, the channel transition probability associated with this bit is calculated and then ML decoding is performed for this bit based on the channel transition probability.

Instead of making a hard decision for each information bit of u_0^{N-1}, the successive cancelation list (SCL) decoding algorithm [5] generates two paths in which the information bit is assumed to be 0 and 1, respectively. When the number of paths is greater than a predefined list size L, the L most reliable paths are kept. At the end of the decoding procedure, the most reliable path is chosen as the decoding result.

The CA-SCL decoding algorithm [6] is used to decode the CRC-concatenated polar codes. It improves the error performance of polar codes at the expense of a smaller code rate. In the CA-SCL decoding algorithm, the cyclic redundancy check (CRC) is used to select the final decoding result. If there is at least one path satisfying the check, the most likely CRC-valid path is chosen. Otherwise, the most reliable path is selected.

3. MULTI-RATE POLAR CODES

3.1. SSD set-up

As shown in Fig. 1, SSDs generally consist of micro control unit (MCU), ECC, NAND flash memory array and interfaces, and these components form a complete microsystem. For data transmission, individual modules are interconnected by an AXI bus, a direct data path between USB 3.0, SATA, PCIe interfaces and NAND flash memory array. ARM (Advanced RISC Machines) is chosen as main control chip, which controls the timing of data transmission by sending specific instructions. When data is written to the SSD from USB 3.0, SATA, PCIe interfaces, data is stored into an asynchronous FIFO (AFIFO) under the control of DMA (Direct Memory Access) through the data bus. After encoded in the ECC module, data is programmed into flash memory through the NFC (NAND flash controller).

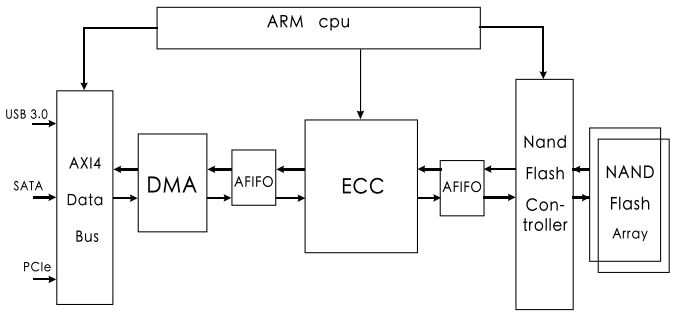

Fig. 1: SSD system configuration

3.2. Multi-rate polar codes

Due to the special construction of NAND flash memory, the data size of read and program processes must be aligned with the page size, which means that at least one page of data is read out during one read process. Hence the block length of polar codes is also determined based on the page size.

In our work, we assume a page size of 4 KBytes and choose a block length of 8192 bits for our polar codes. Thus, each page stores four codewords. The primary reason for our choice of the block length is hardware implementation. Since we implement the ECC on FPGA (more details in Section 4), the block length is also limited by the capacity of the FPGA board. Of course, our work can be extended to polar codes of longer block lengths, such as 2^{15}.

NAND flash memory has a finite number of P/E cycles. Beyond this limit, due to oxide wear-out, the electrons may leak, and hence the information stored in memory cells is no longer reliable. Therefore, the crossover probability is different over the lifetime of SSDs. To ensure the reliability of the store information as well as to reduce redundancy, ideally different code rates should be used in different life stages of SSDs. In early stages, the crossover probability is lower, and hence polar codes with higher code rates can meet the error correction requirements. When the number of P/E cycles increases, correction ability of high rate codes is no longer enough, and instead lower rate codes are necessary in late stages.

To generate our polar codes of length 8192, we use the degrading merge algorithm in [7], which can efficiently construct polar codes. To ensure that the memory requirement does not grow exponentially with code length, this construction method works for any specified memory limit. This algorithm consists of degrading quantization and upgrading quantization, and a parameter μ denotes the approximations of the quantization process. We implement this algorithm and choose $\mu = 256$ to generate our codes. For the binary symmetric channel, we use a set of crossover probabilities measured from 16nm NAND SLC flash memory at different program/erase cycles [8]. When the number of program/erase cycles ranges from 5 to 10000, the crossover probability increases from 1.263×10^{-4} to 6.35×10^{-4}. We picked the crossover probability for 6000 program/erase cycles, which is 0.0004, and use this crossover probability to generate polar codes of different rates. The code rate is a tradeoff between error correction performance and efficiency. Lower rate codes have better error correction capability but need more redundancy, while higher rate codes are just the opposite. Thus, multi-rate codes are necessary over the life time of SSDs. Using the crossover probability 0.0004, we have obtained three polar codes, (8192, 7373), (8192, 7618), and (8192, 7700) codes, which correspond to rates of 0.9, 0.93, and 0.94, respectively. We note that all three have rates at least 0.9, which is often assumed for storage systems.

4. HARDWARE IMPLEMENTATIONS AND PERFORMANCE EVALUATION

4.1. Hardware implementations of Encoder and Decoder

To support multi-rate polar codes, the encoder and decoder need to work for polar codes with different rates.

The encoding process of polar codes is carried out as in Eq. (1). This encoding process can be implemented by recursively using a basic unit of two operations: $u_1 \oplus u_2 = x_1; u_2 = x_2$. This encoding process is the same for any code rate with the same length. The details of the encoder are omitted henceforth.

Our decoder is based on a high throughput CA-SCL decoder proposed in [9]. The decoder in [9] performs list decoding on a binary tree, whose nodes characterize the structure of polar codes. In the hardware implementation of the decoder in [9], the information about the nodes of the tree and the decoding schedule are stored in the ROMs. For polar codes with different rates but the same length, the hardware implementation of our decoder is the same, and the only difference is the node information and decoding schedule stored in the ROMs. Hence, the node information and decoding schedule of all three codes are stored in the ROMs, and they are retrieved accordingly.

We implemented the encoder and decoder in FPGA using a Virtex-7 FPGA VC709 board. We chose a list size of 4 for the CA-SCL decoder. There are two clock domains in our design: the clock rate in the encoder is 166.64 MHz, while the clock rate in the decoder is 41.66 MHz. Based on this clock rate, the throughput of the codec is 391Mbps. The hardware utilized by the encoder and decoder are detailed in Table 1.

Table 1: FPGA implementation results of encoder and decoder in our design

	encoder	CA-SCL decoder	resource usage
LUTs	13,273	161,469	40.3%
FFs	17,758	28,784	5.37%
BRAMs	0	5,823	11.01%

4.2. FPGA Emulation Platform

To evaluate the error performance of polar codes, especially at very low bit error rates, numerical simulations are too time consuming. Hence, we adapt an FPGA emulation platform we developed in prior work [10]. One difference between the FPGA platform in [10] and this work is the FPGA board. The platform in [10] uses a Genesys-2 Kintex-7 FPGA development board with a Xilinx Kintex-7™ FPGA (XC7K325TFFG900-2), whereas the platform in this work uses a Xilinx Virtex-7 FPGA development board with a Xilinx Virtex-7™ VC709 (XC7VX690TFFG1761-2). This switch was made because the latter offers more hardware resources. The FPGA platform in this work also needs to support multi-rate polar codes by reconfiguring its modules. Next we describe the platform and focus on the reconfiguration of different modules.

Our FPGA platform consists of CPU controller, channel module, polar code sequence generator, encoder, decoder, and error bit check modules, as shown in Fig. 2.

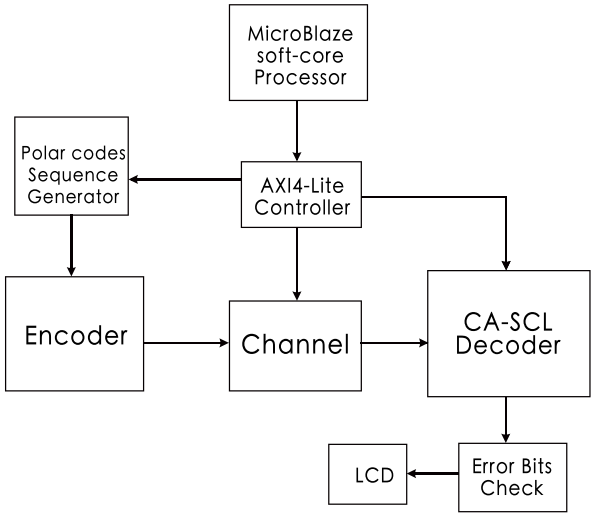

Fig. 2: Block diagram of our FPGA emulation platform

In our design, we use MicroBlaze as main control chip embedded in FPGA, which is connected with other peripheral modules by an AXI4 bus. The AXI4 bus is used to control the peripheral modules. When we connect MicroBlaze and control registers of peripheral modules by the AXI-lite protocol, these registers will be mapped to system memory. Values of control registers of peripheral modules can be easily rewritten by the CPU by altering the corresponding memory. In our performance evaluation, the crossover probability in the channel module and reconfiguration of the decoder for multi-rate polar codes are all changed by using the AXI-lite protocol.

The polar code sequence generator module consists of two parts, a random number generator (RNG) and a random-access memory (RAM). For an (N, K) polar code generated by our design, we use an N-bit binary code structure sequence to represent the positions of frozen bits. For example, a code structure sequence $(1, 0, 0, 1, \cdots)$ means that bits 2 and 3 are frozen bits and bits 1 and 4 are information bits. The information bits used in our emulation are generated by the RNG. We store each code structure sequence into a RAM, whose address ranges from 0 to $N-1$ with 1 bit depth. This process is controlled by software through the AXI-lite protocol. Then the information bit generated by the RNG is stored into the addresses whose values equal 1. After K random information bits are produced, we get an N-bit polar code sequence. Polar code sequence for multi-rate polar codes with the same length can be generated by altering the frozen bit positions

stored in the RAM via the AXI-lite interface.

Our FPGA emulation platform supports two channel models, additive white Gaussian noise (AWGN) channel and binary symmetric channel (BSC). In SSD applications, the binary symmetric channel is mainly used to simulate NAND flash memory, and p denotes the cell crossover probability of flash memory. That is, a binary bit gets flipped with probability p, and remain unchanged with probability $1-p$. We use a codeword sequence x_0^{N-1} as the input for the channel. Given an output sequence $y_0^{N-1}(y_0, y_1, \cdots, y_{N-1})$ by the channel, its corresponding LLR sequence is calculated by

$$LLR_i = \begin{cases} \log\frac{1-p}{p} & \text{if} \quad y_i = 1 \\ \log\frac{p}{1-p} & \text{if} \quad y_i = 0, \end{cases} \tag{2}$$

where $i = 0, 1, \cdots, N-1$. These LLRs are the input of the following decoding process. The registers storing the crossover probability of channel are connected with the AXI-lite interface. Thus, the performance for different P/E cycles of SSDs can be simulated by altering the values of these registers.

4.3. Simulation Results

For the three polar codes, (8192, 7373), (8192, 7618), and (8192, 7700) codes, their bit error rates (BERs) and frame error rates (FERs) are compared in Fig. 3. As we see in Fig. 3, a higher rate code has worse performance than a lower rate code. Each point in Fig. 3 is obtained by accumulating at least 100 frames in error. The emulation process is somewhat time-consuming. For instance, for the rate-0.94 code over the BSC channel with a crossover probability 0.00015, to accumulate 100 error frames it needs to decode a total of 7,504,431,955 frames. Since it takes 3,485 clock cycles to decode each frame, with a clock rate of 41.66 MHz in the decoder, this one data point needs at least 174 hours. In practice, more FPGA boards would reduce the emulation time. Due to the long time needed to accumulate enough frames in error, we focus on the code with the highest rate, the code with rate 0.94, for very low bit error rates, to examine its error floor behavior. Fig. 3 shows that the code with rate 0.94 has no error floor even when the BER gets as low as 10^{-12}. Since the performance of the two codes with rate 0.9 and 0.93 is probably no worse than that of the code with rate 0.94, we conjecture that the two codes with lower rates have no error floor when the BER gets as low as 10^{-12}. We will verify this in our future work.

In Fig. 3, we also compare the bit error rate of our polar code with rate 0.94 with that of a (35840, 33792) LDPC code for SSDs, which is generated from [11]. The two codes have roughly the same rate, but the block length of the LDPC code is much longer. Since the error performance of polar codes improves with their block length in general, the polar code is somewhat at a disadvantage. We see that the LDPC code shows an error floor, starting at bit error rate of 10^{-10}. In contrast, the polar code shows no sign of an error floor when the bit error rate goes down to 10^{-12}. This demonstrates the advantage of polar codes in error floor, which is significant for hard drives due to their stringent requirement on bit error rates.

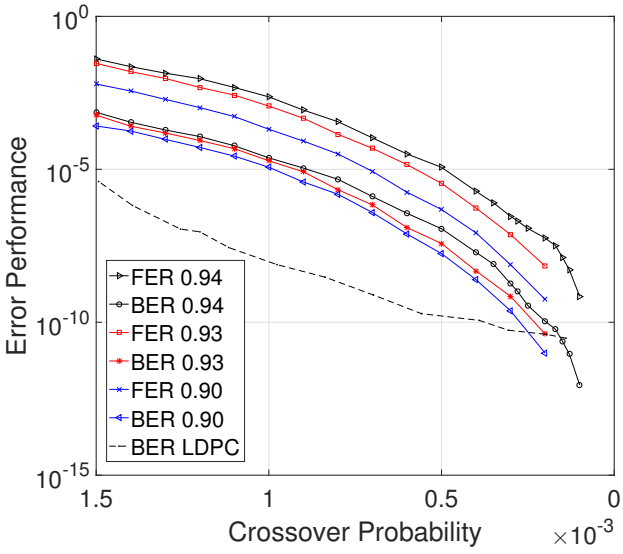

Fig. 3: Error performance of polar codes of different rates and a (35840, 33792) LDPC code

5. CONCLUSION AND FUTURE WORK

In this paper, we first design multi-rate polar codes for SSDs, and then implement encoder and decoder that simultaneously support multiple rates in FPGA. Finally, we use an FPGA emulation platform to evaluate the error performance of our polar codes, and examine their error floor behavior. Our results show that our polar codes have good error floor behavior.

The error performance of the polar codes can be improved by using more sophisticated decoders, such as the CA-SCL decoder with a greater list size. Of course, using greater block length will likely lead to performance improvement. Both approaches will impact the decoder complexity and hence its hardware implementation. We will explore these in our future work. Also, we have used the binary symmetric channel to model NAND flash memory so far in our work. In our future work, we will use actual NAND flash memory in our platform for a more realistic performance evaluation.

6. REFERENCES

[1] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Tong Zhang, Xiaodong Zhang, and Nanning Zheng, "LDPC-in-SSD: making advanced error correction codes work effectively in solid state drives," in *Usenix Conference on File and Storage Technologies*, 2013, pp. 243–256.

[2] Erdal Arikan, "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, 2009.

[3] Ali Eslami and Hossein Pishro-Nik, "On finite-length performance of polar codes: Stopping sets, error floor, and concatenated design," *IEEE Transactions on Communications*, vol. 61, no. 3, pp. 919–929, 2012.

[4] Yue Li, Hakim Alhussien, Erich F Haratsch, and Anxiao Andrew Jiang, "A study of polar codes for mlc nand flash memories," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 608–612.

[5] Ido Tal and Alexander Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[6] Kai Niu and Kai Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[7] Irina Tal and Alexander Vardy, "How to construct polar codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, 2013.

[8] Private Communication with Yue Li, ," .

[9] J. Lin, C. Xiong, and Z. Yan, "A high throughput list decoder architecture for polar codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2378–2391, 2016.

[10] Chenrong Xiong, Yi Zhong, Chun Zhang, and Zhiyuan Yan, "An FPGA emulation platform for polar codes," in *2016 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 1–6.

[11] Yishan Zhang, Chun Zhang, Zhiyuan Yan, Shuang Chen, and Hanjun Jiang, "A high-throughput multi-rate LDPC decoder for error correction of solid-state drives," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2015, pp. 1–6.