# An Efficient Implementation of Multi-interface SSD Controller on SoC

Yifei Niu, Hai Cheng, Songyan Liu(✉), Huan Liu, Xiaowen Wang, and Yanlin Chen

Heilongjiang University, Harbin 150076, China
{2171304, 2171309, 2171414, 2161359}@s.hlju.edu.cn,
{2002066, liusongyan}@hlju.edu.cn

**Abstract.** This paper designs a high-performance multi-interface SSD controller built on Xilinx SoC. An efficient firmware is also implemented, which is elaborate to cooperate with the hardware. Parallelism techniques, such as plane-level, die-level, chip-level and channel-level, are used for improving performance. The system has hard real-time performance of sequential writing, with the minimum bad block management and wear-leveling policy to balance performance and lifetime. A transparent encryption is proposed to guarantee high security storage, that is, connecting an AES-256 core and a RAID core with DMA engine in series. Performance evaluation of physical hardware shows that writing speed can exceed 100 MiB/sec for every logical channel which combines 8 NAND Flash chips.

**Keywords:** SSD · Pipeline · SoC · NAND Flash · Security · AES

## 1 Introduction

Solid state disk (SSD) has been favored not only by consumer electronics but also by enterprise, industry and military due to its high performance. The design of SSD needs to be extensible and optimizable for domain specific requirements [1]. Moreover, NAND flash needs to be carefully managed to extend lifetime and improve performance, which is the challenges of designing high-performance SSD for a special purpose [2]. System on Chip (SoC) is an excellent solution for SSD controller due to its well extensibility and programmability.

We will focus on two points in this SSD system. One is to design a high-performance SSD which gives full play to modern NAND Flash parallelism characters. There are many discussions about improving SSD performance by using various level parallel resources in the NAND Flash. Yan [3] and Kao [4] exploits to maximize effect of parallelism technologies by out-of-order commands schedule. Mao [5] proposes a new scheduler and exploits the high-level request characteristics and low-level parallelism of flash chips. Some research also analyze and contrast the performance and scalability of parallelism technologies in various levels [6]. For the architecture of this paper, processor and FPGA both work on different levels of parallelism resources to play their own advantages.

The other is to enhance the security of data. Transparent AES and RAID functions are built-in mechanisms to achieve this. At a more basic level, enhanced ECC, wear


S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 181–189, 2019.
https://doi.org/10.1007/978-3-030-22971-9_16

leveling and bad block management are implemented to against the restricted endurance of NAND Flash. Park [7] presents a dynamic wear leveling method based on circular queue and a hardware driven 2-channel 4-wafahuiy interleaving model. Chang [8] proposes a hierarchical block management method to reduce RAM space. Other various excellent approaches of this topic have also been proposed by experts.

The rest of this paper is organized as follows. System hardware architecture is described in Sect. 2. Software techniques such as bad block management and pipeline operations are detailed in Sect. 3. In Sect. 4, we analyze the performance under different configurations. Finally, Sect. 5 concludes this paper.

## 2 System Hardware Architecture

This system is designed to receive data from upper computer and store them to SSD via a high speed GTX bus, and download the data by a download board which provides multi-interface for reading data. The download board is also a test platform for the SSD to avoid connecting to real dedicated upper computer.

### 2.1 SSD Controller

The SSD controller uses Xilinx Kintex-7 series SoC. FPGA can interconnect multiple NAND chips and provide hardware acceleration. 4 ARM Cortex-M1 soft cores are built, which suits for doing the scheduling and management work. To guarantee optimum performance, all MCUs run at the frequency of 100 MHz, and tightly coupled memory (TCM) is used to minimize memory access latency. The architecture of SSD controller can be partitioned as Fig. 1, contents are described on below.

- GTX Bus Interface. It uses Xilinx IP core to implement a high-speed differential IO and a scheme of 8b/10b coding. Both data and commands transferred on the bus are driven by DMA.
- Master Control MCU. It is responsible for bad block and wear-leveling management, parallel dispatching commands and handling upper computer commands. An MRAM is connected to Master Control MCU via general SPI bus. It is used to store some software management and runtime information. And when the power is shut down by accident, it is also used to store some last words to ease power failure recovery when the next boot. Master Control MCU also manages a packet buffer that receives packets from upper computer. The ping-pong access operation is implemented to maximize the bandwidth, details of which are described in the next section.
- NAND Array Controller (NAC). It is responsible for connecting flash chips as an array and translating upper computer commands into low level flash operations. The array means that 8 chips are tightly bound together by FPGA so that they can work in parallel. There are two parts in NAC: The FPGA handles data interface which is compliant with the ONFI 3.0 standard and provides ECC correction. An ARM Cortex-M1 core schedules flash operations, notifies DMA Controller to transfer data and reports status to Message Unit finally. The number of NAC is decided by requirements of capacity and performance.

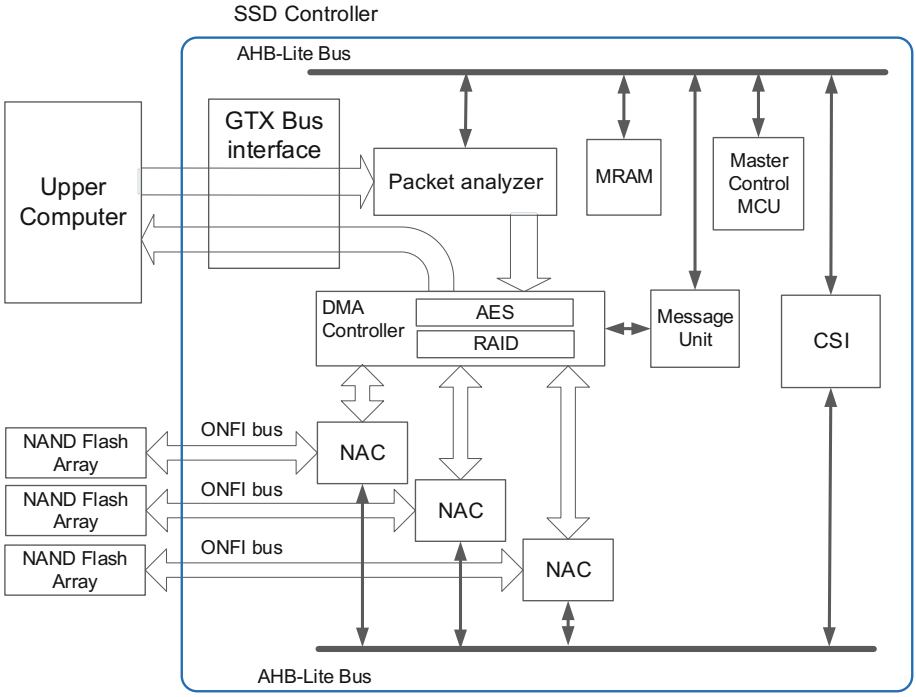


**Fig. 1.** Hardware architecture

- DMA Controller. The DMA controller has two separate channels for sending and receiving data, and provides an interface in the form of a request queue for each NAC. The deep of request queue is 16 levels, so some requests can be buffered. DMA completes most of work on data transfer, improving overall performance and reducing MCU burden. It also provides the transparent RAID and AES function.
- Message Unit. It is a memory pool with synchronization mechanism for exchanging commands and status between master control MCU and NACs. Message Unit can buffer commands in a first-in-first-out way, which contributes to pipeline and parallel commands dispatching.

### 2.2 Inside Pipelined AES Core

The AES algorithm is widely used for protecting the user data, which has the advantage of good safety, high efficiency, practicability and high flexibility [9]. AES can also contribute to suppress the bit error rate in MLC/TLC NAND Flash due to its homogeneous function [10]. To provide transparent encryption in the controller, we propose an implementation to connect an AES-256 core and a RAID core with DMA engine in series. In this scheme, when data are transmitted from packet buffer to NAC, encryption and RAID are also done with no performance loss but a little delay. To achieve this, the AES engine is implemented as pipeline structure. We choose CTR (CounTeR) block

cipher mode of operation, so only encryption need to be implemented. It is better for saving resources to build the pipeline. The DMA controller with inside AES and RAID engine is illustrated in Fig. 2.

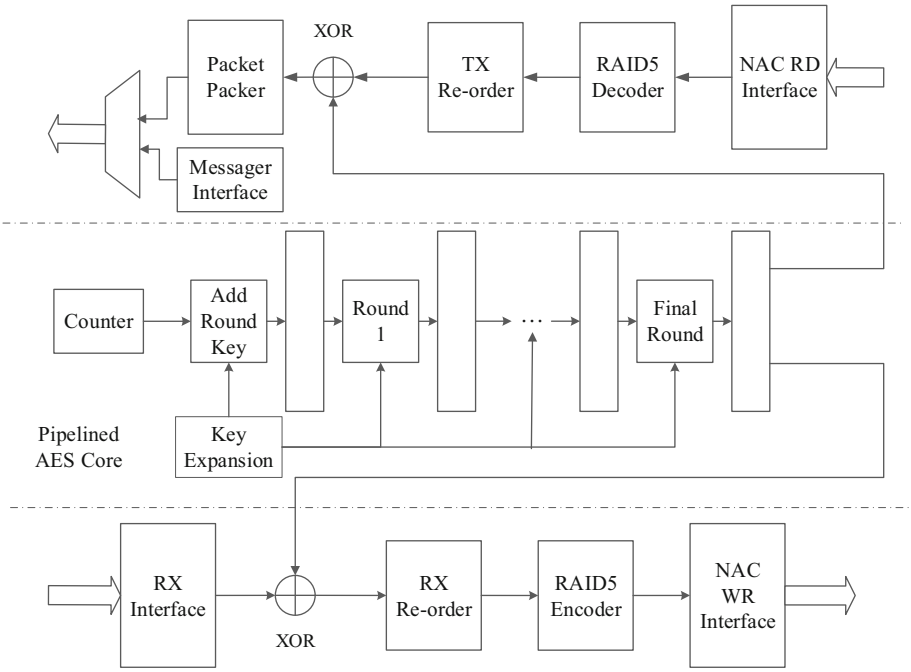


**Fig. 2.** Pipelined AES implementation in DMA controller

As shown in Fig. 2, all 14 rounds in AES-256 are expand into independent units, and cascaded as a 10-stage pipeline. Each independent unit takes only one cycle. For each round, we use 2 S-box in block RAM, one is for encryption process, the other is for key expansion. The transfer bandwidth of DMA controller is 128-bit, which matches AES cipher block size.

### 2.3 Download Board

Download board uses a Xilinx Zynq-7000 series SoC which integrates a dual-core ARM Cortex-A9 processor and FPGAs. A GTX interface is also equipped for simulating upper computer of SSD, so download board can also be a test platform when there is no upper computer. General USB 3.0 and gigabit Ethernet interface are equipped for reading data in different usage scenarios.

# 3 Software Design

## 3.1 Bad Block Management

### 3.1.1 Organization of Bad Block Table

As we all know, there may be bad blocks even factory, or may arise during the lifetime of the device. A bad block table is usually built for managing them [11]. For a new device, firmware will do the initialization of bad block checking. Good and bad blocks marked by the manufacturer will be separately. Each entry of the bad block table contains physical addresses of channel, logical unit (LUN) and block. And the index of entry becomes the logical block address (LBA). LBA is always used for communicating with the download board. A bad block table is divided into 3 parts. The top of it is user zone, which saves good blocks in the initial checking. Reserved zone also saves good blocks under the user zone. The different point is that blocks in the reserved zone are used to replace bad blocks, and they are not addressed by user-visible capacity. The third part is bad block zone, partly overlapping with reserved zone, in fact, it grows up from the bottom of the reserved zone. There is not just one bad block table, the number of table is corresponding to parallel resources of the system. For example, if there are 3 NAND flash arrays and 2 LUNs on a chip, there should be 6 tables.

### 3.1.2 Block Replacement Without Speed Loss

When a bad block appears, a good block must be selected to replace it immediately for later operations. And written data in bad block must be kept and transferred to a new block at the suitable time. Since this system is implemented to have a hard real-time writing performance, the internal data transfer which takes a long time must not happen when data is inputting. The solution is to divide block replacement and data transfer into two parts (top half and bottom half). The top half will be executed immediately when bad block appears, and the bottom half will be executed when system synchronization command is received.

The Fig. 3 shows the work method of block replacement, sN means the state of bad block table, B_N means Nth bad block, R_N means Nth reserved good block, and C_N means Nth temporary block for transferring data.

1. Top half (on the fly)
   - s1. Bad block B_1 is founded.
   - s2. Getting the good block R_1 from bottom of the reserved zone (close to bad block zone), and swapping the address of R_1 and B_1 in bad block table. If other bad blocks are founded, s1 and s2 will repeat in order.
2. Bottom half (on synchronization)
   - s3. Getting a good block C_1 from bottom of the reserved zone. Firstly, copying the written data in corresponding bad block B_1 to C_1, then copying remain data in R_1 to C_1. If the process of copying from B_1 to C_1 is faulty, it means block C_1 is also bad. Moving up the top pointer of bad block zone to put C_1 into bad block zone, and getting another one from the reserved zone.
   - s4. Swapping the address of R_1 and C_1 in bad block table. Erasing the block R_1 to be used as a reserved block.

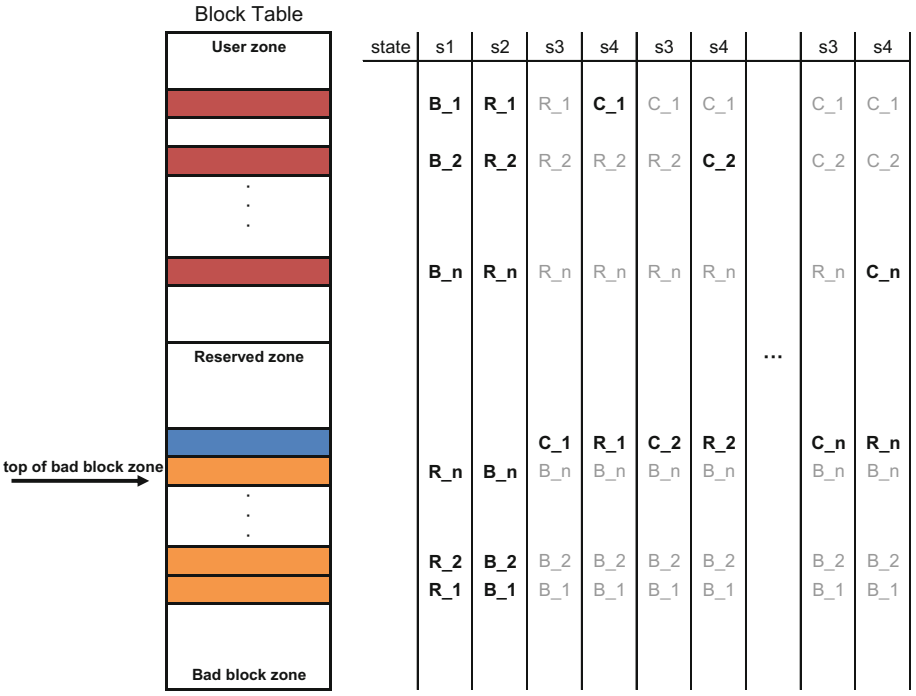


**Fig. 3.** Bad block replacement algorithm

If there are multiple bad blocks in previous steps, repeating s3 and s4 until all bad blocks are handled.

The written data in bad block B_N cannot be copied directly into reserved block R_N, because the programming order of pages in a block is limited to have a sequential order in many NAND flash chips.

## 3.2 Pipeline Operation

### 3.2.1 Die Interleaving

Every LUN (Die) which can independently execute commands and report status is considered as a parallelizable resource. So the MCU does not need to wait for completion of one die, it can send commands to another die until all dies in all channels are accessed, the MCU must wait for completion of the die that has been issued the command this time.

As shown in Fig. 4, the time of writing data is mostly spent on internal program operations of NAND chip, the time of I/O transfer is short and should be counted. Obviously, for writing bandwidth, internal program operations will be the bottleneck, and the time of MCU scheduling and DMA transferring can be hidden behind that. To estimate the performance of this pipeline, we can imagine splicing program operation on one die at the synchronization point, and we can see all dies are paralleled with only

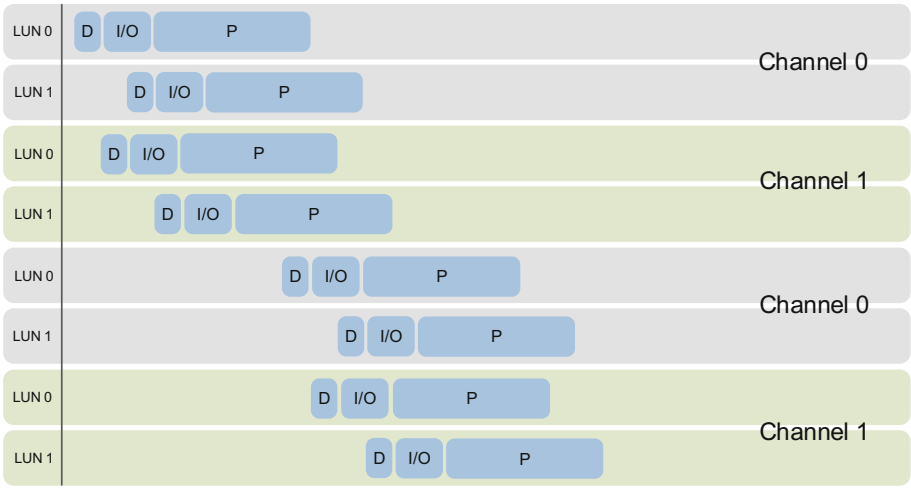


**Fig. 4.** Die-interleaving

a short delay for I/O overhead in one time. So, if we only consider the contribution of die interleaving, the performance can be expressed in Eq. (1).

$$P_{time\to\infty} = N_{dies} \times \frac{C_{page} \times N_{chips\_per\_channel}}{T_{DMA} + T_{I/O} + T_{program}} (MB/\text{ sec}) \tag{1}$$

It should be noted that this is an idealized analysis, an additional time should be counted because the difference of time of each program can cause a gap on the synchronization point.

### 3.2.2 Cached Write/Read Operation

To further improve the performance, internal buffer of NAND chip can be used. Vendors usually called it cache register and they provide special instructions to do cached write and read operations. In program operation, firstly, the data go through cache register. The process of copying data from cache register to NAND is handled by chip automatically. Otherwise, data are read out to cache register and are automatically sent out of the chip in reading operation. This means that MCU does not need to wait for completion of the program operation before issuing a new command to the same die, it only waits for cache register that is available to receive new data. So the $T_{program}$ in formula (1) could be shortened. The local buffer in NAND array controller (NAC) is equivalent to provide a 1-stage pipeline, and cache register increase it to 2 stages.

### 3.2.3 Multi-plane Operation

Planes in a LUN are also parallel resources but with more limits [12]. Each plane is equipped with cache register. And controller input data to every addressed plane cache register and execute a command to do the program operation for all planes together in parallel. That means the program time of planes can be pared down to that of one plane,

but the I/O overhead does not be reduced. The performance can be significantly improved with multi-plane operation, but correspondingly, the number of bad block table should be increased to adapt to more parallel resources. The performance with multi-plane operation can be expressed in the formula (2).

$$P_{time\to\infty} = N_{dies} \times N_{planes} \times \frac{C_{page} \times N_{chips\_per\_channel}}{N_{planes} \times (T_{DMA} + T_{I/O} + T_{program})} (MB/\sec) \quad (2)$$

## 4 Performance Evaluation

In this section, we will take real implementation of this system as an example to analyze the performance with different techniques and configurations. The NAND flash chip is MT29F256G08CEEAB manufactured by Micron, which has 2 planes in a LUN (die) and 2 LUNs in a chip. The size of page is 16 KiB, but only 14 KiB is used for storing data due to the RAID function. The typical time of program page and reading page are 1.6 ms and 115 us, and the time of I/O transfer is about 300 us in Single Data Rate (SDR) mode at 50 MHz. 8 NAND flash chips combine together to work in parallel by hardware, called a logical channel.

For only using die-interleaving technique, for example, we can substitute below parameters into formula (1), the writing performance of one channel, that is, $2 \times 14\,\text{KB} \times 8 \div (0.3 + 1.6)\,\text{ms} \approx 117.9\,\text{MB/s}$. In our solution, the 8 NAND flash chips do program operation in parallel, so the maximum program time of these chips will be used as the actual program time. Results of different configurations are illustrated in Fig. 5.

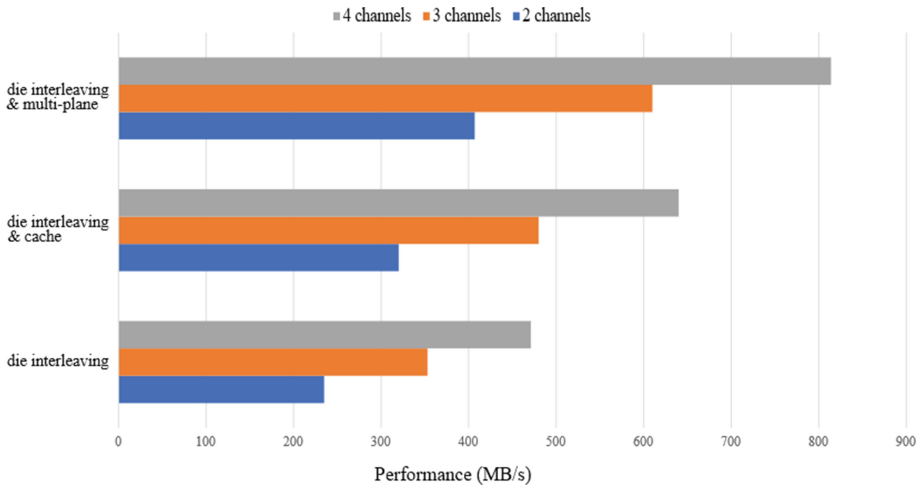


**Fig. 5.** Performance of using various channels and techniques

## 5 Conclusion

In this paper, we present a high performance SSD controller based on SoC, it provides multiple interfaces, low power consumption, high reliability and security. The well programmability of FPGA helps us to build AES and RAID functions inside to give well security of data. We use various parallel and pipeline techniques to enhance the performance.

For future work, 3D NAND Flash are considered to incorporate into our system since their high storage density. But characters of those architectures like less endurance cycle, longer program time and big block size will be challenging issues to flash management and power-off recovery.

## References

1. Alsalibi, A.I., Mittal, S., Al-Betar, M.A., Sumari, P.B.: A survey of techniques for architecting SLC/MLC/TLC hybrid flash memory–based SSDs. Concurrency Comput. Pract. Experience **30**(6), 4420–4421 (2018)
2. Mielke, N.R., Frickey, R.E., Kalastirsky, I., Quan, M., Ustinov, D., Vasudevan, V.J.: A survey of techniques for architecting SLC/MLC/TLC hybrid flash memory–based SSDs. Concurrency Comput. Pract. Experience **105**(9), 19–53 (2017)
3. Yan, W., Liu, Y., Wang, X.: An efficient parallel executing command scheduler for nand flash storage systems. In: 4th International Conference on Electronics Information and Emergency Communication (ICEIEC), pp. 20–24. IEEE, Beijing (2013)
4. Kao, Y.H., Huang, J.D.: High-performance nand flash controller exploiting parallel out-of-order command execution. In: Proceedings of 2010 International Symposium on VLSI Design, Automation and Test, pp. 160–163. IEEE, Taiwan (2010)
5. Mao, B., Wu, S., Duan, L.: Improving the SSD performance by exploiting request characteristics and internal parallelism. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(2), 472–473 (2018)
6. Tavakkol, A., Arjomand, M., Sarbazi-Azad, H.: Design for scalability in enterprise SSDs. In: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, pp. 417–430. ACM, Edmonton (2014)
7. Park, C., et al.: A high performance controller for nand flash-based solid state disk (NSSD). In: Non-Volatile Semiconductor Memory Workshop, pp. 17–20. IEEE (2006)
8. Chang, L.P., Kuo, T.W.: Efficient management for large-scale flash-memory storage systems resource conservation. ACM Trans. Storage (TOS) **1**(4), 381–418 (2005)
9. Xie, M., Li, S., Glova, A.O., Xie, Y., Hu, J.: Securing emerging nonvolatile main memory with fast and energy-efficient AES in-memory implementation. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**(11), 2443–2455 (2018)
10. Fan, L., Luo, J., Liu, H., Geng, X.: Data security concurrent with homogeneous by AES algorithm in SSD controller. IEICE Electron. Express **11**(13), 20140535 (2014)
11. Pan, Y., Li, Y., Zhang, H., Xu, Y.: Lifetime-aware FTL to improve the lifetime and performance of solid-state drives. Future Gener. Comput. Syst. **93**(4), 138–163 (2019)
12. Winata, Y., Kim, S., Shin, I.: Enhancing internal parallelism of solid-state drives while balancing write loads across dies. Electron. Lett. **51**(24), 1978–1980 (2015)