OpenSSD Platform Simulator to Reduce SSD Firmware Test Time

Taedong Jung, Yongmyoung Lee, Ilhoon Shin

Department of Electronic Engineering, Seoul National University of Science and Technology, South Korea
Building 3,206, Seoul 139-743, South Korea
ilhoon.shin@snut.ac.kr

Abstract: Recently, NAND flash memory is widely used as storage media in various devices such as SD card, USB memory, and solid state drives (SSDs). NAND flash memory has different characteristics from hard disks, and thereby a firmware called flash translation layer (FTL) should be deployed to use traditional file systems on NAND flash memory. Because FTL majorly determines performance and stability of NAND-based block devices, various FTL schemes have been developed. However, their efficiency has been evaluated via simulation not on a real device, which restricts the reliability of the results. Recently, OpenSSD platform, which allows implementing a new FTL scheme on real NAND devices, was developed. By using it, evaluating the FTL performance on real NAND devices becomes possible. The problem is that it is difficult and time consuming to debug the new FTL on the OpenSSD platform. In order to address this, we design an OpenSSD simulator that helps find the bugs of FTL and reduces the test time. The FTL developed on the simulator can be migrated to the OpenSSD platform without additional porting.
[Taedong Jung, Yongmyoung Lee, Ilhoon Shin. **OpenSSD Platform Simulator to Reduce SSD Firmware Test Time.** *Life Sci J* 2014;11(7):564-568]. (ISSN:1097-8135). http://www.lifesciencesite.com. 77

Keywords: Bug detection**,** OpenSSD simulator, Flash translation layer, NAND flash memory

1. Introduction

SSD has strong points like the fast speed and safety, contrast to HDD. SSD is guaranteed fast I/O speed via the parallel activity of multiple NAND flash memory. NAND flash memory consists of multiple storage media called as block. One block consists of many pages which are 64 or 128 units, and so on. NAND flash memory's I/O unit is a page. Also, pages in a block should be sequentially written, and the page cannot be overwritten (Kawaguchi et al., 1995). To overcome these restrictions, a firmware called FTL is deployed within SSD, and SSD can be used like a general disk device. Because FTL majorly determines performance and stability of NAND-based block devices, various FTL schemes have been developed (Gupta et al., 2009) (Lee et al., 2007). However, their efficiency has been evaluated via simulation not on a real device, which restricts the reliability of the results. Recently, OpenSSD platform, which allows implementing a new FTL scheme on real NAND devices, was developed. OpenSSD platform has the same construction of the existing SSD. As seen in Figure 1, it consists of real NAND flash memory and controller, DRAM, SATA Interfere, and so on. By using it, evaluating the FTL performance on real NAND devices becomes possible.

There are two ways for debugging FTL in OpenSSD platform. The one method is using RV ICE and RVDS. The second is using GNU compiler. To the reason of character of hardware, the time for I/O at NAND flash memory is evaluated intactly, all of two ways spend lots of time in debugging. The case of logical error that can be detected in a long time after beginning of real FTL evaluation spends lots of time for debugging, these cases makes hard for OpenSSD platform. For solving these problems, if OpenSSD simulator is developed by detecting the logical error of FTL code in a way of software, the time for development can be decreased. Because I/O like real hardware is not operated, it can be verified quickly at the same test environment. After verifying the logical error of FTL via OpenSSD simulator, and then if there is a test in OpenSSD platform, it could be efficient because the time for debugging is decreased very much.

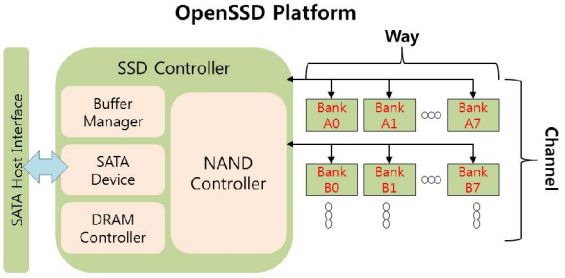

Figure 1. Internal structure of the OpenSSD Platform

This paper will explain about the necessity of simulator in chapter 2, the construction of simulator in chapter 3. In chapter 4, there will be verifying the effectiveness of extracting the error via the experiment, and the conclusion will be derived in chapter 5.

2. Necessity of OpenSSD Simulator

The difficult point at the progress of development of FTL in OpenSSD platform is spending lots of time for finding the logical error. These logical errors can be categorized into two ways in large scale. At first, the error which is founded after the long time has been passed by at the beginning of verification. This error is not detected before the operation of Garbage collection which deletes the unnecessary data is operating. Secondly, even though it is the mistaken I/O, it can be hard to detect in the way of hardware. In the case of overwriting the space which has already the data and improperly access at the region of DRAM, the board can't be recognizing its problem and operating the mistaken I/O.

2.1. Errors that can be detected after a long time

Free Block Count Variable is the variable which administers the free block unit for saving the data in NAND flash memory. At the initialization of NAND flash memory, it can be categorized into the meta-block which saves the meta-data and data-block which saves the data. These category's difference is depend of FTL policy, thus Free Block Count's variable is depend on the developer's development policy for its difference. If all of the blocks are exhausted, and Free Block Count's variable number will be 0, then Garbage Collection should be operating for getting the storage place of its new data. However, if Free Block Count's variable is improperly initialized because of the developer's mistake, it can't be possible to detect the error until all of blocks are exhausted. If Free Block Count variable is initialized less than its normal state; the block which is not used about its difference will be emerging. In contrast to its situation, it is initialized more than its normal state; it will try to find the free block in the situation which there is not unusable block. In this case, the problem of falling into the infinite loop of firmware is emerging because it tries to find the new free block while it checks all of blocks. It can be possible to check the mistaken initialization of Free Block Count's variable with using of ASSERT function of inside of firmware which is the existing debugging method, but it can be spending lots of time to detect the error. The reason for this is that it is operated actually the I/O of the hardware way by OpenSSD.

OpenSSD platform has lots of logical errors that is spending lots of time to detect like Free Block Count variable because of I/O of the hardware way. Because Garbage collection will be operated after NAND flash memory's whole space is exhausted, the first operation is emerging normally after several hours. Therefore, it costs lots of time to detect the logical error with using only OpenSSD platform. In case of OpensSSD platform, it can be possible to detect the logical error at the same test environment in a few minutes.

2.2. Errors that is hard to be detected

There is the situation that can't be recognizing the error in case of I/O that is the problem at the time of FTL development. The invasion of DRAM region is the representative example. At OpenSSD platform, the meta-data of FTL is distributing DRAM region like Figure 2 and saving each region. The improperly accessing is possible about the distributed region; however, the hardware is operating without problem. This is possible because API which is related to memory brings the data of ordered addresses without verification. The data that is saved on DRAM is hard to progress the specific verification according to the situation because it is the randomly saved meta-data. For example, suppose that the case of accessing to DRAM for its changing the number of Valid Count of figure 2. If the developer sets up 16Byte to each entry but if access 32Bytes to the mistaken function and it can access to the excessive region over the real region. If some data is saved on the mistaken region, the problem that its data is improperly recognized to Valid Count is emerging.

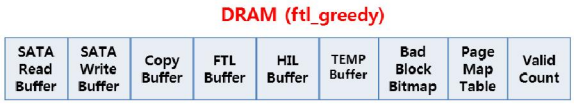

Figure 2. Distribution of DRAM region

Even though it is the problematic situation in reality, various situations which are not recognized are around. If the sequential writing is broken on the block, in case of I/O of block or page that is out of region, the situation of operating the write with the overlapping on the already written page, these cases are the relevant cases.

3. Design of OpenSSD Simulator

OpenSSD simulator is constructed for verifying FTL code of OpenSSD platform. Therefore, Low-Level device Driver API which is ordering to NAND flash memory and Memory Utility API which is ordering to DRAM, these two things are constructed like a Figure 3. This is the necessary API for expressing the I/O of OpenSSD platform in the way of software. However, OpenSSD simulator can't replace to all of the function of OpenSSD platform. These are the function of Power-Off Recovery and Bad Block Management and interrupt. The better simulator can be expected if its part is solved with the part that is hard to realize in the way of software.

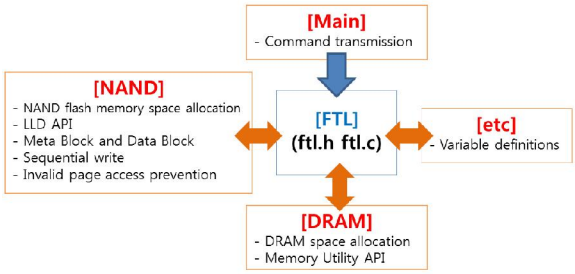

Figure 3. Structure of OpenSSD Simulator

The virtual NAND flash memory at simulator is distributed into the data-block and meta-block. Data block is the place where saves the real data, however the simulator is not needed these entire place. Because the real data is not related to logical error, and the amount of memory is limited to allot in OS. Data block is used to verify the logical error instead of real data that is saved because I/O order is saved. However, all of the meta-block is necessary. In this place the necessary order is saved like a mapping table. In conclusion, it can be using by distributing into the meta-block which demands all of the space like a OpenSSD Platform and data-block which is saving I/O order. FTL is distributing like Read/Write/Copy Buffer, FTL Buffer, and ECT for using effectively in DRAM space. The simulator is using 64MB in the inside of its space with the dynamic allotment, and each meta-data's space is allotting on the order of FTL code.

OpenSSD simulator's activity is beginning from passing its order of I/O. The order of I/O is using the pattern of I/O which is collected from the past based on users. Because the transferred order is based on Disk, it is converting to base order of NAND flash memory via FTL. The converted order is saved on virtual NAND flash memory via using Memory Utility API, Low-Level device Driver API. Both Memory Utility API and Low-Level device Driver API, there are added the various codes to verify the logical error. ASSERT function which is to detect the logical error at the simulator has the same operation of ASSERT function of OpenSSD platform. Lots of codes are already applied to check the logical error that is considered in the simulator contrast to OpenSSD platform that is added the appropriate error check by user. Also, the function for checking logical error by user additionally, it can be easily possible to add with using ASSERT function of simulator. The saved order which was doing before is using to check the logical error. As checking the saved order on Data block, it can be possible to verify the error about the sequential writing on the block or pages overwrite. In OpenSSD Platform which is I/O per page is hard to know whether the progress of Partial Programming is right or not. But, the simulator is possible to know the accuracy of operation via checking the order of data block.

4. Experiment

FTL_GREEDY code which is sample of FTL of OpenSSD platform is using at the experiment of OpenSSD simulator. Some part of FTL code is necessary to delete because there are the functions that is not provided by simulator like a Power of recovery management function and Bad Block management function, etc. These things are like this.

- Static void build_bad_blk_list(void)
- Static BOOL32 is_bad_block(UINT32 const bank, UINT32 const vblk_offset)
- Static void logging_misc_metadata(void)
- Static void logging_pmap_table(void)
- Static void load_metadata(void)
- Static void load_misc_metadata(void)
- Static void load_pmap_table(void)
- Static void write_format_mark(void)
- Static BOOL32 check_format_mark(void)
- void ftl_isr(void)

Except this, FTL of OpenSSD's verification is possible via the simulator without correction of FTL code.

Table 1. PC configuration

CPU	Intel i5-3210M
RAM	8GB
OS	Windows 7 64bit
Complier	Visual Studio 2010

Table 2. Workload Information

File System	NTFS
Partition size	55GB
Total Write Count	322861057
Total Read Count	309456579

Table 1 is the specification of PC which is operating OpenSSD, and table 2 is the detail explanation of workload for using in verification of code. Workload is the result of collecting the I/O patterns of hard-disk in a long time in PC that is installed of windows XP, NTFS file system. If the simulator operates and finishes the order of operation without logical error of FTL, the result like Figure 4 is emerging. The result writes both of number of order and count of writing, reading and erasing. This information helps the judgment of function of FTL. The effective I/O should be operating for making a better process. Effective I/O work is related to less

amount of number of order of writing, erasing at the time of using the same workload. Therefore, it can be helpful to judge the capability of FTL with using the simulator quickly.

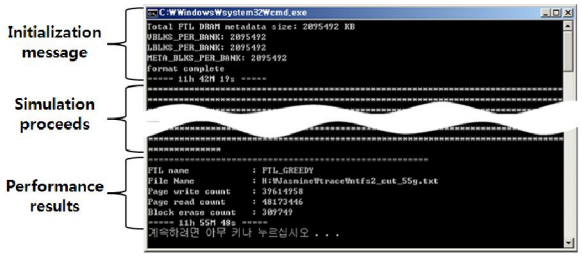

Figure 4. Results of OpenSSD simulator

4.1. Experiment to find logical errors

Free block count variable is the number of block which is not yet used. The garbage collection is operated when its variable number becomes 1 because it means all of blocks is extinguished. Therefore, if the programmer miscalculates this variable and it is initiated, it should be waiting the time as much as all blocks are extinguished, and then it could be possible to confirm the error. The program operates after improperly initialized its free block count variable is added 1 more than its variable for checking the normal operation of simulator. In this case, the emerging of error is detected while the process of simulator program. The detected error shows its number of variable and location of error like Figure 5.

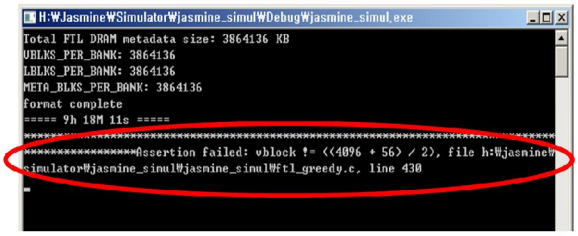

Figure 5. Invalid initialization errors of free block count variables

If OpenSSD platform is used, the time for exhausting all of the blocks should be waited. But OpenSSD simulator is used; you can confirm the result in a few minutes. Also, the logical error can be possible to confirm in the same pattern about ignoring the sequential writing on the block, overwrite the page which has an existed data.

4.2. Experiment of wrong DRAM access

To show DRAM region wrong accessed example in FTL_GREEDY, block write count in the region following valid count region was randomly added.

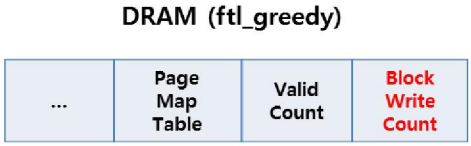

Figure 6. Add Block Write Count region

Valid count metadata entries count is [total bank count (8)] * [total block count per bank], and size per entries is 2bytes (sizeof(UINT16)). The count of added block write count metadata entries is [total bank count (8)]*[total block count per bank], and size per entries is 4bytes (sizeof(UINT32)).

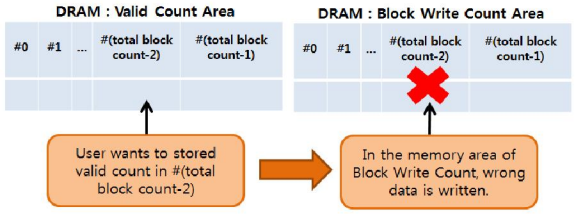

Figure 7. Invading the region of Block Write Count

In order to do experiments, sizeof(UINT32) replaces sizeof(UINT16) for the size of valid count metadata. In this case, valid count per entries is same but valid count metadata region invades the following region which is block write count because of the change in size of valid count metadata. There is a possibility that error is not detected although get_vcount() function for getting valid count data gets block write count. However this kind of error can be easily detected by debug tool in visual studio.

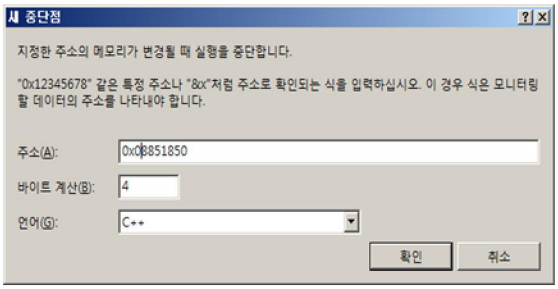

Figure 8. New breakpoint function

The region of DRAM's address is not constant because its program is allotted dynamically. Therefore, at the time of FTL initialization, DRAM region of Block Write Count is derived. After that, it is printing where is the location to access DRAM region via new breakpoint of debugger implement.

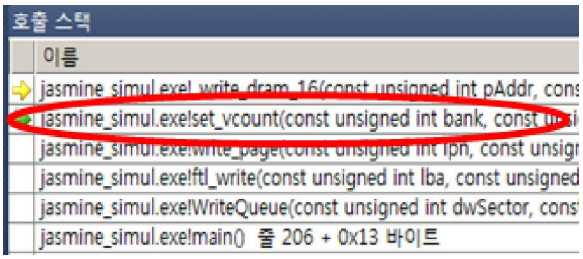

Figure 9. Using the debugger to check access

Via debugger, it should be detected that invasion of Block Write Count region at the function of set_vcount(). From this ability, the fault of set_vcount( ) is corrected.

5. Conclusion

OpenSSD platform is a practical tool to evaluate FTL performance. However, the verification is time consuming and difficult. In order to solve the problem, we designed a simulator for OpenSSD platform, which reduces the verification time and help find the bugs of FTL. Currently, it does not provide power off recovery, bad block management, interrupt control, and package level parallelism capability, which is targets of our future work.

1/7/2014

Acknowledgements:

This study was supported by the Research Program funded by the Seoul National University of Science and Technology.

Corresponding Author:
Dr. Ilhoon Shin
Department of Electronic Engineering
Seoul National University of Science and Technology
NowonGu GongleungDong, Seoul 139-743, South Korea
E-mail: ilhoon.shin@snut.ac.kr

References

1. Kawaguchi A, Nishioka S, Motoda H. A flash-memory based file system. USENIX, USA, 1995.
2. Gupta A, Kim Y, UrgaonKar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. ASPLOS, USA, 2009.
3. Lee S, Park D, Chung T, Lee D, Park S, Song H. A log buffer-based flash translation layer using fully-associative sector translation. ACM Transactions on storage 2007; 6(3).
4. The OpenSSD project. http://www.openssd-project.org/wiki/Jasmine_OpenSSD_Platfrom.