

An Abstract Fault Model for NAND Flash Memory

Ji Hyuck Yun¹, Jin Hyuk Yoon², Eyee Hyun Nam¹, and Sang Lyul Min¹

School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea

¹{jhyun, ehnam, symin}@archi.snu.ac.kr, ²jhyoon@ssrnet.snu.ac.kr

Abstract— We present an abstract fault model for NAND flash memory that describes precisely the effects of various faults during a flash operation. The abstract model is intended to be used to reason about fault-related correctness of key modules of flash memory management software such as a flash translation layer (FTL). We also introduce the concept of “SAO-compliance” to raise awareness about fault-related vulnerabilities of current flash memory management software and to promote much needed research to fix them.

Index Terms— Flash memory, fault model, reliability, flash translation layer (FTL)

I. INTRODUCTION

NAND flash memory [1] [2] has become an increasingly important storage medium, not only in mobile devices but also in PCs and server systems because of its fast random access, low power consumption, small size, and high resistance to shock and vibration.

A NAND flash memory chip is organized into physical blocks, each of which contains a set of pages. It supports three basic operations: read, program, and erase [1] [2]. The read and program operations return the contents of the page and write the supplied data to the page, respectively. The architecture of NAND flash memory does not allow in-place update of data, and all the pages in a block must be erased (i.e., reset to all 0xFF) before any of them can be re-programmed — the erase operation performs this task.

Flash memory is subject to various types of fault. For example, blocks fail over time, which is revealed later by an error returned from an erase or program operation. Another typical fault is power failure that can occur at any time.

Without understanding the effects of faults in a systematic manner, it will be extremely difficult, if not impossible, to reason about the correctness of flash memory management software such as a flash translation layer (FTL) [3] especially when different types of fault are arbitrarily nested. Despite their critical importance, there have been no formal models that describe precisely the effects of various faults in flash memory.

The contributions of this letter are two-fold: (1) we present an abstract fault model for NAND flash memory along with an extension to multi-level cell (MLC) [4] where two or more bits are encoded in a single cell and (2) we introduce the concept of “SAO-compliance” as a criterion of fault resilience of flash management software and discuss its implications in reasoning about fault-related correctness.

II. FAULT CLASSIFICATION

NAND flash memory is subject to both internal and external

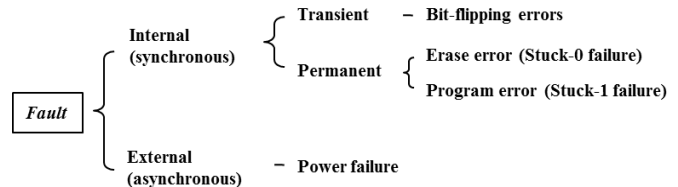


Fig. 1. Classification of faults in NAND flash memory.

faults. Internal faults are synchronous in the sense that they are revealed by the execution of a particular flash operation. An erase operation can report a fault by returning an error code when one or more bits in the target block are stuck at 0, and thus cannot be reset to 0xFF. Similarly, a program operation can report a fault when some bits in the target page are stuck at 1. These two faults are permanent in the sense there is no way of restoring the affected cells to their normal states. Whenever a block exhibits either of these permanent faults, it should be remapped to a reserved spare block and never used in the future. A read operation can also reveal an internal fault called bit-flipping errors due mainly to program/read disturbance and charge leakage [5]. Unlike the previous two types of internal fault that are permanent, this fault is transient, meaning that its effects are removed by a subsequent erase operation on the affected block.

Unlike internal faults, an external fault is not associated with flash operations, and is thus asynchronous with them. Power failure is a typical example of an external fault and it can occur at any time. More importantly, power failure can lead to anomalous behaviors in NAND flash memory. For example, power failure during a program operation can leave the target page in an indeterminate state and in such a case the reliability of the programmed data cannot be guaranteed even when the page appears to be properly programmed [6]. Similarly, an erase operation that has been interrupted by power failure can put the target block into an unreliable state so that even in the case when a page appears to be erased (i.e., contains all 0xFF) the reliability of a subsequent program to the page cannot be guaranteed [6].

Fig. 1 gives a classification of various faults in NAND flash memory discussed in this section.

III. ABSTRACT FAULT MODEL

A. Baseline Model

Here we present a baseline abstract fault model assuming each cell encodes a single bit (an extension to MLC is explained in Section III-C). Before we define the fault model, we make a few assumptions. First, we make the usual assumptions about error control. We assume that the (external)

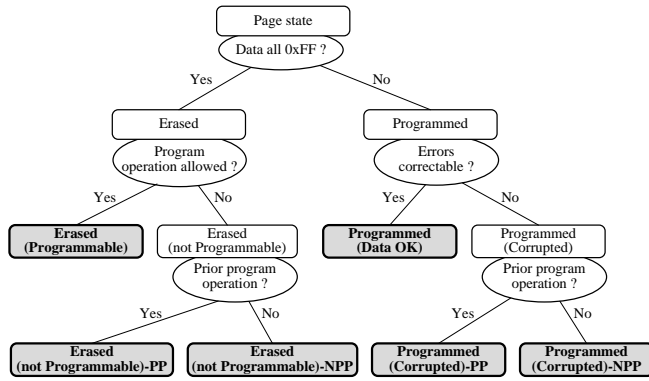


Fig. 2. Classification diagram for page state.

error control logic returns one of the following three symbolic page values for every read operation from a page: **Erased** (i.e., all 0xFF), **Data OK** (errors correctable), or **Corrupted** (errors detectable but uncorrectable). We also assume that the error detection is powerful enough so that it always raises an error on a read from a page containing random data. For error correction, we assume that it is sufficiently strong to correct bit-flipping errors resulting from program/read disturbance and charge leakage. The two assumptions above on error detection and correction are necessary to provide any meaningful fault-related correctness guarantee.

We also assume that a program operation to a page is performed only after the block that contains the page has been successfully erased and that pages within a block are programmed only in an ascending order and each page is programmed at most once. These assumptions are required by the manufacturers to guarantee the reliability of the programmed page data [1] [2].

In our fault model, we associate a non-deterministic finite state machine with each page. There can be many possible fault models depending on the set of states and the associated transitions in the non-deterministic finite state machine. In the baseline model, the states are defined by the symbolic values that a page may have (i.e., **Erased**, **Data OK**, and **Corrupted**) and also by the past history if it affects the page's future behavior on faults.

Fig. 2 shows a classification diagram that defines the states of our baseline abstract fault model. In the classification diagram, the page state is divided into two classes depending on whether the page contains all 0xFF (i.e., **Erased**) or not (**Programmed**). The **Erased** class further divides into two subclasses depending on whether the page has not undergone any operation since the last successful erase operation and thus is programmable (i.e., **Erased (Programmable)**) or not (i.e., **Erased (not Programmable)**). The **Erased (Programmable)** subclass is one of the states of the finite state machine whereas the **Erased (not Programmable)** subclass contains two states depending on whether there has been a prior program operation since the last successful erase operation (i.e., **Erased (not Programmable)-PP**) or not (i.e., **Erased (not Programmable)-NPP**). The difference between the two states in the **Erased (not Programmable)** subclass lies in the page's future behavior on faults, as we will explain when we describe

transitions in the finite state machine.

Similar to the **Erased** class, the **Programmed** class has two subclasses depending on whether the errors in the data are correctable (i.e., **Programmed (Data OK)**) or not (i.e., **Programmed (Corrupted)**). The **Programmed (Data OK)** is one of the states of the finite state machine whereas the **Programmed (Corrupted)** subclass leads to two states depending on whether there has been a prior program operation since the last successful erase operation (i.e., **Programmed (Corrupted)-PP**) or not (i.e., **Programmed (Corrupted)-NPP**). To summarize, our baseline abstract fault model has the following six states.

- **Erased (Programmable)**: The block that contains this page was successfully erased and after the successful erasure, no attempt has been made to erase the block or to program the page. By the semantics of the erase operation, all the bytes in the page are 0xFF.
- **Erased (not Programmable)-PP**: All the bytes in this page are 0xFF and an attempt has been made to program the page after the last successful erasure of the containing block. The program operation must have been unsuccessful due to an internal or external fault. Otherwise, the page should be in the **Programmed (Data OK)**.
- **Erased (not Programmable)-NPP**: All the bytes in this page are 0xFF and no attempt has been made to program this page (but an unsuccessful erase operation has been performed) after the last successful erasure of the containing block.
- **Programmed (Data OK)**: This page contains data in the **Data OK** state. To reach this state, there must have been a program operation to the page after the last successful erasure of the containing block.
- **Programmed (Corrupted)-PP**: This page is in the **Corrupted** state and an attempt has been made to program the page (although unsuccessful for the same reason as in the **Erased (not Programmable)-PP** case) after the last successful erasure of the containing block.
- **Programmed (Corrupted)-NPP**: This page is in the **Corrupted** state and no attempt has been made to program this page (but an unsuccessful erase operation has been performed) after the last successful erasure of the containing block.

In the baseline abstract fault model, each transition is specified by **OP/F** where

- **OP** is Erase (of the containing block) or Program (of the page) and
- **F** is **OK** when there is no fault, or **IF** when there is an internal fault (erase or program error depending on OP), or **PF** when there is power failure.

In deriving the set of transitions for our fault model, we make the weakest assumptions about the fault behavior to make the model as general as possible. For example, for a program operation, which is allowed only to a page in the **Erased (Programmable)**, we assume that all the three symbolic page

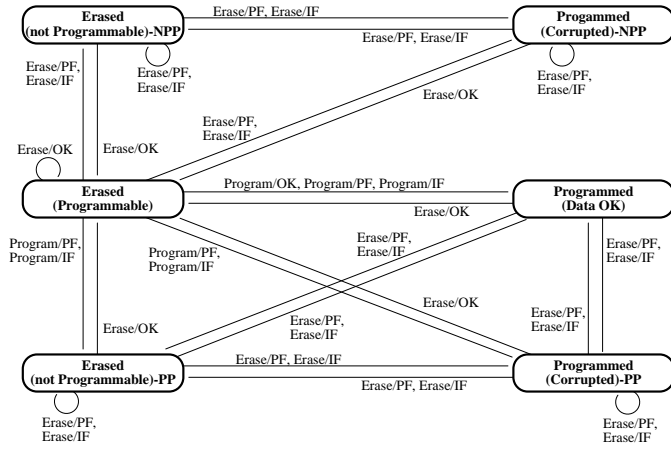


Fig. 3. State transition diagram of baseline fault model.

values (i.e., **Erased**, **Data OK**, and **Corrupted**) are possible when there is an internal fault or power failure. This leads to inclusion of transitions from **Erased (Programmable)** to **Erased (not Programmable)-PP**, **Programmed (Data OK)**, and **Programmed (Corrupted)-PP**.

Unlike a program operation, which is allowed only to a page in the **Erased (Programmable)** state, an erase operation is allowed to a page in any state since the containing block can be erased at any time. We again make the weakest assumption and allow, on an internal fault or power failure, all the symbolic page values that are possible from the current page state. For example, for the three states where their history includes a program operation after the last successful erasure (i.e., **Erased (not Programmable)-PP**, **Programmed (Data OK)**, and **Programmed (Corrupted)-PP**), we add transitions to all of them on an internal fault or power failure.

Similarly, for states where there has been no program operation after the last successful erasure (i.e., **Erased (Programmable)**, **Erased (not Programmable)-NPP**, and **Programmed (Corrupted)-NPP**), we include transitions to **Erased (not Programmable)-NPP** and **Programmed (Corrupted)-NPP** on an internal fault or power failure. The exclusion of a transition to the **Programmed (Data OK)** state is due to the fact that the page contains random data because there has been no program operation since the last successful erase operation and thus the error detection logic will raise an error according to our assumption made earlier in this section. This also explains why we needed to make distinction between **Erased (not Programmable)-PP** and **Erased (not Programmable)-NPP** and also between **Programmed (Corrupted)-PP** and **Programmed (Corrupted)-NPP**. Fig. 3 shows the resulting state transition diagram corresponding to our baseline fault model.

B. Extension (Non-persistent Binding Model)

In the baseline fault model of NAND flash memory in the previous section, transitions between states are triggered only by erase and program operations. This implies that although the model exhibits non-deterministic behavior on a fault during an erase or program operation, a subsequent read operation binds

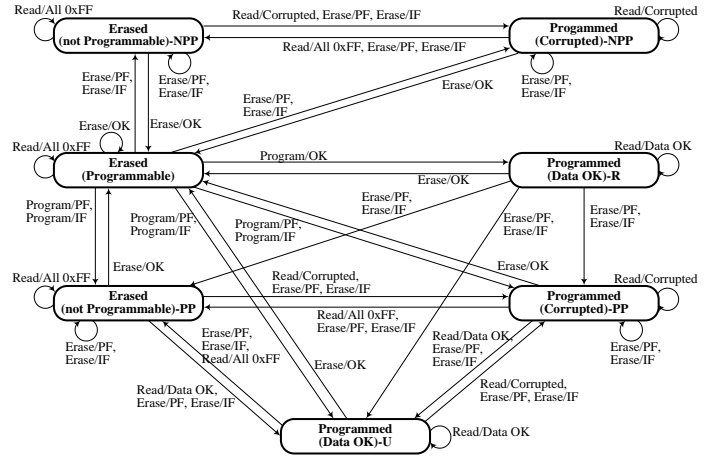


Fig. 4. State transition diagram of non-persistent fault model

the page with one of the three symbolic page values and later reads to the page always return the same symbolic page value. In this sense, we call our previous fault model a persistent binding model. Although simple, this model fails to capture the empirical observation in [6] that if there is power failure during a program operation, the reliability of the programmed data cannot be guaranteed even when the page appears to be properly programmed.

To model such behavior, we should allow transitions between “unreliable” states where an unreliable state is defined as one reached by a fault during an erase or program operation. This can be achieved by allowing a read operation to trigger transitions when the target page is in an unreliable state. To incorporate this, first we need to divide the **Programmed (Data OK)** state into two different states, one that can be reached only by the **Program/OK** transition (and thus is reliable) and the other by a program or erase operation with a fault (and thus is unreliable). We denote the former by **Programmed (Data OK)-R** (R for reliable) and the latter by **Programmed (Data OK)-U** (U for unreliable).

With this setting, there are two reliable states (i.e., **Erased (Programmable)** and **Programmed (Data OK)-R**) and five unreliable states (i.e., **Erased (not Programmable)-PP**, **Erased (not Programmable)-NPP**, **Programmed (Data OK)-U**, **Programmed (Corrupted)-PP**, and **Programmed (Corrupted)-NPP**). From reliable states, we do not add any transitions on a read operation. However, all the possible transitions are added between unreliable states on a read operation. These additional transitions include all the possible transitions among **Erased (not Programmable)-PP**, **Programmed (Data OK)-U**, and **Programmed (Corrupted)-PP** and also those between **Erased (not Programmable)-NPP** and **Programmed (Corrupted)-NPP**. The resulting state transition diagram is shown in Fig. 4 where denoted by Read/SV is a transition by a read operation that returns symbolic page value SV.

C. Extension (MLC)

The only difference between SLC and MLC NAND flash memories from the fault modeling point of view is that in MLC

NAND flash memory page data can potentially be corrupted if there is an internal or external fault during the program operation of a sibling page (i.e., a page that shares the same cells in the MLC NAND flash memory) as specified in the manufacturers' datasheets [1] [2] and also observed empirically in [6]. Our fault model can easily be extended to capture this MLC behavior by including additional transitions corresponding to program operations to sibling pages. For a successful program of a sibling page, there is no state change and thus no need to include additional transitions. On the other hand, when a program operation to a sibling page is unsuccessful due to an internal or external fault, we need to allow all the symbolic page values that are possible from the current page state and add transitions accordingly. This situation is analogous to an unsuccessful erasure due to an internal or external fault in our previous two models and thus we need to add the same transitions for an unsuccessful program to a sibling page as for an unsuccessful erase operation.

IV. SAO-COMPLIANCE

From the fault model in the previous section, it is clear that we need to program a page according to the following rule to guarantee the reliability of the programmed data, which we call the SAO rule:

- Successful
 - erasure of the block containing the page,
 - program of the target page as well as sibling pages
- Program only in an Ascending order, and
- Program at most Once

Also, we call a page to be "SAO-compliant" if all the conditions above are met.

We argue that the SAO-compliance is critical to ensure the FTL's robustness with respect to various faults. For example, if all the accessible pages are SAO-compliant after power-on recovery in an FTL, the anomalous behaviors on power failure reported in [6] (explained in Section II) can be completely avoided.

Despite the critical importance of correct handling of faults, many FTLs do not even mention about the relevant power-on recovery and bad block handling. Even for those few FTLs that do address these fault-related issues, it is not clear whether they are SAO-compliant or not.

To remedy this situation, the abstract fault model presented in this letter can be integrated into the testing of the FTL. In a practical setting, the FTL runs on top of emulated NAND flash memory during testing. The emulated NAND flash memory supports not only the usual read/program/erase operations but also fault injection capability. On each fault, the emulated NAND flash memory records all the possible symbolic states the target page can have based on the finite state machine corresponding to the abstract fault model. If there is an attempt to read a page that has a symbolic page value other than **Data OK**, the FTL may have fault-related vulnerabilities resulting

from failure to meet SAO-compliance. This indicates a need for further debugging to identify the source of the non-compliance.

The abstract fault model is also useful to prove that after power-on recovery a page with a symbolic value other than **Data OK** cannot be read. This proof can be carried out using a model checking technique [7] by integrating the finite state machine corresponding to the abstract fault model and showing that the model never reaches a state where a page with a symbolic value other than **Data OK** is read. This is the approach we used to prove the correctness of a bad block management scheme called X-BMS [8].

V. CONCLUSIONS

In this letter, we have presented an abstract fault model for NAND flash memory that considers not only internal faults (i.e., erase and program errors) but also external ones (i.e., power failure) in flash memory. In constructing the fault model, we make the weakest assumptions on the possible outcomes when a flash operation is subject to faults to make the resulting model as general as possible. This generality also makes possible easy extension to MLC flash memory. Finally, we introduce the concept of "SAO-compliance" as the key criterion of fault resilience of an FTL and explain its relevance to the FTL's correctness.

ACKNOWLEDGEMENTS

The authors would like to thank Associate Editors Nikil Dutt and Jason Xue, and the anonymous reviewers for their detailed and helpful feedback. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No. 2012-0005620). The corresponding author for this paper is Eeye Hyun Nam.

REFERENCES

- [1] Samsung Electronics. Datasheet: K9GBG08U0A 4 G x 8 Bit NAND Flash Memory. November 2010.
- [2] Hynix. Datasheet: H27UAG8T2A 2 G x 8 Bit NAND Flash Memory. July 2009.
- [3] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138-163, 2005.
- [4] T.-S. Jung, Y.-J. Choi, K.-D. Suh, B.-H. Suh, J.-K. Kim, Y.-H. Lim, Y.-N. Koh, J.-W. Park, K.-J. Lee, J.-H. Park, K.-T. Park, J.-R. Kim, J.-H. Yi, and H.-K. Lim, "A 117-mm² 3.3-v only 128-Mb Multilevel NAND Flash Memory for Mass Storage Applications," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1575-1583, 1996.
- [5] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L.R. Nevill, "Bit Error Rate in NAND Flash Memories," in *Proceedings of the 46th International Reliability Physics Symposium*, Phoenix, Arizona, USA, 2008.
- [6] H.-W. Tseng, L. Grupp, and S. Swanson, "Understanding the Impact of Power Loss on Flash Memory," in *Proceedings of the 48th Design Automation Conference (DAC)*, San Diego, California, USA, 2011.
- [7] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," *Lecture Notes on Concurrency and Petri Nets*, LNCS 3098, Springer-Verlag, pp. 87-124, 2004.
- [8] J.H. Yun, "X-BMS: A Provably-Correct Bad Block Management Scheme for Flash Memory Based Storage Systems," Ph.D Dissertation, School of Computer Science and Engineering, Seoul National University, 2011.