

Preface

Solid-State-Drives (SSDs) gained a lot of popularity in the recent few years; compared to traditional HDDs, SSDs exhibit higher speed and reduced power, thus satisfying the tough needs of mobile applications (like smartphones, tablets, and ultrabooks).

A Solid-State-Drive (Chap. 1) is made of a Flash controller plus a bunch of NAND Flash memories (Chap. 2), which are organized in groups called “Flash channels”; each group can have 8, 16 or even more die. Four or eight Flash channels are common in consumer applications but, especially in the enterprise field, the most recent developments span to 32, because of the very high bandwidth requirements. In total, we are easily talking about managing 128, 256 or 512 NAND Flash die. A schematic block diagram of a Solid-State-Drive is sketched in Fig. 1.

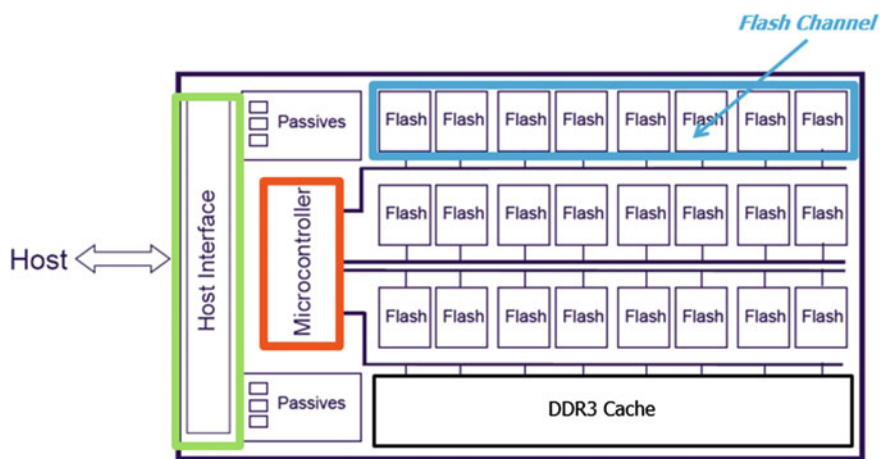


Fig. 1 Solid-State-Drive block diagram

In all SSDs, a Flash controller sits between one or multiple Hosts (i.e. CPUs) and NAND Flash memories, and on each side there are a lot of challenges that designers need to overcome. Moreover, a single controller can have multiple cores, with all the complexity associated with developing a multi-threaded firmware. This book is about how to make simulations of such a complex system, by providing insights on available tools and simulation strategies. As usual, speed and precision don't go hand in hand and it is important to understand when to simulate what, and with which tool. Of course, being able to simulate SSD's performances is mandatory to meet time-to-market, together with product cost and quality.

In order to understand where the challenges are, let's have a closer look at SSD's architectures, by starting from the Host interface. The storage industry has been dominated by *Hard-Disk-Drives* (HDDs) for many decades. HDDs are a rotational media and their data throughput is constrained by the electro-mechanical limitations of the spinning plate and of the servo controllers: this is why HDD's interface speed has remained constant for many years, mainly at 3Gb/s (SATA-II). On the contrary, SSDs don't have mechanical moving parts and they can operate many Flash devices in parallel, thus allowing much higher data transfer rate. As a result, new interfaces have gained momentum in the market: SATA-III and SAS run at 6Gb/s, while PCIe Gen3 is already at 8Gb/s/lane, and PCIe Gen4 is on its way. Of course, there is a practical limit due to area and power, but designers can use parallelism and high speed interfaces to increase performances, and this is why SSD's architectures with 16 or 32 Flash channels are not an exception anymore. When dealing with systems made of hundreds of memory die, queuing effect are not trivial and they strongly depend on the workload and on the scheduling of the data transfers coming from the Host. Transactions could be either sequential or random, and there could be a mixture of read and write operations (typical use cases are 25% read and 75% write, or vice versa). Especially with random traffic, when more requests hit the same NAND die at the same time, a request collision might occur; in other words, the second request in the queue needs to wait until the first one is serviced. This collision effect usually ends up being the root cause for an increased read latency.

Read latencies are part of the *Quality Of Service* (QoS) and they need to be carefully evaluated during the design phase. Not only the worst case matters, but also the shape of the read latency distribution. If the QoS figure doesn't match the requirements, the application running on the Operating System will stall or even hang up. Problem with collisions is that queues are non-linear in nature, and simulations with high statistics are the only practical way to estimate QoS before SSDs prototyping. Is one queue per channel good enough? Do we need to implement high and low priority queues? Can we handle priority at the firmware level? Do we need a queue priority management at the hardware level? Without a prototype, only an SSD simulator can answer these questions.

Let's now have a look at what the Flash controller needs to handle on the Flash side. As a matter of fact, NAND Flash memories have outpaced DRAMs in the technology race. Today we are talking about 256 Gb NAND in a monolithic silicon die, i.e. in less than 150 mm². This unbelievable density is not coming for free. In fact, NAND memories are a storage media characterized by a large number errors

even at early life and they do require a lot of management algorithms. Besides the well-known Wear Leveling and Garbage Collection algorithms, NANDs in ultra-scaled geometries (below 20 nm) require a plethora of signal processing techniques, from Data Randomization to Read Retry, from V_{TH} -shift to Soft Decoding.

Most of these signal processing techniques translate into read oversampling: more reads means higher probability of collisions, bringing us back into the above mentioned QoS problem. Again, because of the non-linearity, simulations are the only way to predict the impact of these management algorithms on performances (QoS, bandwidth, IOPS).

Error Correction Code (ECC) implementation is another task that SSD's controllers need to perform. BCH codes have been used for many years but the higher and higher NAND BER (Bit Error Rate) has driven towards the adoption of LDPC: these codes are known to be the codes that can get closer to the Shannon limit in real implementations. One drawback of LDPC codes is their need of Soft Information, which has to be extracted from the NAND and then fed into the LDPC decoder by the Flash controller (Chap. 4).

Another important topic is power (Chap. 7). In order to avoid changing existing infrastructures, SSDs need to fit in the same physical space of HDDs and, therefore, in their power envelope. Unfortunately, NAND power consumption is strongly dependent upon the Flash operation (i.e. read, write, erase). For many generations, SSD's power has been over- or under-estimated by adopting the worst case power consumption scenario (i.e. assuming write operations all the time); this approach is running out of steam because it always leads to exceeding power specs with the most recent NAND technologies. In order to correctly evaluate power, it is necessary to estimate the number of operations per time interval and their specific type (read, write, erase). Simulations are again the only practical way to address this problem: when the prototype is ready (especially if the controller is a multi-million dollar ASIC) there are no chances to significantly reduce power, it's too late!

Last but not least, NAND Flash memories wear out, i.e. their performances change over time and they have a limited lifetime. This is even more challenging. Think about the following situation. Let's assume that we have built a prototype of a particular SSD and that performances are within the spec limits. How can we make sure that specs are met when NANDs reach their end of life? Without the proper simulator, the only viable solution is to age a bunch of drives: every time we want to change/test the configuration of a single parameter, we need to consume a number of SSDs, and this is an extremely expensive and time-consuming validation flow.

What's the fastest way to assess QoS? What's the most precise? What's the fastest way to simulate Flash wear-out? ... these are just some of the questions that designers need to answer during the development of a modern SSD.

The authors have taken a careful look at both literature and available simulation tools, including popular solutions like VSSIM, NANDFlashSim, and DiskSim. All these solutions are benchmarked against performances of real SSDs, an OCZ VERTEX 120GB and a large enterprise storage platform, that have been

measured under different traffic workloads. PROs and CONs of each simulator are analyzed and it is clearly indicated which kind of answers each of them can give and at a what price (Chap. 3).

Over the last few years the authors of this book developed an advanced simulator named “SSDExplorer” (Chap. 3) which has been used for evaluating multiple phenomena, from QoS to Read Retry, from LDPC Soft Information to power, from Flash aging to FTL (Chap. 4). Basically, SSDExplorer is a fine-grained SSD virtual platform that was developed with the following goals in mind:

- offer a RAD tool (Rapid Application Development) for SSD design space exploration;
- accurately predict performance figures of the target architecture;
- offer a wear-out aware framework for complex correction algorithm exploration;
- avoid the overdesign of the resources of the SSD for a target performance.

In Chap. 6, SSDExplorer is used to evaluate the possible impact of emerging non-volatile memories, such as Resistive RAM (RRAM, Chap. 5), on future SSD architectures. Does it make sense to fully replace NANDs with one of the emerging memories? What’s the benefit? Is it better to develop a hybrid drive, where a RRAM is used as cache? Most of the new memories are still in the development phase, well before a real mass production; as a matter of fact, simulations are the only way to figure out where a new non-volatile technology can really help, in terms of both performances and cost saving.

In the last chapter of this book, SSD simulators are addressed in a much broader context, i.e. the analysis of what happens when SSDs are connected to the OS (*Operating System*) and to the user application (for example a database search). QoS is again a good example: if a read request is not serviced within a specific time window, the application hangs up, with a very clear impact on the user experience, as the reader can easily imagine. In order to allow this high level simulation, with the required level of precision, SSD simulators need to be directly connected to the end application; if requirements are not met, SSD’s firmware can be changed and its impact immediately evaluated. In Chap. 8 authors walk the reader through the full simulation flow of a real system-level by combining SSDExplorer with the QEMU virtual platform. The reader will be impressed by the level of know-how and the combination of models that such simulations are asking for.

In this short introduction we have shown several reasons and examples of why developing a Solid-State-Drive requires a solid simulation strategy and a set of reliable simulation tools. SSDs are very complex, they have to manage several NAND Flash memories in parallel, without forgetting the more and more stringent requirements of end user applications. This book provides an overview of the state-of-the-art simulation techniques, together with an outlook of the challenges that designers will have to address in the coming few years.

Solid-State-Drives (SSDs) Modeling

Simulation Tools & Strategies

Micheloni, R. (Ed.)

2017, XIII, 170 p. 154 illus., 58 illus. in color., Hardcover

ISBN: 978-3-319-51734-6