

# Chapter 1

## Solid State Drives (SSDs)

Rino Micheloni and Luca Crippa

**Abstract** *Solid-state drives* (SSDs) are unanimously considered the enabling factor for bringing enterprise storage performances to the next level. Indeed, the rotating-storage technology of *Hard Disk Drives* (HDDs) can't achieve the access-time required by applications where response time is the critical factor. On the contrary, SSDs are based on solid state memories, namely NAND Flash memories: in this case, there aren't any mechanical parts and random access to stored data can be much faster, thus addressing the above mentioned needs. In many applications though, the interface between host processors and drives remains the performance bottleneck. This is why SSD's interface has evolved from legacy storage interfaces, such as SAS and SATA, to PCIe, which enables a direct connection of the SSD to the host processor. In this chapter we give an overview of the SSD's architecture by describing the basic building blocks, such as the Flash controller, the Flash File System (FFS), and the most popular I/O interfaces (SAS, SATA and PCIe).

### 1.1 Introduction

*Solid State Drives* (SSDs) promise to greatly enhance enterprise storage performance. While electromechanical *Hard Disk Drives* (HDDs) have continuously ramped in capacity, the rotating-storage technology doesn't provide the access-time or transfer-rate performance required in demanding enterprise applications, including on-line transaction processing, data mining, and cloud computing. Client applications are also in need of an alternative to electromechanical disk drives that can deliver faster response times, use less power, and fit into smaller mobile form factors.

Flash-memory-based SSDs can offer much faster random access to data and faster transfer rates. Moreover, SSD's capacity is now at the point where solid state

---

R. Micheloni (✉) · L. Crippa  
Performance Storage Business Unit, Microsemi Corporation, Vimercate, Italy  
e-mail: rino.micheloni@ieee.org

L. Crippa  
e-mail: luca.crippa@ieee.org

drives can serve as rotating-disk replacements. But in many applications the interface between host and drives remains the performance bottleneck. SSDs with legacy storage interfaces, such as SAS and SATA, are proving useful, and PCI-Express (PCIe) SSDs will further increase performance and improve responsiveness, being directly connected to the host processor.

## 1.2 SSD's Architecture

A basic block diagram of a solid state drive is shown in Fig. 1.1. In addition to memories and a Flash controller, there are usually other components. For instance, an external DC-DC converter can be added in order to drive the internal power supply, or a quartz can be used for a better clock precision. Of course, reasonable filter capacitors are inserted for stabilizing the power supply. It is also very common to have an array of temperature sensors for power management reasons. For data caching, a fast DDR memory is frequently used: during a write access, the cache

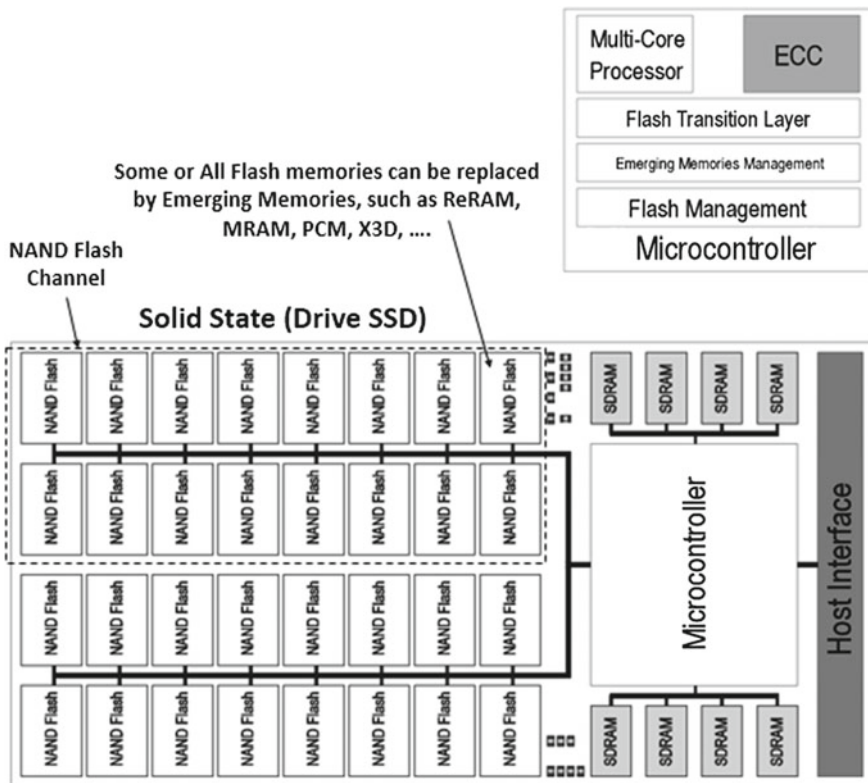


Fig. 1.1 Block diagram of an SSD

is used for storing data before their transfer to the Flash. The benefit is that data updating, e.g. of routing tables, is faster and does not wear out the Flash.

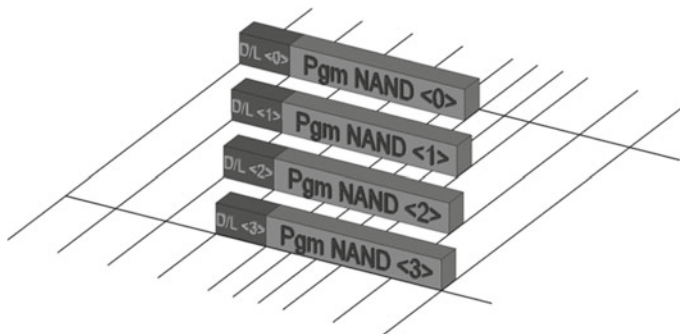
A typical memory system is composed of several NAND memories [1]. Typically, an 8-bit bus [2, 3], usually called “channel”, is used to connect different memories to the controller (Fig. 1.1). It is important to underline that multiple Flash memories in a system are both a means for increasing storage density and read/write performances [4].

Operations on a channel can be interleaved, which means that a second chip can be addressed while the first one is still busy. For instance, a sequence of multiple write operations can be directed to a channel, addressing different NANDs, as shown in Fig. 1.2: in this way, the channel utilization is maximized by pipelining the data load phase. In fact, while the program operation takes place inside a memory chip, the corresponding Flash channel is free. The total number of Flash channels is a function of the target application, but tens of channels are becoming quite common. Thanks to interleaving, given the same Flash programming time, SSD’s throughput greatly improves.

The memory controller is responsible for scheduling the accesses to the memory channels. The controller uses dedicated engines for the low level communication protocol with the Flash.

Moreover, it is clear that the data load phase is not negligible compared to the program operation (the same comment is valid for data output): therefore, increasing I/O interface speed is another smart way to improve performances: DDR-like interfaces are discussed in more details in Chap. 2. As the speed increases, more NAND can be operated in parallel before saturating the channel. For instance, assuming a target of 30 MB/s, 2 NANDs are needed with a minimum DDR frequency of about 50 MHz. Given a page program time of 200  $\mu$ s, at 50 MHz four NANDs can operate in interleaved mode, doubling the write throughput. Of course, power consumption is another metric to be carefully considered.

After this high level overview of the SSD’s architecture, let’s move to the heart of the architecture: the memory (Flash) controller.



**Fig. 1.2** Interleaved operations on one Flash channel

### 1.3 Flash Controller

A memory controller has two fundamental tasks:

1. provide the most suitable interface and protocol towards both the host and the Flash memories;
2. efficiently handle data, maximizing transfer speed, data integrity and retention of the stored information.

In order to carry out such tasks, an application specific device is designed, embedding a standard processor—usually 8–16 bits—together with dedicated hardware to handle time-critical tasks.

Generally speaking, the memory controller can be divided into four parts, which are implemented either in hardware or in firmware (Fig. 1.3).

Moving from the host to the Flash, the first part is the host interface, which implements the required industry-standard protocol (PCIe, SAS, SATA, etc.), thus ensuring both logical and electrical interoperability between SSDs and hosts. This block is a mix of hardware (buffers, drivers, etc.) and firmware, which decodes the command sequence invoked by the host and handles the data flow to/from the Flash memories.

The second part is the *Flash File System* (FFS) [5]: that is, the file system which enables the use of SSDs like magnetic disks. For instance, sequential memory access on a multitude of sub-sectors which constitute a file is organized by linked lists (stored on the SSD itself), which are used by the host to build the *File Allocation Table* (FAT). The FFS is usually implemented in form of firmware inside the controller, each sub-layer performing a specific function. The main functions are: *Wear leveling Management*, *Garbage Collection* and *Bad Block Management*. For all these functions, tables are widely used in order to map sectors and pages from the logical domain to the physical domain (Flash Translation Layer or FTL) [6, 7], as shown in Fig. 1.4. The top row is the logical view of the memory, while the bottom row is the physical one. From the host perspective, data are transparently written and overwritten inside a given logical sector: due to Flash limitations, overwrite on the same page is not possible; therefore, a new page (sector) must be allocated in the physical block, and the previous one is marked as invalid. It is clear that, at some point in time, the current physical block becomes full and a second one (coming from the pool of “buffer” blocks) has to take over that logic address.

The required translation tables are stored on the SSD itself, thus reducing the overall storage capacity.

### 1.4 Wear Leveling

Usually, not all the data stored within the same memory location change with the same frequency: some data are often updated while others don't change for a very

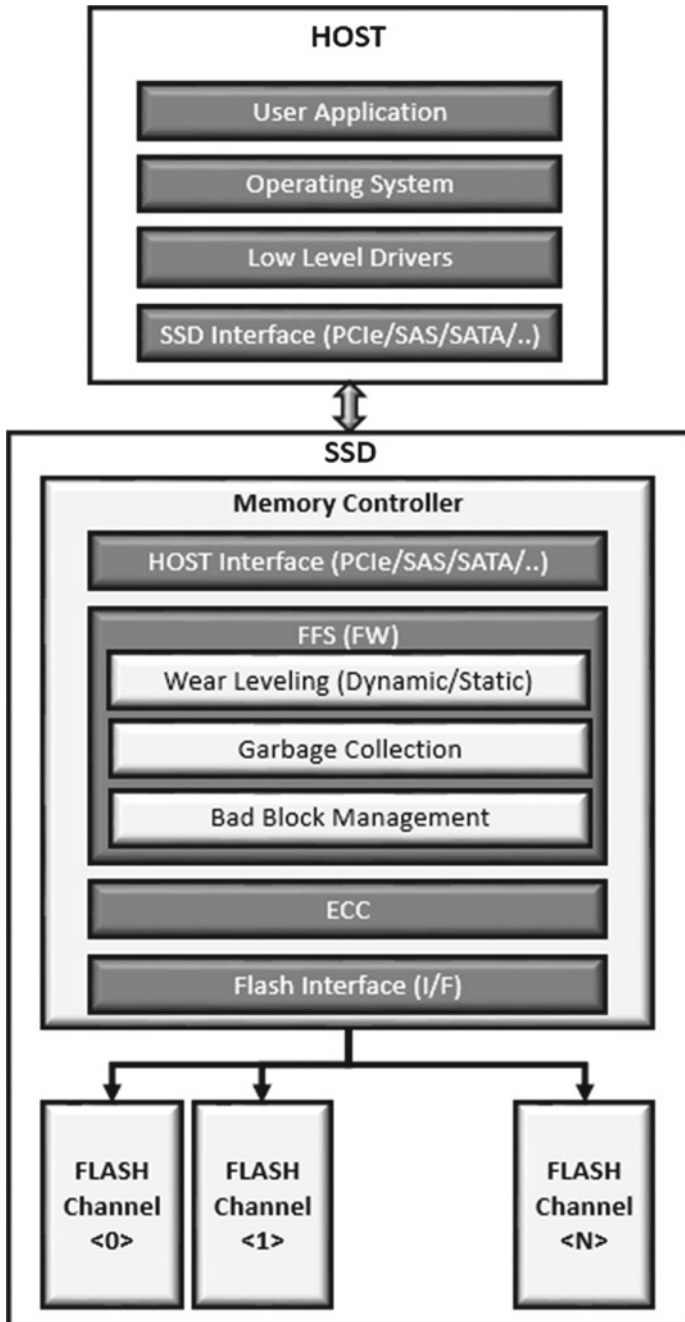
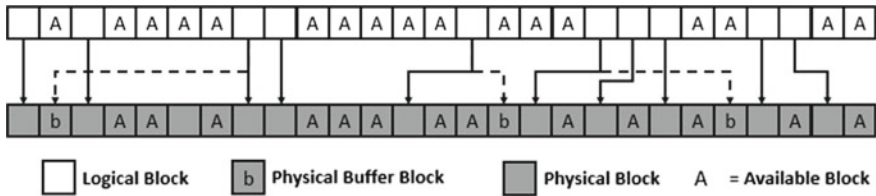


Fig. 1.3 High level view of a Flash controller



**Fig. 1.4** Logical to physical block management

long time—in the extreme case, for the whole life of the device. It’s clear that the blocks containing frequently-updated information are stressed with a larger number of write/erase cycles, while the blocks containing information updated very rarely are much less stressed.

In order to mitigate disturbs, it is important to keep the aging of each page/block as minimum and as uniform as possible: that is, the number of both read and program cycles applied to each page must be monitored. Furthermore, the maximum number of allowed program/erase cycles for a block (i.e. its endurance) should be considered: in case SLC NAND memories are used, this number is in the order of 20–30 k cycles, which is reduced to 10–15 k and 1–5 k for MLC and TLC NAND, respectively.

Wear Leveling techniques exploit the concept of logical to physical translation: each time the host application needs to update the same (logical) sector, the memory controller dynamically maps that sector to a different (physical) sector, of course keeping track of the mapping. The out-of-date copy of the sector is tagged as invalid and eligible for erase. In this way, all the physical blocks are evenly used, thus keeping the aging under a reasonable value.

Two kinds of approaches are possible: Dynamic Wear Leveling is normally used to follow up a user’s request of update, writing to the first available erased block with the lowest erase count; with Static Wear Leveling every block, even the least modified, is eligible for re-mapping as soon as its aging deviates from the average value.

## 1.5 Garbage Collection

Both wear leveling techniques rely on the availability of free sectors that can be filled up with the updates: as soon as the number of free sectors falls below a given threshold, sectors are “compacted” and multiple, obsolete copies are deleted. This operation is performed by the Garbage Collection module, which selects the blocks containing the invalid sectors, it copies the latest valid content into free sectors, and then erases such blocks (Fig. 1.5).

In order to minimize the impact on performances, garbage collection can be performed in background. The aging uniformity driven by wear leveling distributes wear out stress over the whole array rather than on single hot spots. Hence, given a specific

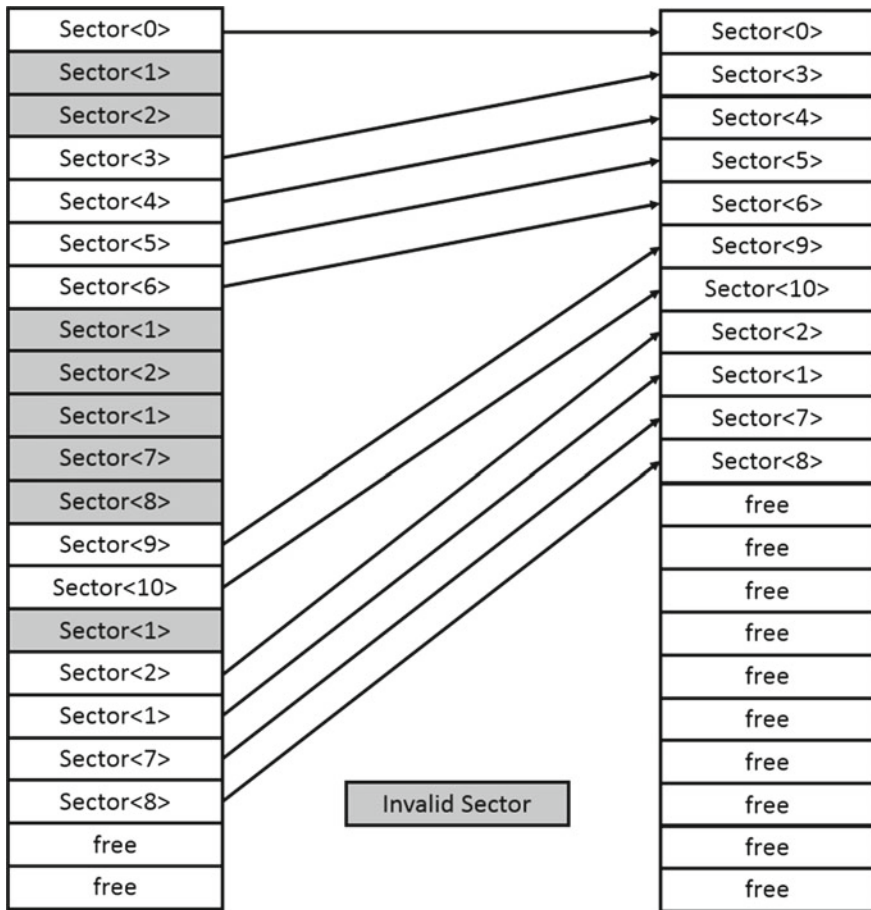


Fig. 1.5 Garbage collection

workload and usage time, the bigger the memory density, the lower the wear out per cell.

### 1.6 Bad Block Management

No matter how smart the Wear Leveling algorithm is, an intrinsic limitation of NAND Flash memories is represented by the presence of the so-called *Bad Blocks* (BB), i.e. blocks which contain one or more locations whose reliability is not guaranteed. The *Bad Block Management* (BBM) module creates and maintains a map of bad blocks, as shown in Fig. 1.6: this map is created in the factory and then updated during SSD's lifetime, whenever a block becomes bad.

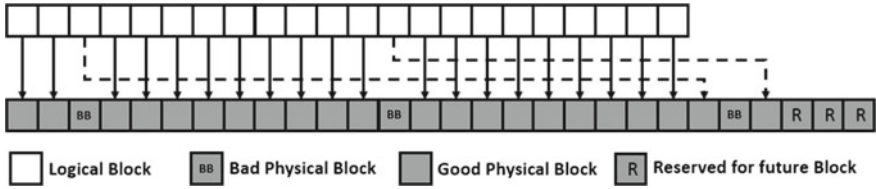


Fig. 1.6 Bad Block Management (BBM)

### 1.7 Error Correction Code (ECC)

This task is typically executed by a hardware accelerator inside the memory controller. Examples of memories with embedded ECC were also reported [8–10]. Most popular ECC codes, correcting more than one error, are Reed-Solomon and BCH [11].

NAND raw BER gets worse generation after generation approaching, as a matter of fact, the Shannon limit. As a consequence, correction techniques based on soft information processing are becoming more and more popular: LDPC (*Low Density Parity Check*) codes are an example of this soft information approach. Chapter 4 provides details about how these codes can be handled by SSD simulators.

### 1.8 SSD’s Interfaces

There are 3 main interface protocols used to connect SSDs into server and/or storage infrastructure: *Serial Attached SCSI* (SAS), *Serial ATA* (SATA) and *PCI-Express*. *PCI-Express* based SSDs deliver the highest performances and are mainly used in server based deployments as a plug-in card inside the server itself. *SAS* SSDs deliver pretty good level of performances and are used in both high-end servers and mid-range and high-end storage enclosures. *SATA* based SSDs are used mainly in client applications and in entry-level and mid-range server and storage enclosures.

### 1.9 SAS and SATA

*Serial Attached SCSI* (SAS) is a communication protocol traditionally used to move data between storage devices and host. SAS is based on a serial point-to-point physical connection. It uses a standard SCSI command set to drive device communications. Today, SAS based devices most commonly run at 6 Gbps, but 12 Gbps SAS are available too. On the other side, SAS interface can also be run at slower speeds—1.5 Gbps and/or 3 Gbps to support legacy systems.



SAS offers backwards-compatibility with second-generation SATA drives. The T10 technical committee of the *International Committee for Information Technology Standards* (INCITS) develops and maintains the SAS protocol; the *SCSI Trade Association* (SCSITA) promotes the technology.

*Serial ATA* (SATA or *Serial Advanced Technology Attachment*) is another interface protocol used for connecting host bus adapters to mass storage devices, such as hard disk drives and solid state drives. Serial ATA was designed to replace the older parallel ATA/IDE protocol. SATA is also based on a point-to-point connection. It uses ATA and ATAPI command sets to drive device communications. Today, SATA based devices run either at 3 or 6 Gbps.

Serial ATA industry compatibility specifications originate from the Serial ATA International Organization [12] (aka. SATA-IO).

A typical SAS eco-system consists of SAS SSDs plugged into a SAS backplane or a host bus adapter via a point to point connection, which, in turn, is connected to the host microprocessor either via either an expander or directly, as shown in Fig. 1.7.

Each expander can support up to 255 connections to enable a total of 65535 (64k) SAS connections. Indeed, SAS based deployments enable the usage of a large number of SAS SSDs in a shared storage environment.

SAS SSDs are built with two ports. This dual port functionality allows host systems to have redundant connections to SAS SSDs. In case one of the connections to the SSD is either broken or not properly working, host systems still have the second port that can be used to maintain continuous access to the SAS SSD. In enterprise applications where high availability is an absolute requirement, this feature is key.

SAS SSDs also support hot-plug: this feature enables SAS SSDs to be dynamically removed or inserted while the system is running; it also allows automatic detection of

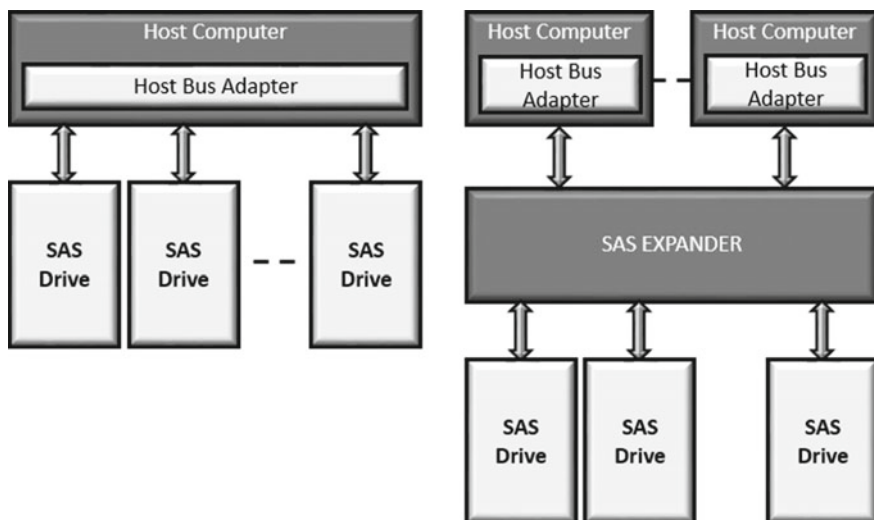


Fig. 1.7 SAS connectivity

newly inserted SAS SSDs. In fact, while a server or storage system is running, newly inserted SAS SSDs can be dynamically configured and put in use. Additionally, if SAS SSDs are pulled out of a running system, all the in-flight data that were already committed to the host system are stored inside the SAS drive, and can be accessed at a later point, when the SSD is powered back on.

Differently from SAS, a typical SATA infrastructure consists of SATA SSDs point-to-point connected to a host bus adapter driven by the host microprocessor. In addition, SATA drives are built with a single port, unlike SAS SSDs. These two main differences make SATA based SSDs a good fit for entry or mid-range deployments and consumer applications.

The SATA protocol supports hot-plug; however, not all SATA drives are designed for it. In fact, hot-plug requires specific hardware (i.e. additional cost) to guarantee that committed data, which are actually still in-flight, are safely stored during a power drop.

It is worth highlighting that SATA drives may be connected to SAS backplanes, but SAS drives can't be connected to SATA backplanes. Of course, this another reason for the broad SATA penetration in the market.

Similarities between SAS and SATA are:

- Both types plug into the SAS backplane;
- The drives are interchangeable within a SAS drive bay module;
- Both are long proven technologies, with worldwide acceptance;
- Both employ point-to-point architecture;
- Both are hot pluggable.

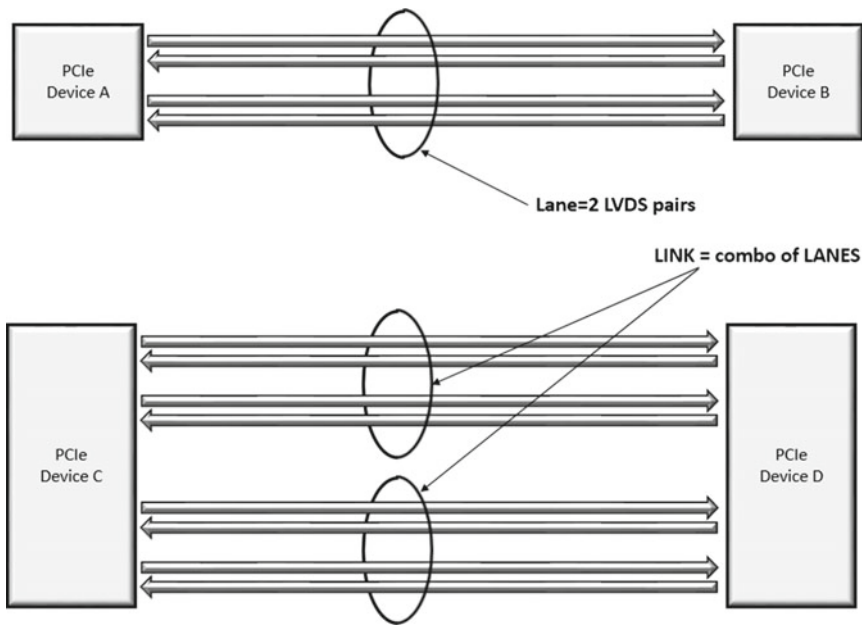
Differences between SAS and SATA are:

- SATA devices are cheaper;
- SATA devices use the ATA command set, SAS the SCSI command set;
- SAS drives have dual port capability and lower latencies;
- While both types plug into the SAS backplane, a SATA backplane cannot accommodate SAS drives;
- SAS drives are tested against much more rigid specifications;
- SAS drives are faster and offer additional features, like variable sector size, LED indicators, dual port, and data integrity;
- SAS supports link aggregation (wide port).

## 1.10 PCI-Express

PCI-Express (*Peripheral Component Interconnect Express*) or PCIe is a bus standard that replaced PCI and PCI-X. PCI-SIG (*PCI Special Interest Group*) creates and maintains the PCIe specification [13].

PCIe is used in all computer applications including enterprise servers, consumer personal computers (PC), communication systems, and industrial applications.



**Fig. 1.8** PCI Express lane and link. In Gen2, 1 lane runs at 5Gbps/direction; a 2-lane link runs at 10 Gbps/direction

Unlike the older PCI bus topology, which uses shared parallel bus architecture, PCIe is based on point-to-point topology, with separate serial links connecting every device to the root complex (host). Additionally, a PCIe link supports full-duplex communication between two endpoints. Data can flow upstream (UP) and downstream (DP) simultaneously. Each pair of these dedicated unidirectional serial point-to-point connections is called a *lane*, as depicted in Fig. 1.8. The PCIe standard is constantly under improvement, with PCIe 3.0 being the latest version of the standard available in the market (Table 1.1). The standardization body is currently working on defining Gen4.

Other important features of PCIe include power management, hot-swappable devices, and the ability to handle peer-to-peer data transfers (sending data between two end points without routing through the host) [14]. Additionally, PCIe simpli-

**Table 1.1** Throughput of different PCIe generations

PCIe version	Year introduced	Throughput per lane
PCIe 1.0 (Gen1)	2003	250 MB/s
PCIe 2.0 (Gen2)	2007	500 MB/s
PCIe 3.0 (Gen3)	2010	1 GB/s

fies board design by utilizing a serial technology, which drastically reduces the wire count when compared to parallel bus architectures.

The PCIe link between two devices can consist of 1–32 lanes. The packet data is striped across lanes, and the lane count is automatically negotiated during device initialization.

The PCIe standard defines slots and connectors for multiple widths:  $\times 1$ ,  $\times 4$ ,  $\times 8$ ,  $\times 16$ ,  $\times 32$ . This allows PCIe to serve lower throughput, cost-sensitive applications as well as performance-critical applications.

PCIe uses a packet-based layered protocol, consisting of a transaction layer, a data link layer, and a physical layer, as shown in Fig. 1.9.

The transaction layer handles packetizing and de-packetizing of data and status-message traffic. The data link layer sequences these *Transaction Layer Packets* (TLPs) and ensures that they are reliably delivered between two endpoints. If a transmitter device sends a TLP to a remote receiver device and a CRC error is detected, the transmitter device gets a notification back. The transmitter device automatically replays the TLP. With error checking and automatic replay of failed packets, PCIe ensures very low *Bit Error Rate* (BER).

The Physical Layer is split in two parts: the Logical Physical Layer and the Electrical Physical Layer. The Logical Physical Layer contains logic gates for processing packets before transmission on the Link, and processing packets from the Link to the Data Link Layer. The Electrical Physical Layer is the analog interface of the Physical Layer: it consists of differential drivers and receivers for each lane.

TLP assembly is shown in Fig. 1.10. Header and Data Payload are TLP's core information: Transaction Layer assembles this section based on the data received from the application software layer. An optional End-to-End CRC (ECRC) field can be appended to the packet. ECRC is used by the ultimate targeted device of this packet to check for CRC errors inside Header and Data Payload. At this point, the Data Link Layer appends a sequence ID and local CRC (LCRC) field in order to protect the ID. The resultant TLP is forwarded to the Physical Layer which concatenates a Start and End framing characters of 1 byte each to the packet. Finally, the packet is encoded and differentially transmitted on the Link by using the available Lanes.

Today, PCIe is a high volume commodity interconnect used in virtually all computers, from consumer laptops to enterprise servers, as the primary motherboard technology that interconnects the host CPU with on-board ICs and add-on peripheral expansion cards.

## 1.11 The Need for High Speed Interfaces

Processor vendors have continued to ramp the performance of individual processor cores, to combine multiple cores in one chip, and to develop technologies that can closely couple multiple chips in multi-processor systems. Ultimately, all of the cores in such a scenario need access to the same storage subsystem.

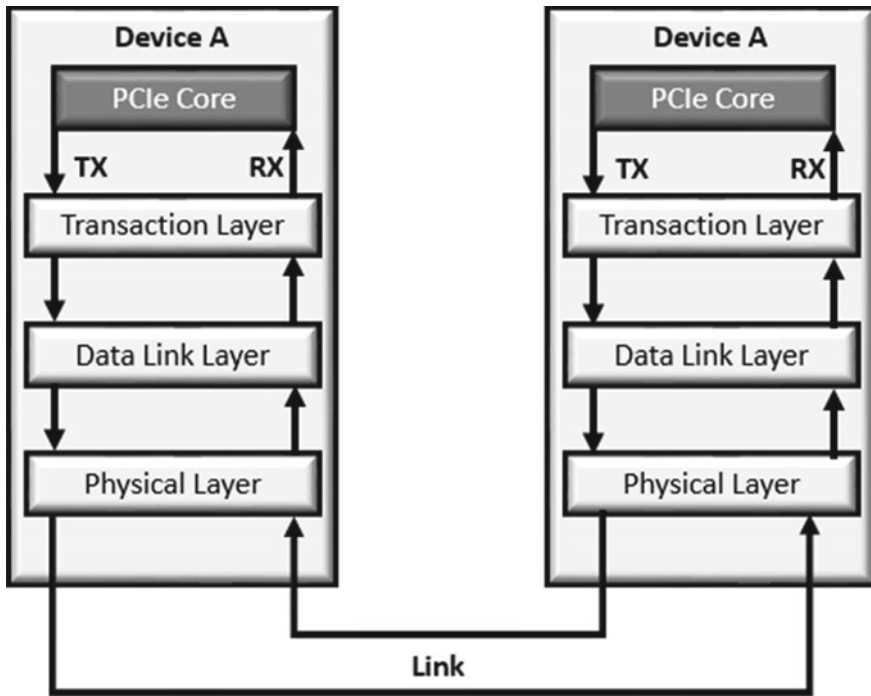


Fig. 1.9 PCIe Layered architecture

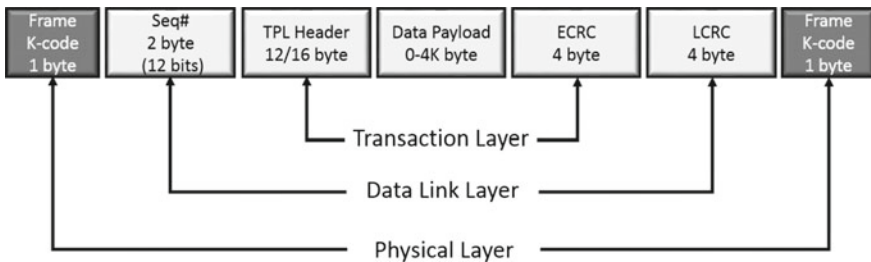


Fig. 1.10 Transaction Layer Packet (TLP) assembly

Enterprise IT managers are eager to utilize the multiprocessor systems because they have the potential of boosting the number of I/O operations per second (IOPS) that a system can process and also the number of IOPS per Watt. This multiprocessing computing capability offers better IOPS relative to cost and power consumption—assuming the processing elements can get access to the data in a timely fashion. Active processors waiting on data waste time and money.

There are, of course, multiple levels of storage technology in a system that ultimately feed code and data to each processor core. Generally, each core includes local cache memory that operates at core speed. Multiple cores in a chip share a second-

level and, sometimes, a third-level cache. And DRAM feeds the caches. DRAM and caches access-times, together with data-transfer speed have scaled to match processor's performance.

The issue is the performance gap between DRAM and HDD in terms of access time and data rate. Disk/drive vendors have done a great job at designing and manufacturing higher-capacity, lower-cost-per-Gbyte disks/drives; but the drives inherently have limitations in terms of how fast they can access data, and then how fast they can transfer these data to DRAM.

Access time depends on how quickly a hard drive can move the read head over the required data track on a disk, and the rotational latency of the addressed sector to move underneath the head. The maximum transfer rate is dictated by the rotational speed of the disk and the data encoding scheme: together they determine the number of bytes per second that can be read from the disk.

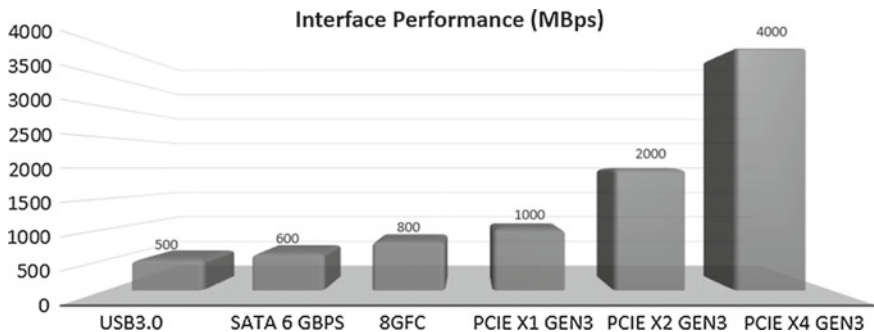
Hard drives perform relatively well in reading and transferring sequential data. But random seek operations add latency. And even sequential read operations can't match the data appetite of the latest processors.

Meanwhile, enterprise systems that perform on-line transaction processing, such as financial transactions and data mining (e.g. applications for customer relationship management) require highly random access to data. Also cloud computing has strong random requirements, especially when looking at virtualization, which expands the scope of different applications that a single system has active at any one time. Every microsecond of latency directly relates to money, utilization of processors and system power.

Fortunately, Flash memories can help reducing the performance gap between DRAM and HDD. Flash is slower than DRAM but offers a lower cost per Gbyte of storage. That cost is more expensive than disk storage, but enterprises will gladly pay the premium because Flash also offers much better throughput in terms of Mbyte/s and faster access to random data, resulting in better cost-per-IOPS compared to rotating storage.

Neither the legacy disk-drive form factor nor the interface is ideal for Flash-based storage. SSD manufacturers can pack enough Flash devices in a 2.5-in form factor to easily exceed the power profile developed for disk drives. And Flash can support higher data transfer rates than even the latest generation of disk interfaces.

Let's examine the disk interfaces more closely (Fig. 1.11). The third-generation SATA and SAS support 600 Mbyte/s throughput, and drives based on those interfaces have already found usage in enterprise systems. While those data rates support the



**Fig. 1.11** Interface performance. PCIe improves overall system performance by reducing latency and increasing throughput

fastest electromechanical drives, new NAND Flash architectures and multi-die Flash packaging deliver aggregate Flash bandwidth that exceeds the throughput capabilities of SATA and SAS interconnects. In short, the SSD performance bottleneck has shifted from the storage media to the host interface. Therefore, many applications need a faster host interconnect to take full advantage of Flash storage.

The PCIe host interface can overcome this storage performance bottleneck and deliver unparalleled performance by attaching the SSD directly to the PCIe host bus. For example, a 4-lane (x4) PCIe Generation 3 (Gen3) link can deliver 4 GByte/s data rates. Simply put, PCIe meets the desired storage bandwidth. Moreover, the direct PCIe connection can reduce system power and slash the latency that’s attributable to the legacy storage infrastructure.

Clearly, an interface such as PCIe can handle the bandwidth of a multi-channel Flash storage subsystem and can offer additional performance advantages. SSDs that use a disk interface also suffer latency added by a storage-controller IC that handles disk I/O. PCIe devices connect directly to the host bus, thus eliminating the architectural layer associated with the legacy storage infrastructure. The compelling performance of PCIe SSDs has resulted in system manufacturers placing PCIe drives in servers as well as in storage arrays to build tiered storage systems that accelerate applications while improving cost-per-IOPS.

The benefits of using PCIe as a storage interconnect are clear. You can achieve over 6x the data throughput compared to SATA or SAS. You can eliminate components such as host bus adapters and SerDes ICs on the SATA and SAS interfaces—saving money and power at the system level. And PCIe moves the storage closer to the host CPU reducing latency, as shown in Fig. 1.12.

Latency, IOPS, bandwidth, power, interface speed, number of channels, NAND-type (SLC, MLC, TLC, QLC) are all parameters that SSD designers need to take into account to meet their target specifications at minimum cost. Looking forward,

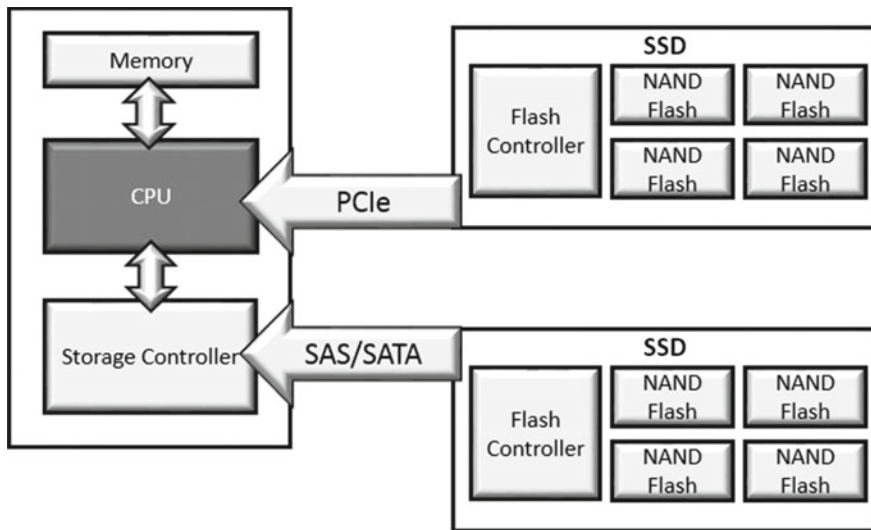


Fig. 1.12 PCIe SSD versus SAS/SATA SSD

emerging memories will be part of the game too. It is clear that, given this number of variables, a simple approach based on hardware prototyping is hard to pursue, especially when looking at the time-to-market. Therefore, SSD simulators become a must have and they will be covered in great details in the following chapters.

## References

1. G. Campardo, R. Micheloni, D. Novosel, "VLSI-Design of Non-Volatile Memories", Springer-Verlag, 2005.
2. [www.onfi.org](http://www.onfi.org)
3. [www.jedec.org](http://www.jedec.org)
4. C. Park et al., "A High Performance Controller for NAND Flash-based Solid State Disk (NSSD)", IEEE Non-Volatile Semiconductor Memory Workshop NVSMW, pp. 17–20, Feb. 2006.
5. A. Kawaguchi, S. Nishioka, and H. Motoda. "A flash-memory based file system", Proceedings of the USENIX Winter Technical Conference, pp. 155–164, 1995.
6. J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho. A space-efficient flash translation layer for compactflash systems. IEEE Transactions on Consumer Electronics, 48(2):366–375, May 2002.
7. S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S.-W. Park, and H.-J. Songe. "FAST: A log-buffer based ftl scheme with fully associative sector translation", 2005 US-Korea Conference on Science, Technology, & Entrepreneurship, August 2005.
8. T. Tanzawa, T. Tanaka, K. Takekuchi, R. Shirota, S. Aritome, H. Watanabe, G. Hemink, K. Shimizu, S. Sato, Y. Takekuchi, and K. Ohuchi, "A compact on-chip ECC for low cost Flash memories," IEEE J.Solid-State Circuits, vol. 32, pp. 662–669, May 1997.



9. G. Campardo, R. Micheloni et al., “40-nm<sup>2</sup> 3-V-only 50-MHz 64-Mb 2-b/cell CHE NOR Flash memory” – IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, Nov. 2000, pp. 1655–1667.
10. R. Micheloni et al., “A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput”, IEEE International Solid-State Circuits Conference Dig. Tech. Papers, pp. 142–143, Feb. 2006.
11. R. Micheloni, A. Marelli, R. Ravasio, “Error Correction Codes for Non-Volatile Memories”, Springer-Verlag, 2008.
12. <http://www.sataio.org>
13. [www.pcisig.com](http://www.pcisig.com)
14. R. Budruk, D. Anderson, T. Shanley, “PCI Express System Architecture”, Mindshare 2003.



<http://www.springer.com/978-3-319-51734-6>

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