Chapter 2
SSD Architecture and PCI Express Interface

K. Eshghi and R. Micheloni

Abstract Flash-memory-based solid-state disks (SSDs) provide faster random access and data transfer rates than electromechanical drives and today can often serve as rotating-disk replacements, but the host interface to SSDs remains a performance bottleneck. PCI Express (PCIe)-based SSDs together with an emerging standard called NVMe (Non-Volatile Memory express) promises to solve the interface bottleneck.

This chapter walks the reader through the SSD block diagram, from the NAND memory to the Flash controller (including wear leveling, bad block management, and garbage collection). PCIe basics and different PCIe SSD architectures are reviewed. Finally, an overview on the standardization effort around PCI Express is presented.

2.1 Introduction

> Creativity is just connecting things. When you ask creative people how they did something, they feel a little guilty because they didn't really do it, they just saw something. It seemed obvious to them after a while.
>
> – Steve Jobs

Solid-state drives promise to greatly enhance enterprise storage performance. While electromechanical disk drives 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

K. Eshghi (✉) • R. Micheloni
Enterprise Computing Division, Integrated Device Technology, Inc.,
San Jose, CA, USA
e-mail: kamyar.eshghi@alum.mit.edu; rino.micheloni@ieee.org

R. Micheloni et al., *Inside Solid State Drives (SSDs)*, Springer Series in Advanced Microelectronics 37, DOI 10.1007/978-94-007-5146-0_2,

an alternative to electromechanical disk drives that can deliver faster response times, use less power, and fit in smaller mobile form factors.

Flash-memory-based *Solid-State Disks* (SSDs) can offer much faster random access to data and faster transfer rates. Moreover, SSD capacity is now at the point that the drives can serve as rotating-disk replacements. But for many applications the host interface to SSDs remains a bottleneck to performance. PCI Express (PCIe)-based SSDs together with emerging host control interface standards address this interface bottleneck. SSDs with legacy storage interfaces are proving useful, and PCIe SSDs will further increase performance and improve responsiveness by connecting directly to the host processor.

2.2 SSD Architecture

Flash cards, USB keys and Solid State Disks are definitely the most known examples of electronic systems based on non-volatile memories, especially of NAND type (Sect. 2.4).

Several types of memory cards (CF, SD, MMC, ...) are available in the market [1–3], with different user interfaces and form factors, depending on the needs of the target application: e.g. mobile phones need very small-sized removable media like μSD.

SSDs are the emerging application for NAND. A SSD is a complete, small system where every component is soldered on a PCB and is independently packaged: NANDs are usually available in TSOP packages.

A basic block diagram of solid state disk is shown in Fig. 2.1. In addition to memories and a controller, there are usually other components. For instance, an

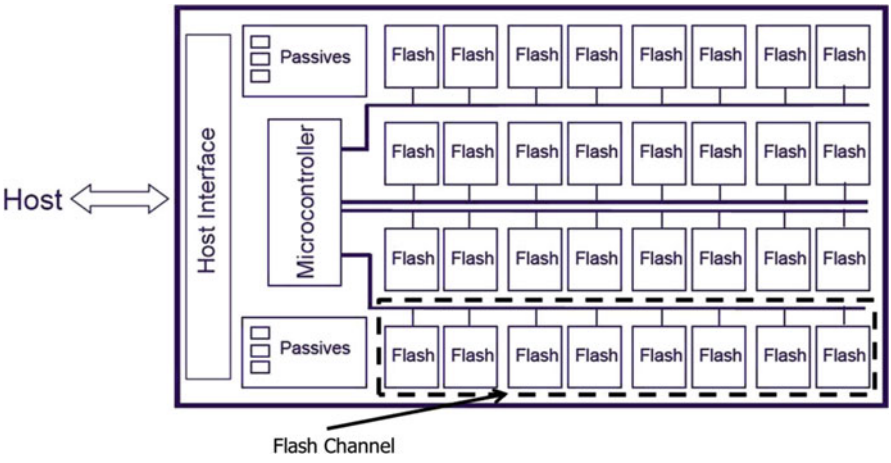


Fig. 2.1 Block diagram of a SSD

external DC-DC converter can be added in order to derive 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 a temp sensor for power management reasons. For data caching, a fast DDR memory is frequently used: during a write access, the cache is used for storing data before transfer to the Flash. The benefit is that data updating, e.g. in routing tables, is faster and does not wear out the Flash.

In order to improve performances, NANDs are organized in different Flash channels, as shown in Fig. 2.1.

2.3 Non-volatile Memories

Semiconductor memories can be divided into two major categories: RAM (*Random Access Memories*) and ROM (*Read Only Memories*): RAMs lose their content when power supply is switched off, while ROMs virtually hold it forever. A third category lies in between, i.e. NVM (*Non-Volatile Memories*), whose content can be electrically altered but it is also preserved when the power supply is switched off. NVMs are more flexible than the original ROM, whose content is defined during manufacturing and cannot be changed by the user anymore.

NVM's history began in the 1970s, with the introduction of the first EPROM memory (*Erasable Programmable Read Only Memory*). In the early 1990s, Flash memories came into the game and they started being used in portable products, like mobile phones, USB keys, camcorders, and digital cameras. Solid State Disk (SSD) is the latest killer application for Flash memories. It is worth mentioning that, depending on how the memory cells are organized in the memory array, it is possible to distinguish between NAND and NOR Flash memories. In this book we focus on NAND memories as they are one of the basic elements of SSDs. NOR architecture is described in great details in [4].

NAND Flash cell is based on the Floating Gate (FG) technology, whose cross section is shown in Fig. 2.2. A MOS transistor is built with two overlapping gates

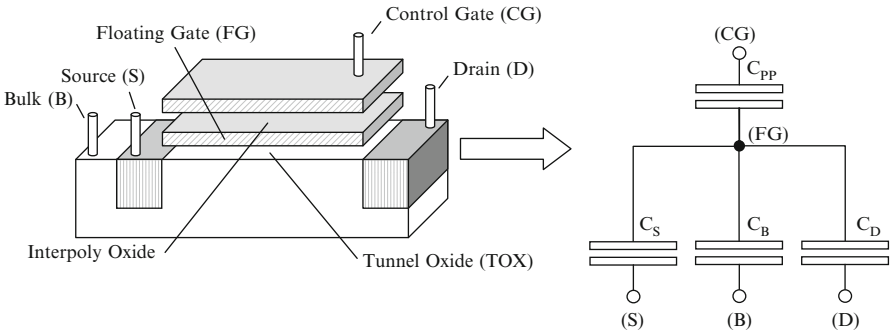


Fig. 2.2 Schematic representation of a floating gate memory cell (*left*) and the corresponding capacitive model (*right*)

rather than a single one: the first one is completely surrounded by oxide, while the second one is contacted to form the gate terminal. The isolated gate constitutes an excellent "trap" for electrons, which guarantees charge retention for years. The operations performed to inject and remove electrons from the isolated gate are called program and erase, respectively. These operations modify the threshold voltage V_{TH} of the memory cell, which is a special type of MOS transistor. Applying a fixed voltage to cell's terminals, it is then possible to discriminate two storage levels: when the gate voltage is higher than the cell's V_{TH}, the cell is on ("1"), otherwise it is off ("0").

It is worth mentioning that, due to floating gate scalability reasons, charge trap memories are gaining more and more attention and they are described in Chap. 5, together with their 3D evolution.

2.4 NAND Flash

2.4.1 NAND Array

A Flash device contains an array of floating-gate transistors: each of them acts as memory cell. In Single Level Cell (SLC) devices, each memory cell stores one bit of information; Multi-Level Cell (MLC) devices store 2 bits per cell.

The basic element of a NAND Flash memory is the NAND string, as shown in Fig. 2.3a. Usually, a string is made up by 32 (M_{C0} to M_{C31}), 64 or 128 cells connected in series. Two selection transistors are placed at the edges of the string: M_{SSL} ensures the connection to the source line. M_{DSL} connects the string to the bitline BL. The cell's control gates are connected through the wordlines (WLs). Figure 2.3b shows how the matrix array is built starting from the basic string. In the WL direction, adjacent NAND strings share the same WL, DSL, BSL and SL. In the BL direction, two consecutive strings share the bitline contact. Figure 2.4 shows a section of the NAND array along the bitline direction.

All the NAND strings sharing the same group of WL's form a Block. In Fig. 2.3b there are three blocks:

– BLOCK0 is made up by $WL_0 < 31{:}0 >$;
– BLOCK1 is made up by $WL_1 < 31{:}0 >$;
– BLOCK2 is made up by $WL_2 < 31{:}0 >$.

Logical pages are made up of cells belonging to the same WL. The number of pages per WL is related to the storage capabilities of the memory cell. Depending on the number of storage levels, Flash memories are referred to in different ways:

- SLC memories stores 1 bit per cell;
- MLC memories stores 2 bits per cell;
- 8LC memories stores 3 bits per cell;
- 16LC memories stores 4 bits per cell.

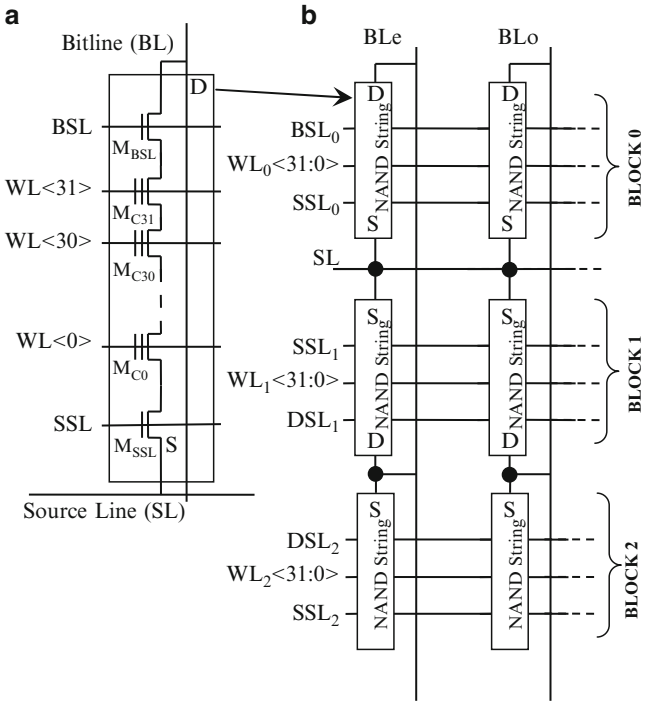


Fig. 2.3 NAND String (**a**) and NAND array (**b**)

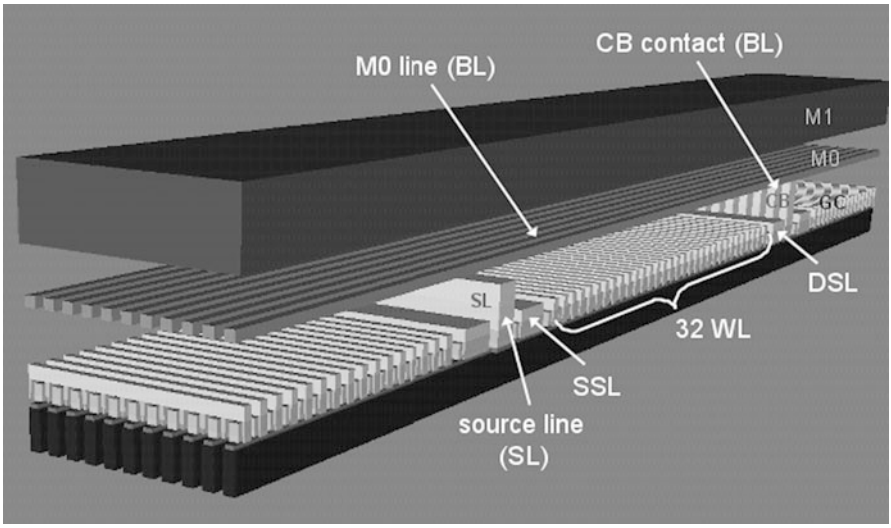


Fig. 2.4 NAND array section along the bitline direction

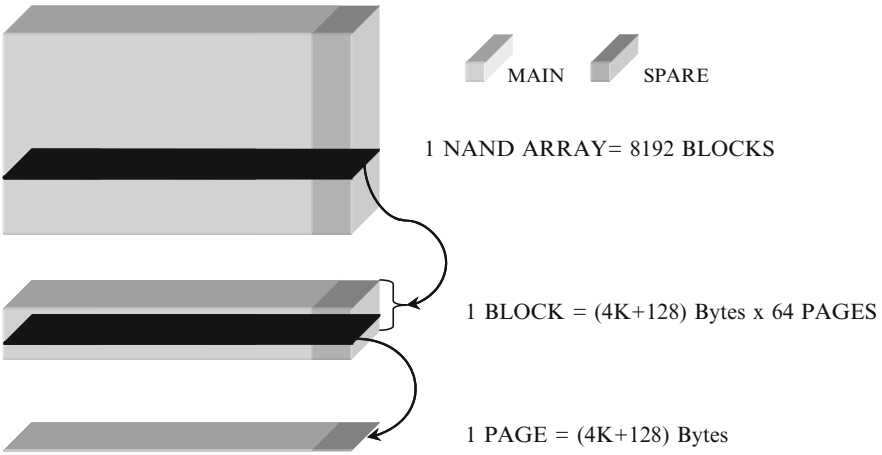


Fig. 2.5 32 Gbit memory logic organization

If we consider the SLC case with interleaved architecture (Chap. 6), even cells belong to the "even" page (BLe), while odd pages belong to the "odd" page (BL0). For example, a SLC device with 4 kB page has a WL of 32,768 + 32,768 = 65,536 cells. Of course, in the MLC case there are four pages as each cell stores one Least Significant Bit (LSB) and one Most Significant Bit (MSB). Therefore, we have MSB and LSB pages on even BL, and MSB and LSB pages on odd BL.

In NAND Flash memories, a logical page is the smallest addressable unit for reading and writing; a logical block is the smallest erasable unit (Fig. 2.5).

Each page is made up by main area (data) and spare area as shown in Fig. 2.5. Main area can be 4 kB, 8 kB or 16 kB. Spare area can be used for ECC and is in the order of hundred of bytes every 4 kB of main area.

Figure 2.5 shows the logic organization of a SLC device with a string of 32 cells, interleaving architecture, 4 kB page, and 128 bytes of spare.

NAND basic operations, i.e. read, program, and erase are described in Chap. 5 and Chap. 6 of this book.

2.4.2 NAND Interface

For many years, the asynchronous interface (Fig. 2.6) has been the only available option for NAND devices.

Asynchronous interface is described below.

- CE# : it is the Chip Enable signal. This input signal is "1" when the device is in stand-by mode, otherwise it is always "0".
- R/B# : it is the Ready/Busy signal. This output signal is used to indicate the target status. When low, the target has an operation in progress.

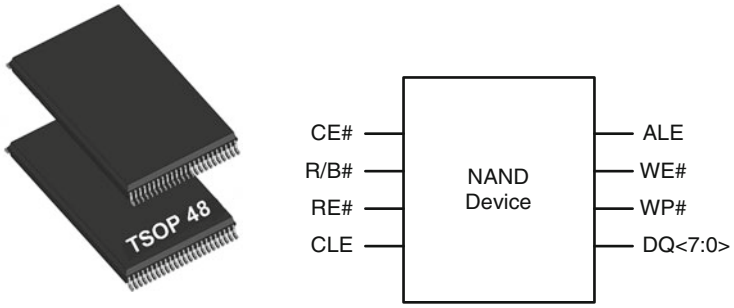


Fig. 2.6 TSOP package (*left*) and related pinout (*right*)

- RE# : it is the Read Enable signal. This input signal is used to enable serial data output.
- CLE : it is the Command Latch Enable. This input is used by the host to indicate that the bus cycle is used to input the command.
- ALE : it is the Address Latch Enable. This input is used by the host to indicate that the bus cycle is used to input the addresses.
- WE# : it is the Write Enable. This input signal controls the latching of input data. Data, command and address are latched on the rising edge of WE#.
- WP# : it is the Write Protect. This input signal is used to disable Flash array program and erase operations.
- DQ < 7:0> : these input/output signals represent the data bus.

As a matter of fact, this interface is a real bottleneck, especially looking at high performance systems like SSDs.

NAND read throughput is determined by array access time and data transfer across the DQ bus. The data transfer is limited to 40 MB/s by the asynchronous interface. As technology shrinks, page size increases and data transfer takes longer; as a consequence, NAND read throughput decreases, totally unbalancing the ratio between array access time and data transfer on the DQ bus. A DDR-like interface (Chap. 6) has been introduced to balance this ratio.

Nowadays two possible solutions are available on the market. ONFI (Open NAND Flash Interface) organization published the first standard at the end of 2006 [5]; other NAND vendors like Toshiba and Samsung use the Toggle-Mode interface. JEDEC [6] is now trying to combine these two approaches together.

Figure 2.7 shows ONFI pinout. Compared to the Asynchronous Interface, there are three main differences:

- RE# becomes W/R# which is the Write/Read direction pin;
- WE# becomes CLK which is the clock signal;
- DQS is an additional pin acting as the data strobe, i.e. it indicates the data valid window.

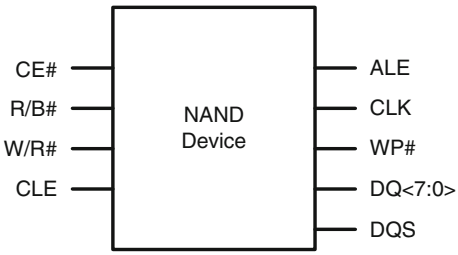


Fig. 2.7 Pinout of a NAND flash supporting ONFI interface

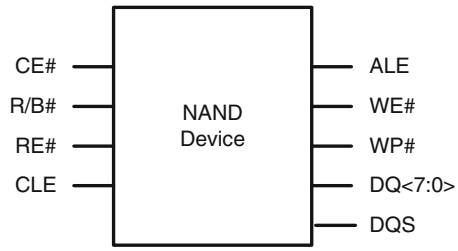


Fig. 2.8 Pinout of a NAND Flash supporting Toggle-Mode Interface

Hence, the clock (CLK) is used to indicate where command and addresses should be latched, while a data strobe signal (DQS) is used to indicate where data should be latched. DQS is a bi-directional bus and is driven with the same frequency as the clock. Toggle-Mode DDR interface uses the pinout shown in Fig. 2.8.

It can be noted that only the DQS pin has been added to the asynchronous interface. In this case, higher speeds are achieved increasing the toggling frequency of RE#.

2.5 Memory Controller

A memory controller has two fundamental tasks:

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

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 timing-critical tasks.

Generally speaking, the memory controller can be divided into four parts, which are implemented either in hardware or in firmware (Fig. 2.9).

Proceeding 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

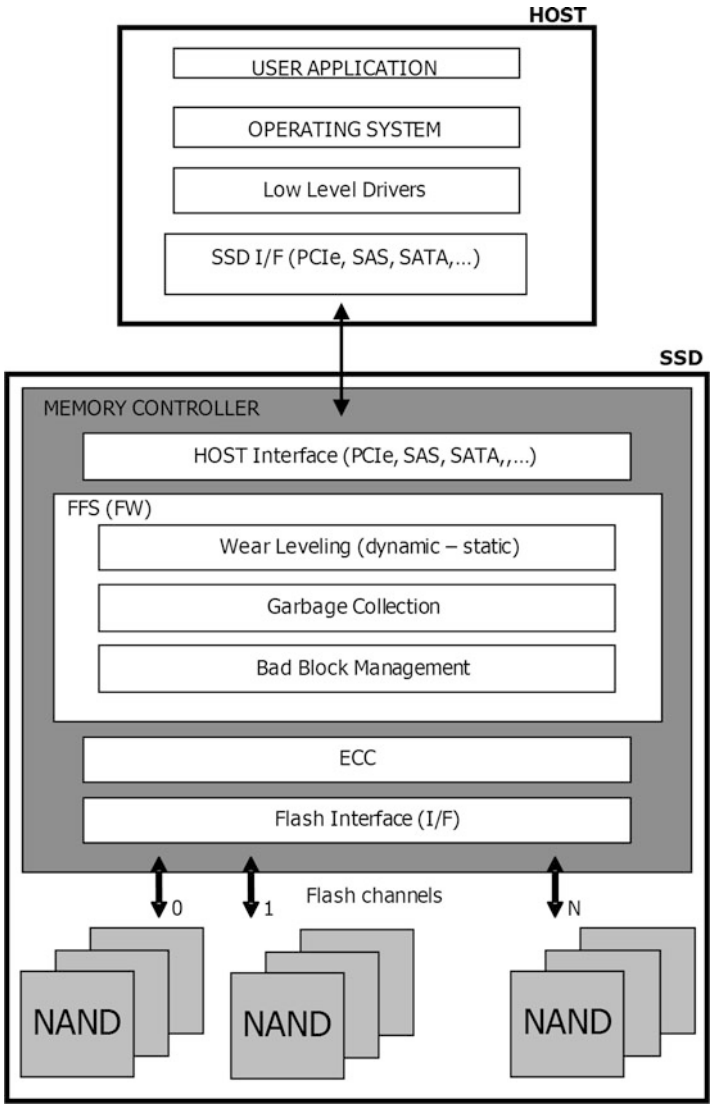


Fig. 2.9 High level view of a Flash controller

block is a mix of hardware – buffers, drivers, etc. – and firmware – command decoding performed by the embedded processor – 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) [7]: 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

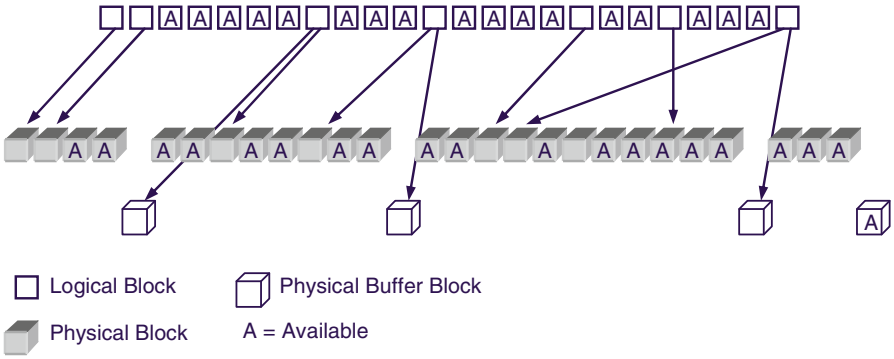


Fig. 2.10 Logical to physical block management

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 logical to physical (Flash Translation Layer or FTL) [8, 9], as shown in Fig. 2.10. The upper block row is the logical view of the memory, while the lower 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 therefore a second one (Buffer) is assigned to the same logical block.

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

2.5.1 Wear Leveling

Usually, not all the information stored within the same memory location change with the same frequency: some data are often updated while others remain always the same for a very 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 large 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 100 k cycles, which is reduced to 10 k when MLC NAND memories are used.

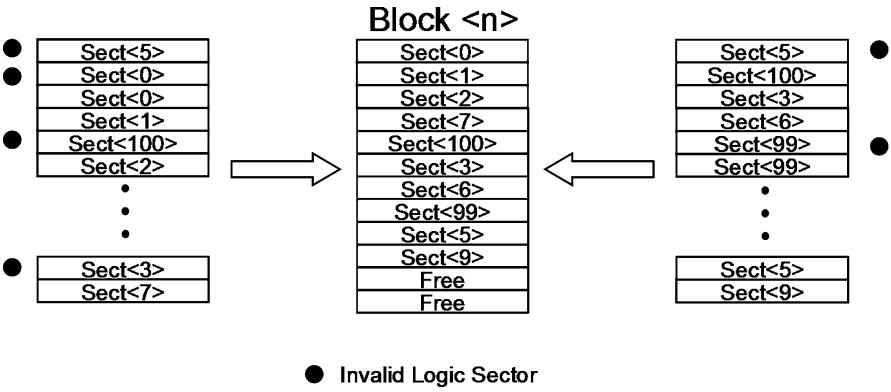


Fig. 2.11 Garbage collection

Wear Leveling techniques rely on the concept of logical to physical translation: that is, each time the host application requires updates to the same (logical) sector, the memory controller dynamically maps the sector onto a different (physical) sector, keeping track of the mapping either in a specific table or with pointers. The out-of-date copy of the sector is tagged as both 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; Static Wear Leveling can also be implemented, where every block, even the least modified, is eligible for re-mapping as soon as its aging deviates from the average value.

2.5.2 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, copies the latest valid copy into free sectors and erases such blocks (Fig. 2.11).

In order to minimize the impact on performance, garbage collection can be performed in background. The equilibrium generated by the wear leveling distributes wear out stress over the array rather than on single hot spots. Hence, the bigger the memory density, the lower the wear out per cell is.

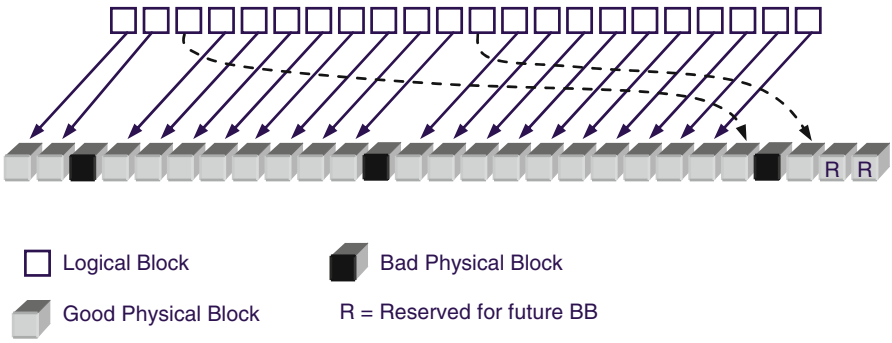


Fig. 2.12 Bad Block Management (BBM)

2.5.3 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 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. 2.12: this map is created during factory initialization of the memory card, thus containing the list of the bad blocks already present during the factory testing of the NAND Flash memory modules. Then it is updated during device lifetime whenever a block becomes bad.

2.5.4 Error Correction Code (ECC)

This task is typically executed by a specific hardware inside the memory controller. Examples of memories with embedded ECC are also reported [10–12]. Most popular ECC codes, correcting more than one error, are Reed-Solomon and BCH [13]. Chapter 10 gives an overview of how BCH is used in the NAND world, including an analysis of its detection properties, which are essential for concatenated architectures. The last section of Chap. 10 covers the usage of BCH in high-end SSDs, where the ECC has to be shared among multiple Flash channels.

With the technology shrink, NAND raw BER gets worse, approaching the Shannon limit. As a consequence, correction techniques based on soft information processing are required: LDPC (Low Density Parity Check) codes are an example of this soft information approach and they are analyzed in Chap. 11.

2.6 Multi-channel Architecture

A typical memory system is composed by several NAND memories. Typically, an 8-bit bus, usually called channel, is used to connect different memories to the controller (Fig. 2.1). It is important to underline that multiple Flash memories in a system are both a means for increasing storage density and read/write performance [14].

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. 2.13: in this way, the channel utilization is maximized by pipelining the data load phase; in fact, while the program operation takes place within a memory chip, the corresponding Flash channel is free. The total number of Flash channel is a function of the target applications, but tens of channels are becoming quite common. Figure 2.14 shows the impact of interleaving. As the reader can notice, given the same Flash programming time, SSD's throughput greatly improves.

The memory controller is responsible for scheduling the distributed accesses at 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. 6. Impact of DDR frequency on program throughput is reported in Fig. 2.15. 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 μs, at 50 MHz four NANDs can operate in interleaved mode, doubling the write throughput. Of course, power consumption has then to be considered.

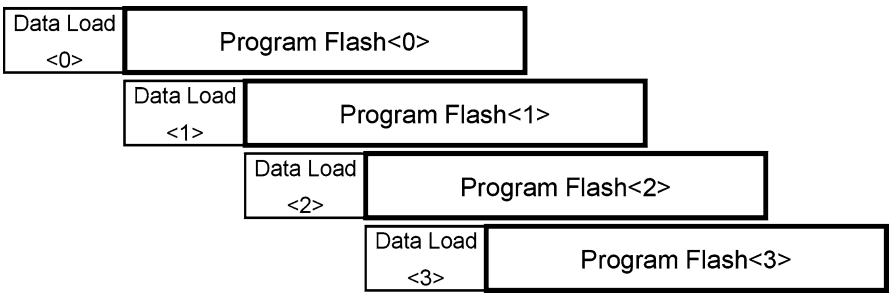


Fig. 2.13 Interleaved operations on one Flash channel

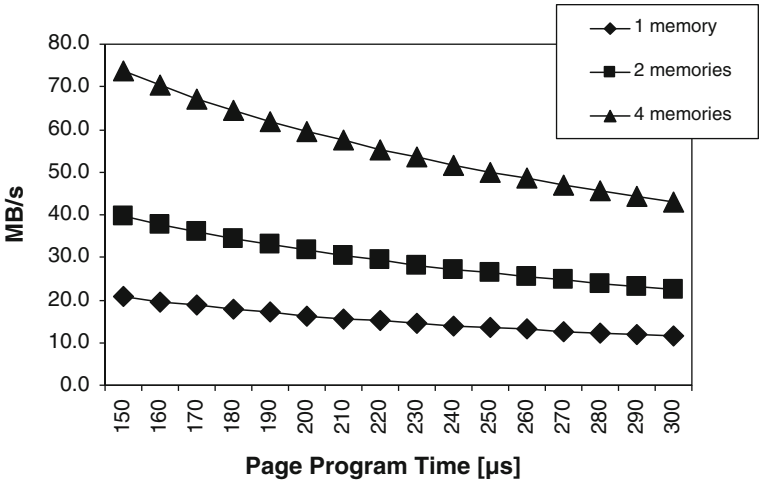


Fig. 2.14 Program throughput with an interleaved architecture as a function of the NAND page program time

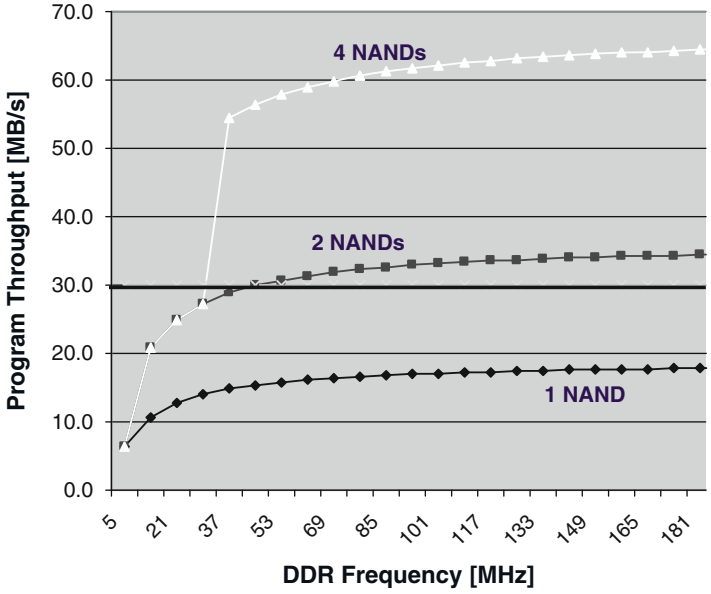


Fig. 2.15 Program throughput with an interleaved architecture as a function of the channel DDR frequency. 4 kB page program time is 200 µs

After this high level overview of the SSD architecture, let's move to the interface towards the host. PCI Express (PCIe) is the emerging interface for high performance SSDs.

2.7 What Is PCIe?

PCIe (Peripheral Component Interconnect Express) is a bus standard that replaced PCI and PCI-X. PCI-SIG (PCI Special Interest Group) creates and maintains the PCIe specification [15].

PCIe is used in all computer applications including enterprise servers, consumer personal computers (PC), communication systems, and industrial applications. Unlike 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. 2.16. The PCIe standard is constantly under improvement, with PCIe 3.0 being the latest version of the standard (Table 2.1).

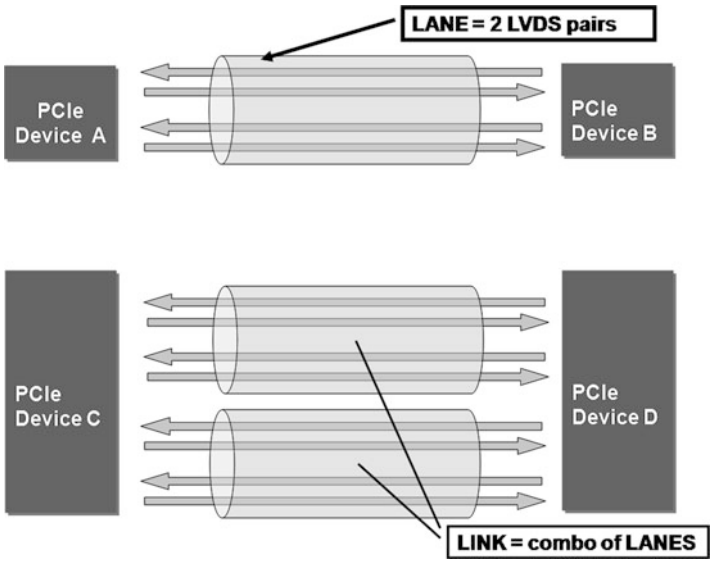


Fig. 2.16 PCI Express lane and link. In Gen2, 1 lane runs at 5 Gbps/direction; a 2-lane link runs at 10 Gbps/direction

Table 2.1 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

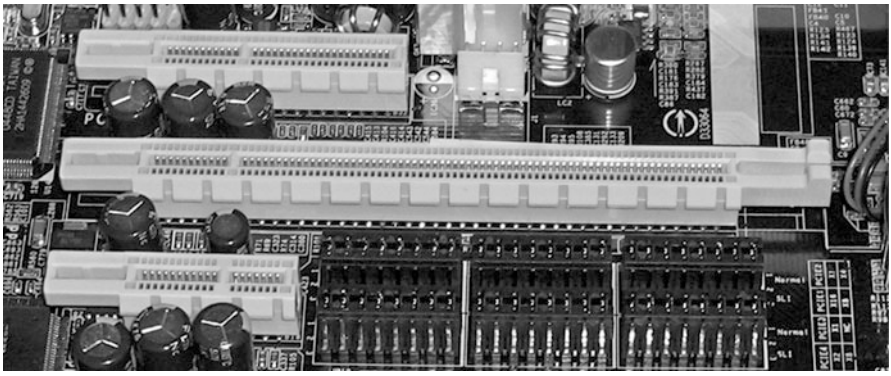


Fig. 2.17 Various PCIe slots. From top to bottom: PCIe × 4, PCIe × 16, PCIe × 1

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) [16]. Additionally, PCIe simplifies board design by utilizing serial technology, which eliminates wire count of 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: ×1, ×4, ×8, ×16, ×32 (Fig. 2.17). This allows PCIe to serve lower throughput, cost-sensitive applications as well as performance-critical applications.

There are basically three different types of devices in a native PCIe system as shown in Fig. 2.18 [17]: *Root Complexes* (RCs), PCIe switches, and *EndPoints* (EPs). A Root Complex should be thought of as a single processor sub-system with a single PCIe port, even though it consists of one or more CPUs, plus their associated RAM and memory controller. PCIe routes data based on memory address or ID, depending on the transaction type. Therefore, every device must be uniquely identified within the PCI Express tree. This requires a process called enumeration. During system initialization, the Root Complex performs the enumeration process to determine the various buses that exist and the devices that reside on each bus, as well as the required address space. The Root Complex allocates bus numbers to all the PCIe buses and configures the bus numbers to be used by the PCIe switches.

A PCIe switch behaves as if it were multiple PCI-PCI Bridges, as shown in the inset of Fig. 2.18. Basically, a switch decouples every UP and DP ports so that each link can work as a point-to-point connection.

Within a PCIe tree, all devices share the same memory space. RC is in charge of setting the Base Address Register (BAR) of each device.

In multi-RC systems, more than one processor sub-system exists within a PCIe tree. For example, a second Root Complex may be added to the system via the DP

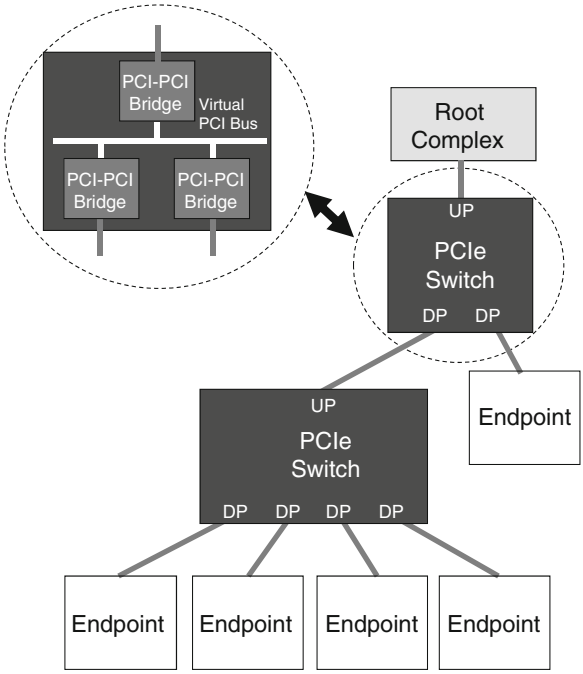


Fig. 2.18 PCIe tree topology

of a PCIe switch, possibly to act as a warm stand-by to the primary RC. However, an issue arises when the second RC also attempts the enumeration process: it sends out Configuration Read Messages to discover other PCIe devices on the system. Unfortunately, configuration transactions can only move from UP to DP. A PCIe switch does not forward configuration messages that are received on its DP. Thus, the second RC is isolated from the rest of the PCIe tree and will not detect any PCIe devices in the system. So, simply adding processors to a DP of a PCIe switch will not provide a multi-Root Complex solution.

One method of supporting multiple RCs is to use a *Non-Transparent Bridging* (NTB) function to isolate the address domains of each of the Root Complexes [18]. NTB allows two Root Complexes or PCIe trees to be interconnected with one or more shared address windows between them.

In other words, NTB works like an address translator between two address domains. Of course, multiple NTBs can be used to develop multi-RC applications. An example of PCIe switch with embedded NTB functions is shown in Fig. 2.19: an additional bus, called NT Interconnect, is used for exchanging Transaction Layer Packets among RCs.

PCIe uses a packet-based layered protocol, consisting of a transaction layer, a data link layer, and a physical layer, as shown in Fig. 2.20.

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 they are reliably delivered between two endpoints (devices A

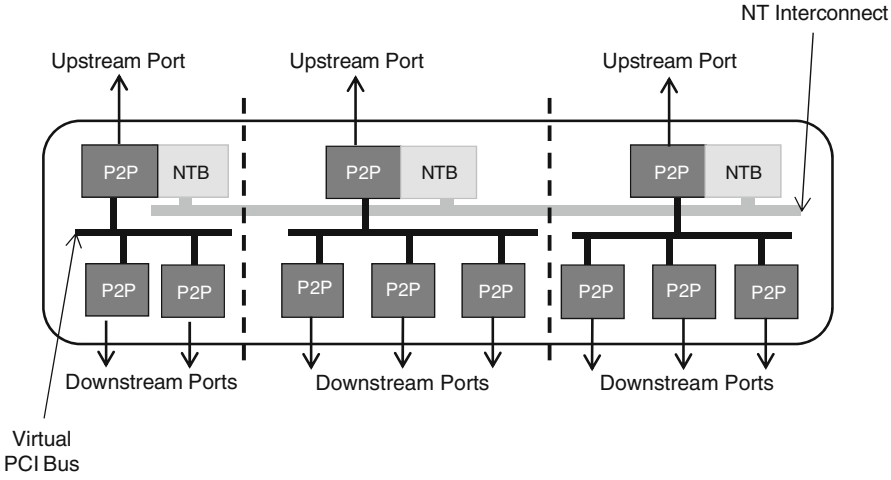
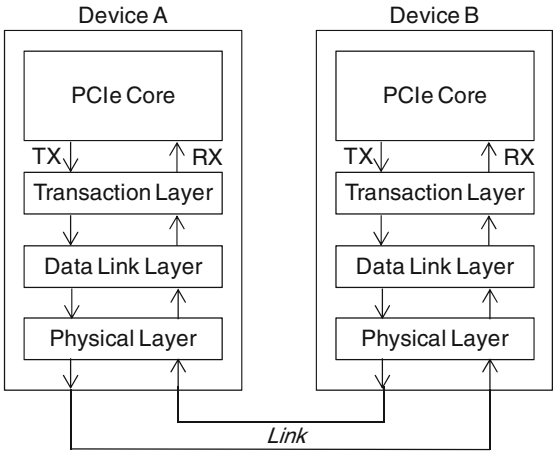


Fig. 2.19 PCIe switch with multiple NTB functions

Fig. 2.20 PCIe Layered architecture



and B in Fig. 2.5). 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. 2.21. Header and Data Payload are TLP's core information: Transaction Layer assembles this section based on the data received

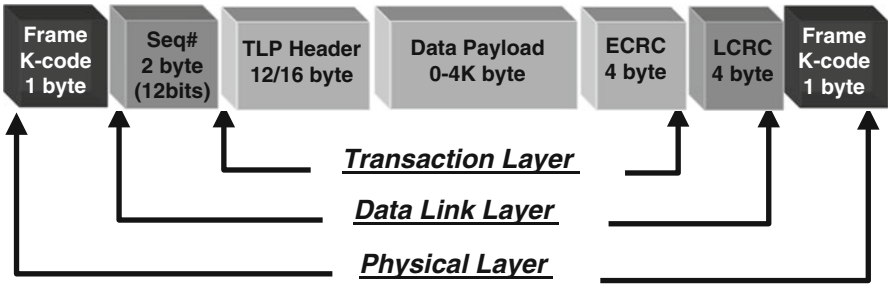


Fig. 2.21 Transaction Layer Packet (TLP) assembly

from the application software layer. An optional End-to-End CRC (ECRC) field is 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 character of 1 byte each to the packet. Finally, the packet is encoded and differentially transmitted on the Link using the available number of 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.

2.8 The Need for Storage Speed

The real issue at hand is the need for storage technology that can match the exponential ramp in processor performance over the past two decades. Processor vendors have continued to ramp the performance of individual processor cores, to combine multiple cores on one IC, and to develop technologies that can closely-couple multiple ICs in multi-processor systems. Ultimately, all of the cores in such a scenario need access to the same storage subsystem.

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 (IOPS/W) in power consumption. The ramping multi-processing 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 feeds 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. The DRAM and cache access-time and data-transfer performance has scaled to match the processor performance.

The disconnect has come in the performance gap that exist between DRAM and rotating storage in terms of access time and data rate. Disk-drive vendors have done a great job of designing and manufacturing higher-capacity, lower-cost-per-Gbyte disk drives. But the drives inherently have limitations in terms of how fast they can access data and then how fast they can transfer that data into DRAM.

Access time depends on how fast a hard drive can move the read head over the required data track on a disk, and the rotational latency for the sector where the data is located to move under the head. The maximum transfer rate is dictated by the rotational speed of the disk and the data encoding scheme that together determine the number of bytes per second 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 that mine data in applications such as customer relationship management require highly random access to data. Cloud computing also has a random element and the random issue in general is escalating with technologies such a virtualization expanding the scope of different applications that a single system has active at any one time. Every microsecond of latency relates directly to money lost and less efficient use of the processors and the power dissipated by the system.

Fortunately Flash memory offers the potential to plug the performance gap between DRAM and rotating storage. 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 Mbytes/s and faster access to random data, resulting in better cost-per-IOPS compared to rotating storage.

Ramping Flash capacity and reasonable cost has led to a growing trend of SSDs that package Flash in disk-drive-like form factors. Moreover, the SSDs have most often utilized disk-drive interfaces such as SATA (serial ATA) or SAS (serial attached SCSI).

2.9 Why PCIe for SSD Interface?

The disk-drive form factor and interface allows IT vendors to substitute an SSD for a magnetic disk drive seamlessly. There is no change required in system hardware or driver software. An IT manager can simply swap to an SSD and realize significantly better access times and somewhat faster data-transfer rates.

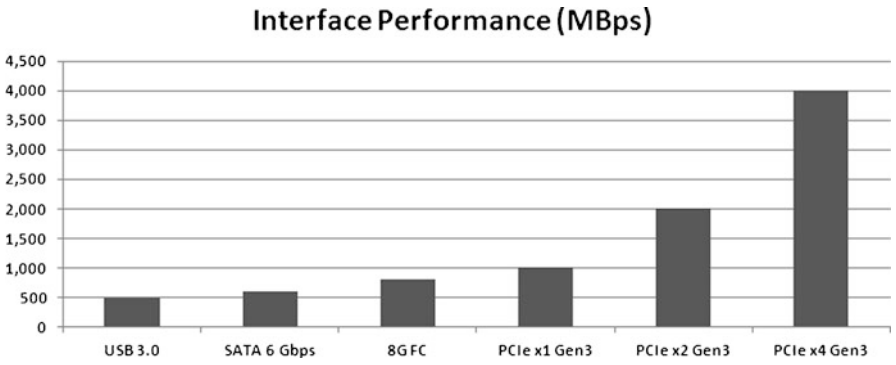


Fig. 2.22 Interface performance. PCIe improves overall system performance by reducing latency and increasing throughput

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. 2.22). Most mainstream systems today are migrating to third-generation SATA and SAS that support 600 Mbytes/s throughput, and drives based on those interfaces have already found usage in enterprise systems. While those data rates support the 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 Flash devices 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, shipping in volume in 2012, can deliver 4 GByte/s data rates. Simply put, PCIe affords the needed 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 could 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 eliminating the architectural layer associated with the legacy storage infrastructure. The compelling performance of PCIe SSDs has resulted in system manufacturers placing PCIe SSDs in servers as well as in storage arrays to build tiered storage systems (Fig. 2.23) that accelerate applications while improving cost-per-IOPS (*Input/Output Operations per Second*).

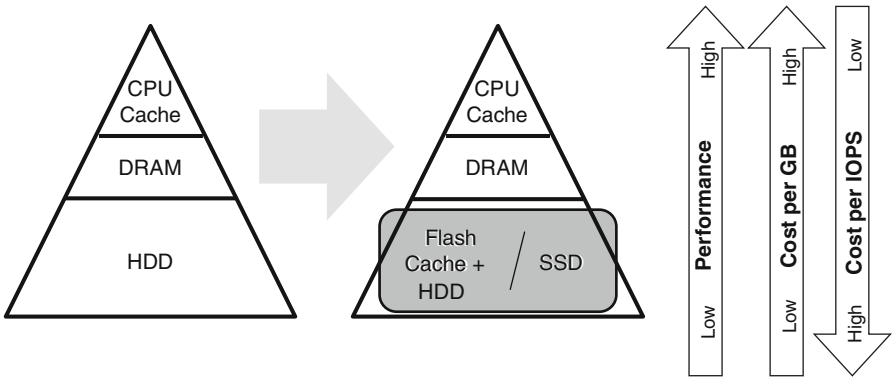


Fig. 2.23 Enterprise memory/storage hierarchy paradigm shift

Moving storage to a PCIe link brings additional challenges to the system designer. As mentioned earlier, the SATA- and SAS-based SSD products have maintained software compatibility and some system designers are reluctant to give up that advantage. Any PCIe storage implementation will create the need for some new driver software.

Despite the software issue, the move to PCIe storage in enterprises is already happening. Performance demands in the enterprise are mandating this transition. There is no other apparent way to deliver improving IOPS, IOPS/W, and IOPS per dollar characteristics that IT managers are demanding.

The benefits of using PCIe as a storage interconnect are clear. You can achieve over 6x the data throughput relative 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. 2.24.

So the question the industry faces isn't really whether to use PCIe to connect with Flash storage, but how to do so. There are a number of options with some early products already in the market.

Let's now take a deeper look at PCIe-based SSD architectures.

2.10 PCIe SSD Implementations

The simplest PCIe SSD implementations can utilize legacy Flash memory controller ICs that while capable of controlling memory read and write operations, have no support for the notion of system I/O. Such Flash controllers would typically work behind a disk interface IC in existing SATA- or SAS-based SSD products (Fig. 2.25).

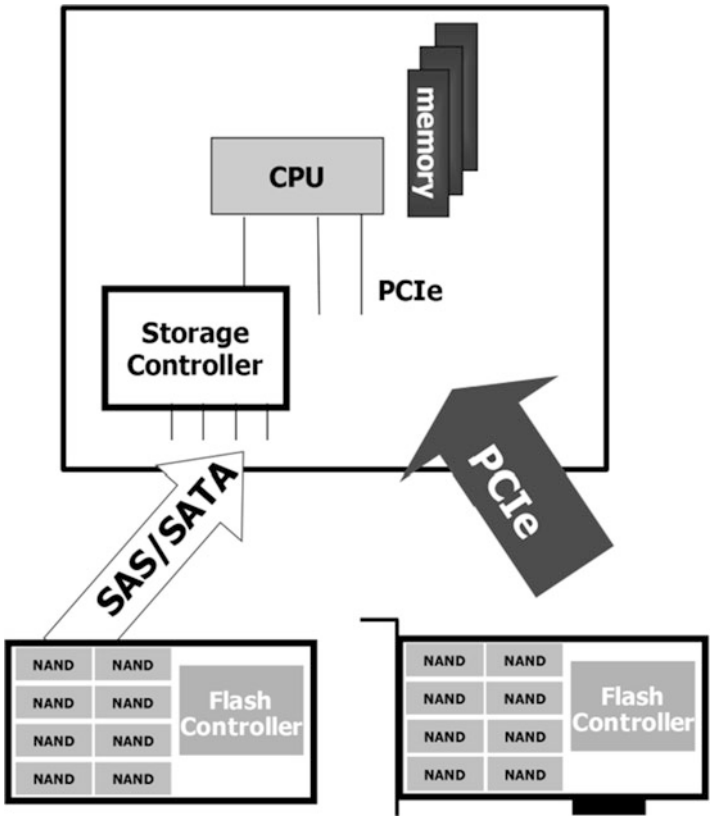


Fig. 2.24 PCIe SSD vs. SAS/SATA SSD

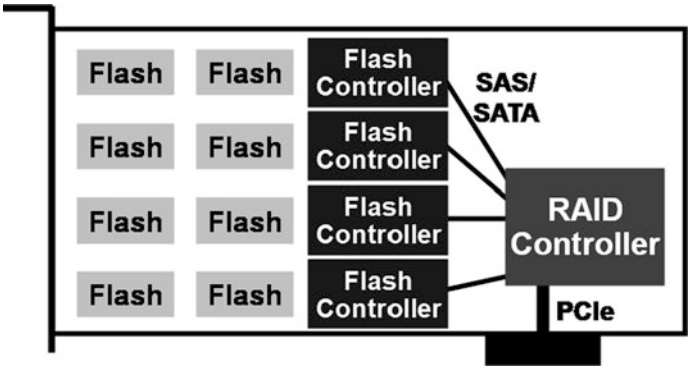


Fig. 2.25 RAID-based PCIe SSDs not optimized for performance/power

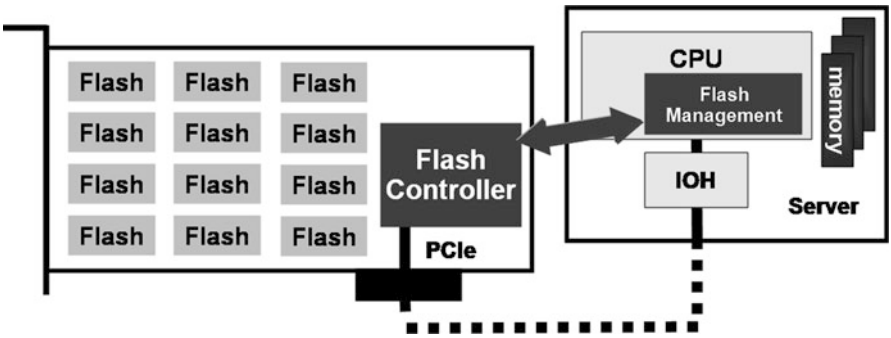


Fig. 2.26 Running flash management algorithms on the host drains the host CPU/RAM resources

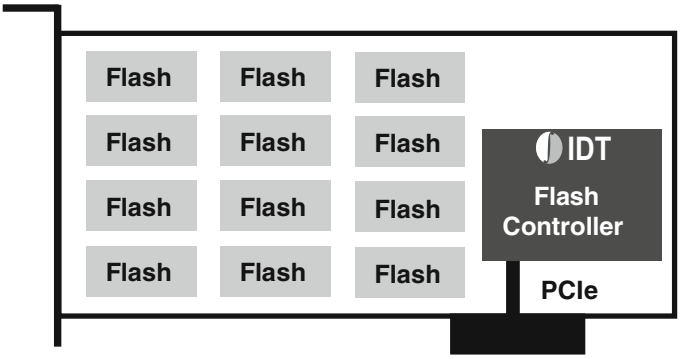


Fig. 2.27 Native PCIe Flash Controller improves performance, while reducing cost & complexity

Alternatively, it is possible to run Flash-management software on the host processor to enable a simple Flash controller to function across a PCIe interconnect (Fig. 2.26).

That approach has several drawbacks. First it consumes host processing and memory resources that ideally would be handling more IOPS. Second it requires proprietary drivers and raises OEM qualification issues. And third it doesn't deliver a bootable drive because the system must be booted for the Flash-management software to execute and enable the storage scheme.

Clearly, these designs have found niche success. These products are used by early adopters as caches for hard disk drives rather than mainstream replacements of high-performance disk drives.

Longer term, more robust and efficient PCIe SSD designs are relying on a complex SoC that natively supports PCIe, integrates Flash controller functionality, and that completely implements the storage-device concept (Fig. 2.27). Such a product offloads the host CPU and memory of handling Flash management, and ultimately enables standard OS drivers that support plug-and-play operations just as we enjoy with SATA and SAS today.

2.11 Standards Driving Broader Adoption of PCIe SSDs

New standards will ultimately deliver plug-and-play functionality for PCIe-connected SSDs. Table 2.2 is a summary of the industry effort in this direction.

The NVM Express (NVMe) 1.0 specification, developed cooperatively by more than 80 companies from across the industry, was released in March, 2011, by the NVMHCI Work Group – now more commonly known as the NVMe Work Group. The specification defines an optimized register interface, command set, and feature set for PCIe SSDs. The goal of the standard is to help enable the broad adoption of PCIe-based SSDs, and to provide a scalable interface that realizes the performance potential of SSD technology now and into the future. By maximizing parallelism and eliminating complexity of legacy storage architectures, NVMe supports future memory developments that will drive latency overhead below one microsecond and SSD IOPS to over one million. The NVMe 1.0 specification may be downloaded from www.nvmexpress.org.

The NVMe specification is specifically optimized for multi-core system designs that run many threads concurrently with each thread capable of instigating I/O operations. Indeed it's optimized for just the scenario that IT managers are hoping to leverage to boost IOPS. NVMe specification can support up to 64 k I/O queues with up to 64 k commands per queue. Each processor core can implement its own queue.

In June, 2011, the NVMe Promoter Group was formed to enable the broad adoption of the NVMe Standard for PCIe SSDs. Seven companies hold permanent seats on the board: Cisco, Dell, EMC, IDT, Intel, NetApp, and Oracle. NVMe supporters include IC manufactures, Flash-memory manufacturers, operating-system vendors, server manufacturers, storage-subsystem manufacturers, and network-equipment manufacturers.

SCSI Express is another industry initiative that is planning to address the host control interface of PCIe SSDs, with support for legacy enterprise storage command set. SCSI Express uses SCSI over PCIe (SOP) and PCIe architecture queuing interface (PQI) model being defined within the T10 Technical Committee.

Table 2.2 Industry standards for PCIe SSDs

Standard	Status	Benefits
NVMe	Spec 1.0 released March 2011 Standard OS driver implementations available (Windows and Linux) 80+ members of NVMe working group	Designed for servers and clients
SCSI over PCIe (SOP)	Under development in T10	Utilizes SCSI software infrastructure
SSD Form Factor Working Group [19] (2.5" with new connector)	Spec 1.0 released December 2011	Serviceability Hot-plugability

The NVMe and SOP standards do not address the subject of form factors for SSDs and that's another issue that has been addressed through another working group.

In enterprise-class storage, devices such as disk drives and SSDs are typically externally accessible and support hot-plug capabilities. In part the hot-plug capability was required due to the fact that disk drives are mechanical in nature and generally fail sooner than ICs. The hot-plug feature allows easy replacement of failed drives.

With SSDs, IT managers and storage vendors will want to stay with an externally-accessible modular approach. Such an approach supports easy addition of storage capacity either by adding SSDs, or replacing existing SSDs with more capacious ones.

Indeed another standards body was formed to address the form factor issue. The SSD Form Factor Working Group is focused on promoting PCIe as an SSD interconnect. The working group is driven by five Promoter Members including Dell, EMC, Fujitsu, IBM, and Intel.

Enterprise SSD Form Factor Version 1.0 specification was released in December 2011, focusing on three areas:

- a connector specification that will support PCIe as well as SAS/SATA;
- a form factor that builds upon the current 2.5-in. standard while supporting the new connector definition and expanding the power envelope in support of higher performance;
- the support for hot plug capability.

The building blocks are all falling into place for broader usage of PCIe-connected SSDs and deliverance of the performance improvements that the technology will bring to enterprise applications. And while our focus has been more on the enterprise, the NVMe standard will also trickle down to client systems, offering a performance boost even in notebook PCs while reducing cost and system power. The standard will drive far more widespread use of PCIe SSD technology as compatible ICs and drivers come to fruition.

References

1. www.mmca.org
2. www.compactflash.org
3. www.sdcard.com
4. G. Campardo, R. Micheloni, D. Novosel, *VLSI-Design of Non-Volatile Memories* (Springer, Berlin, 2005)
5. www.onfi.org
6. www.jedec.org
7. A. Kawaguchi, S. Nishioka, H. Motoda, A flash-memory based file system, in *Proceedings of the USENIX Winter Technical Conference*, 1995, pp. 155–164
8. J. Kim, J.M. Kim, S. Noh, S.L. Min, Y. Cho, A space-efficient flash translation layer for compactflash systems. IEEE Trans. Consum. Electron. **48**(2), 366–375 (May 2002)

9. S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S.-W. Park, H.-J. Songe, FAST: A log-buffer based FTL scheme with fully associative sector translation, in *2005 US-Korea Conference on Science, Technology, & Entrepreneurship*, Seoul, Aug 2005
10. T. Tanzawa, T. Tanaka, K. Takekuchi, R. Shirota, S. Aritome, H. Watanabe, G. Hemink, K. Shimizu, S. Sato, Y. Takekuchi, K. Ohuchi, A compact on-chip ECC for low cost Flash memories. IEEE J. Solid-State Circuits **32**(May), 662–669 (May 1997)
11. G. Campardo, R. Micheloni et al., 40-mm^2 3-V-only 50-MHz 64-Mb 2-b/cell CHE NOR Flash memory. IEEE J Solid-State Circuits. **35**(11), 1655–1667 (Nov 2000)
12. R. Micheloni et al., A 4Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36 MB/s system read throughput, in *IEEE International Solid-State Circuits Conference Dig. Tech. Papers*, Feb 2006, pp. 142–143
13. R. Micheloni, A. Marelli, R. Ravasio, *Error Correction Codes for Non-Volatile Memories* (Springer, Dordrecht, 2008)
14. C. Park et al., A high performance controller for NAND flash-based Solid State Disk (NSSD), in *IEEE Non-Volatile Semiconductor Memory Workshop NVSMW*, Feb 2006, pp. 17–20
15. www.pcisig.com
16. R. Budruk, D. Anderson, T. Shanley, Mindshare, *PCI Express System Architecture* (Addison-Wesley, Boston, 2003)
17. K. Kong, *Enabling Multi-peer Support with a Standard-Based PCI Express Multi-ported Switch*, White Paper, www.idt.com Jan 2006
18. K. Kong, Non-Transparent Bridging with IDT 89HPES32NT24G2 PCI Express NTB Switch, AN-724, www.idt.com (Sept 2009)
19. www.ssdformfactor.org

http://www.springer.com/978-94-007-5145-3

Inside Solid State Drives (SSDs)

Micheloni, R.; Marelli, A.; Eshghi, K.

2013, XVIII, 382 p., Hardcover
ISBN: 978-94-007-5145-3