AHCI and NVMe as interfaces for SATA Express™ Devices

By Dave Landsman, SanDisk
and Don Walker, Dell

11/15/2013

Table of Contents

1 Introduction

The Serial ATA (SATA™) storage interface has evolved almost unrecognizably from its beginnings as IDE/ATA (Parallel ATA - PATA) in the late 1980's. SATA is now the most widely used hard disk drive (HDD) interface in the global storage market. SATA has held up incredibly well to the performance demands of HDDs; however, in 2009-2010, solid state drives (SSDs) began to outstrip the capabilities of even 3rd generation SATA, with its 6Gb/s bandwidth.

It was considered impractical to take the SATA interface above 6Gb/s, and so the client storage market needed a way forward to accommodate the huge base of legacy SATA devices, the still vibrant market for new SATA devices, including SATA-based SSDs, as well as to incorporate new classes of very high performance devices, such as PCI Express (PCIe)-based SSDs.

SATA Express™ was created by SATA-IO in 2011 to enable a path beyond SATA 6Gb (see "Why SATA Express") for PCIe-based client SSD's.

The hardware model of SATA Express is that a host can accept a legacy SATA device, or a SATA Express (PCIe) device, per figure 1.

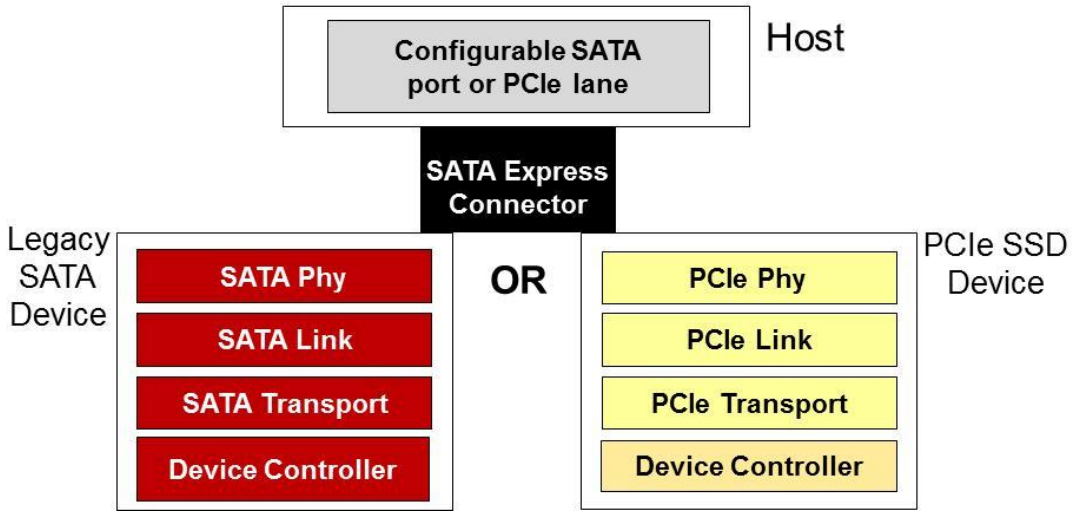

Figure 1 – SATA Express HW System Overview

While the SATA Express hardware model enables SATA and PCIe devices to be used in the same platform electro-mechanical environment, there is a separate issue of the logical device interface to be used for SATA Express PCIe devices. There are two standards-based choices for the SATA Express PCIe device interface, the Advanced

Host Controller Interface (AHCI) and NVM Express (NVMe). This paper describes the relationship of the AHCI and NVMe interfaces, in the SATA Express environment, as PCIe device interfaces for SATA Express devices. It also provides some technical background and comparison of the performance aspects of both interfaces.

2 SATA Express Interface Architecture

Figure 2 describes the SATA Express software architecture and how legacy SATA, SATA Express/AHCI, and SATA Express/NVMe relate to one another.

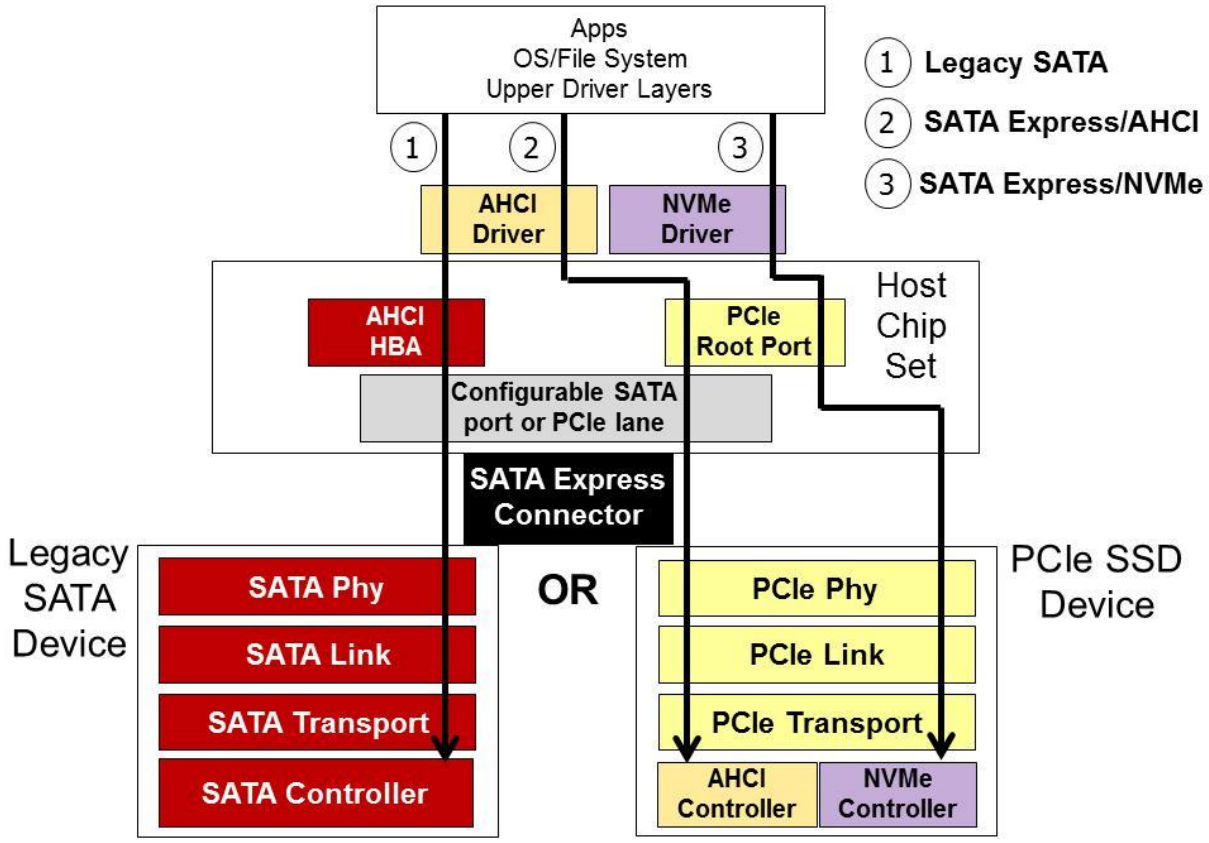

Figure 2 – SATA Express High Level Architecture

In a legacy SATA implementation (left side of figure), the AHCI interface is implemented as a Host Bus Adapter (HBA), and often built into the host chip set. In this case, applications talk to the host side of the AHCI HBA via PCIe or an internal system bus. The device side of the HBA connects to the SATA device over the legacy SATA Link and PHY channel.

In a SATA Express PCIe case (right side of figure), two standard device interface options are possible, AHCI and NVMe. In both cases, the SATA Express device appears to the system as an attached PCIe device through the host's PCIe Root Complex Port. The difference is mainly in the device driver and, of course, the device controller in the device itself.

In path (2) the PCIe endpoint, the SSD, appears to the system as an AHCI device and is accessed through the AHCI/SATA device driver, supported in nearly all client platforms by a standard in-box device driver.

In path (3), the endpoint, the SSD, appears to the system as an NVMe device and is accessed through the NVMe device driver. In the case of NVMe, in-box driver support is available, but still evolving (see Driver Availability).

3 AHCI and NVMe as SATA Express Device Interfaces

To compare AHCI and NVMe as device interfaces for SATA Express, it is necessary to provide some background on the evolution of AHCI.

AHCI was originally created to provide a host controller interface to SATA hard disk drives (HDDs) enabling queuing, an asynchronous command-response mechanism, power management, a better architected application programming interface, and more. It was a significant improvement over the legacy SATA interface, which had evolved from IDE.

As noted in Figure 2 (path 1) and shown here in figure 3, in a legacy SATA implementation, the AHCI interface is implemented as an HBA. As an HBA, the AHCI controller can support up to thirty two (32) ports (physical connections), each of which are exposed to the host via dedicated registers inside the HBA.

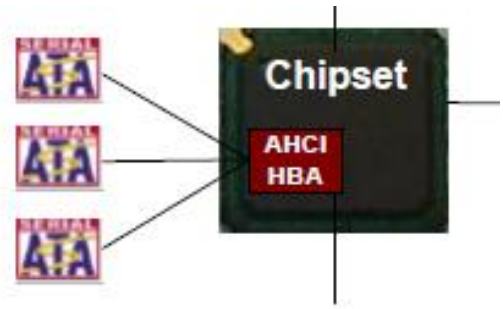

Figure 3 – AHCI as HBA

In contrast, when implemented as a SATA Express/AHCI device, the AHCI controller is integrated into the device, as shown in Figure 4. In the terms of the AHCI theory of operation, an AHCI controller in a SATA Express/AHCI device has a single physical host port.

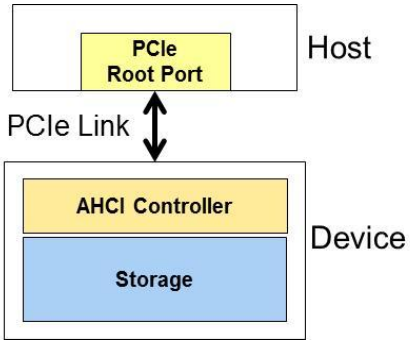

Figure 4 – AHCI as SATA Express Device Controller

When comparing NVMe and AHCI as interfaces for SATA Express devices, the comparison must be between NVMe and AHCI, as attached PCIe device controllers, not between NVMe as an attached PCIe device controller and AHCI as an (up to 32-port) HBA. In the following sections, all discussion of AHCI is as an attached PCIe device controller with a single AHCI port.

4 AHCI Overview

The AHCI controller implemented in a SATA Express device is exposed to the host through a set of memory-mapped PCIe registers. The key device specific registers are:

- Capabilities registers - Describe support for optional features of the AHCI interface as well as optional features of the attached SATA devices. Examples of the former include 64 bit addressing and power state capabilities. Examples of the latter include queued commands, staggered spin-up, and SATA interface speeds.

- Configuration registers - Allow the host to configure the HBA's operational modes. Examples include enable/disable AHCI mode in the HBA, enable/disable interrupts, and reset HBA.

- Status registers - Report events and status, such as pending interrupts, implemented ports, timeout values, interrupt/command coalescing, and HBA readiness.

- Port registers – Used by the host to set up commands for the controller. As noted above, AHCI in a SATA Express device implements only one set of port registers.

AHCI commands are processed using a series of data structures (Figure 5). In an AHCI HBA, a Command List is associated with each of the up to 32 ports in a device. In a SATA Express/AHCI device, which permits only a single port, only a single Command List is maintained. In all cases, each Command List can reference up to 32 individual commands. Commands are contained in structures called Command Tables.

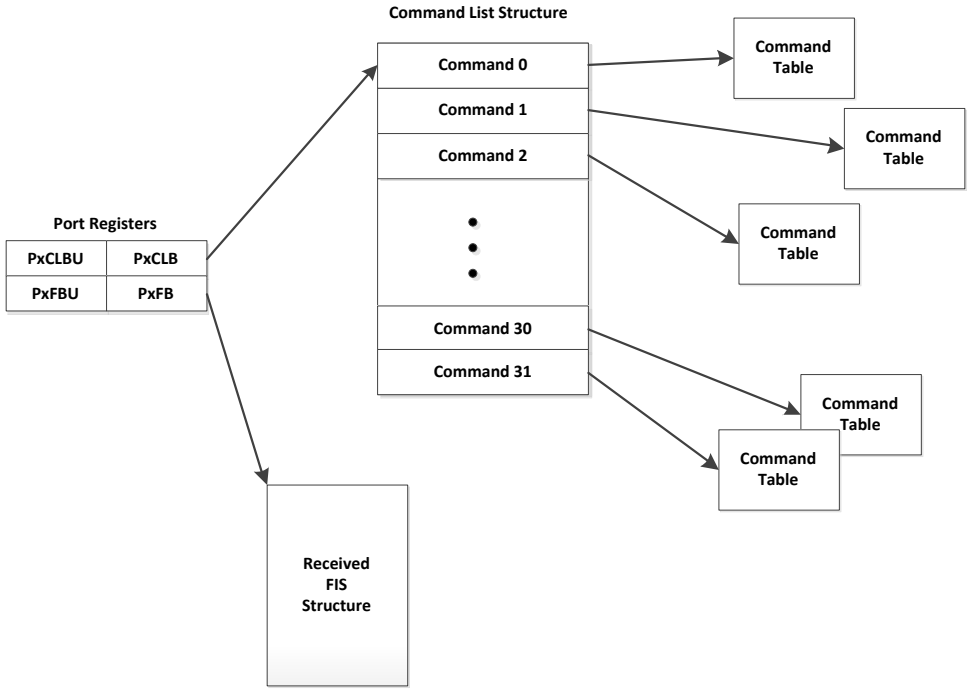

Figure 5 – AHCI Command Data Structures

Once host memory has been allocated for the Command List and Command Table, the Command Table is initialized with the command and other metadata associated with the command (e.g., Read or Write command, associated scatter/gather lists, etc). Prior to initializing the Command List and Command Table entries, host software (typically the AHCI driver) ensures that any previous commands occupying the Command List slot(s) have completed.

The register operations needed to construct and launch a command cannot be cached; thus, a processor must generate a PCIe bus access any time a new read or write is performed. This overhead can be amortized over as many as 32 commands.

Overall, for an AHCI-based SATA Express device, the execution of a non-queued operation requires six (6) register accesses, and the execution of a queued operation requires nine (9) register accesses.

Another aspect of AHCI is that a port's Command List and Command Table resources are not sharable between multiple threads. If two or more threads or processes wish to access the same device, then a mechanism (i.e., a lock, mutex, or other semaphore) must be used to coordinate access to the device. Such coordination takes place at a layer above the device interface.

5 NVMe Overview

The register sets of an NVMe interface are conceptually very similar to those of AHCI:

- Capabilities and Configuration registers – Declaring support for, and enabling of, features such as submission/completion queue size, arbitration mechanisms, and command sets supported.

- Status registers – Enable reporting of global device controller status, such as transitional device states during an orderly shutdown and global error/exception conditions.

Unlike AHCI, the NVMe architecture did not need to consider legacy interfaces and associated software stacks with which backward compatibility had to be maintained. This has allowed an interface design that is optimized to take advantage of those very characteristics that make NAND flash-based devices so attractive.

Powerful Queuing Model

a) The interface allows multiple command submission/completion pathways, or IO channels, based on submission/completion queue pairs. An IO channel is composed of one or more submission queues associated with a particular completion queue.

b) Multiple submission and completion queues can be created dynamically. Up to 64K submission and completion queues can be supported, each up to 64K entries deep.

c) Priorities can be assigned to queues. NVMe provides support for "simple round robin", "weighted round robin with urgent priority" or a vendor-defined priority scheme.

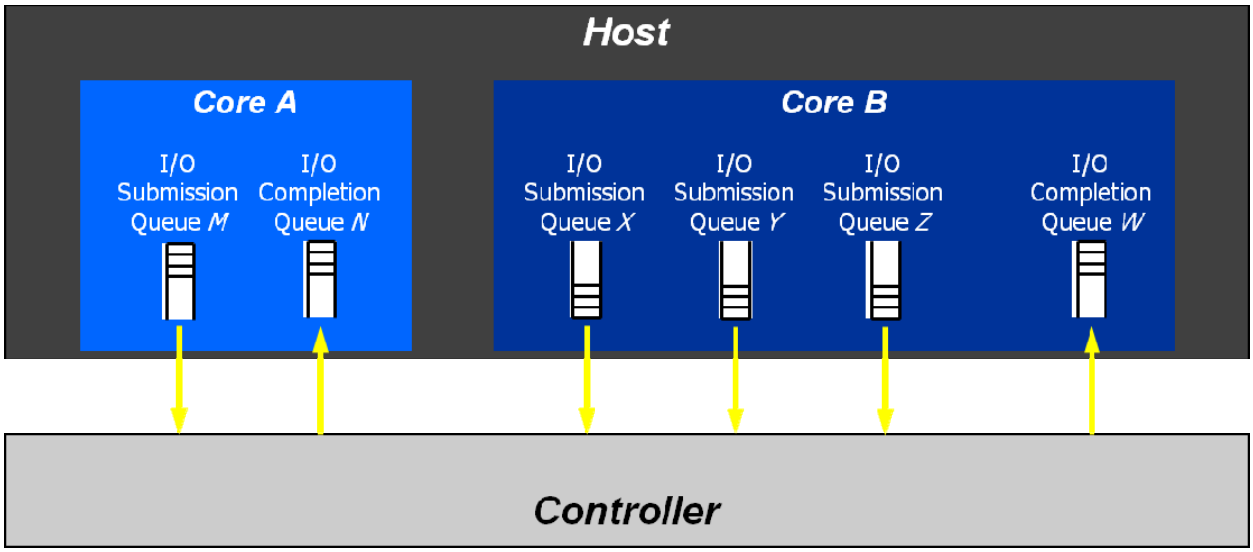

Figure 6 – NVMe Queuing/IO Channel Structure

Efficient Transport

a) The IU (Information Unit), the packet of information sent over the PCIe bus carrying commands and status, is of fixed size, supporting easy parsing of the information stream.

b) Non-cacheable PCIe register reads/writes, used to move commands and status in and out of the command and status queues are kept to a minimum.

c) Interrupts are distributed across completion queues, allowing processor cores and/or threads to avoid interruption by notifications that aren't part of their tasks-at-hand.

Simple-yet-powerful Device Model

a) The command set is small, totaling only eight commands at this time. The command set has been optimized to enhance the performance and expose the enhanced functionality of devices based on persistent memory technology.

b) NVMe defines the concept of Namespace, a construct which enables the partitioning of the physical storage extent into multiple logical storage extents, each of which can be accessed independently of other logical extents. Each NVMe Namespace may have its own pathway, or IO channel, over which the host may access a Namespace. Namespaces allow the parallelism available in upper layers of today's platforms to be fully exploited in the storage system.

Overall, the NVMe interface requires only two (2) register accesses per command issue/completion cycle, avoiding locks on any queue or associated data structures. With this base of efficient IO, plus the powerful queuing model and features like Namespace, NVMe more fully exploits the hardware and software parallelism that has become an increasing part of today's platforms.

6 Applications

Although significant amounts of performance data are not yet available this early in the NVMe life cycle, it is expected that any NVMe-based system, as well as most existing storage applications, will see measurable performance boosts over that of AHCI-based systems. Furthermore, the NVMe specification includes a number of advanced features that, while unlikely to be utilized in early application implementations, will yield additional performance improvements, over AHCI, once they've become widely accepted throughout the industry.

7 Driver Availability

In today's client SATA platforms, the vast majority of which are Windows-based systems, the SATA software stack ships with Windows. Consequently, the SATA Express/AHCI solution, which utilizes legacy drivers and software, "just works" out of the box.

The NVMe driver ecosystem is rapidly maturing. An NVMe driver (StorNVMe.sys) became available from Microsoft in Windows 8.1 as of Aug-2013. The features are still evolving (for example, full boot support) and implementers should consult Microsoft for the latest status. An NVMe community Windows NVMe driver is also available (see https://www.openfabrics.org/resources/developer-tools/nvme-windows-development.html). Early NVMe Windows devices may still ship with proprietary mini-port drivers, but this will change quickly. Linux and VMware NVMe drivers are also available.

An NVMe compliance and interoperability program is being run by the University of New Hampshire Interoperability Laboratory to test NVMe devices and software for plug and play (see https://www.iol.unh.edu/services/testing/NVMe/). The first NVMe Compliance and Plug Fest event was held in spring 2013 and a 2nd is being planned for Q1 2014.

8 Summary

The SATA/AHCI interface evolved to address the needs of high performance HDD devices and has served this use case ably, enabling SATA devices to be used ubiquitously from laptops to servers. The parallelism built into AHCI, while not fully enabling the parallelism available in today's host platforms, is more than sufficient for the relatively slower SATA devices it was intended to serve. SATA/AHCI is even sufficient for 1st-generation SSD devices, especially in relatively lower-end mobile platforms, such as tablets and ultra-lite laptops.

However, in recognition of the evolution of storage requirements, and due to the technical challenges involved in taking the SATA PHY from its current maximum of 6Gb/s to 12Gb/s, SATA-IO chose to adopt PCIe, with its 8GT/s link and its multi-lane capability, as the physical interface for client storage. SATA Express hosts can interoperate electromechanically with both legacy SATA and SATA Express devices based on standard connectors now being defined, but system designers must also consider the choice of the NVMe or AHCI device interfaces.

Both AHCI and NVMe bring advantages and costs as summarized in the table below. At a high level, AHCI brings compatibility, with attendant performance inefficiencies due to legacy architecture, and NVMe brings higher performance in the near term, far better performance and scalability in the long term, but with a slight lag (vs. AHCI) in full function in-box driver Windows support (which is, however, rapidly maturing with the introduction of StorNVMe.sys by Microsoft).

	AHCI	NVMe
Maximum Queue Depth	1 command queue 32 commands per Q	64K queues 64K Commands per Q
Un-cacheable register accesses (2K cycles each)	6 per non-queued command 9 per queued command	2 per command
MXI-X and Interrupt Steering	Single interrupt; no steering	2K MSI-X interrupts
Parallelism & Multiple Threads	Requires synchronization lock to issue command	No locking
Efficiency for 4KB Commands	Command parameters require two serialized host DRAM	Command parameters in one

	fetches	64B fetch
Driver Support	Typically in-box	Installed with device

Together, the NVMe and AHCI device interfaces enable options for SATA Express devices that provide a path from the legacy SATA environment to a PCIe-based client SSD solution.

9 References

1) Why SATA Express (www.sata-io.org/sites/default/files/documents/Why_SATA_Express.pdf)
2) NVMe and AHCI – A Technical Overview (http://www.sata-io.org/sata-express)
3) Serial ATA Revision 3.2 Gold (www.sata-io.org)
4) Serial ATA Advanced Host Controller Interface (AHCI) 1.3 (www.intel.com/content/www/us/en/io/serial-ata/ahci.html)
5) PCI Express® Base Specification Revision 3.0 (www.pcisig.com)
6) NVM Express Revision 1.0c (www.nvmexpress.org)