SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2015

I/O Virtualization in Enterprise SSDs

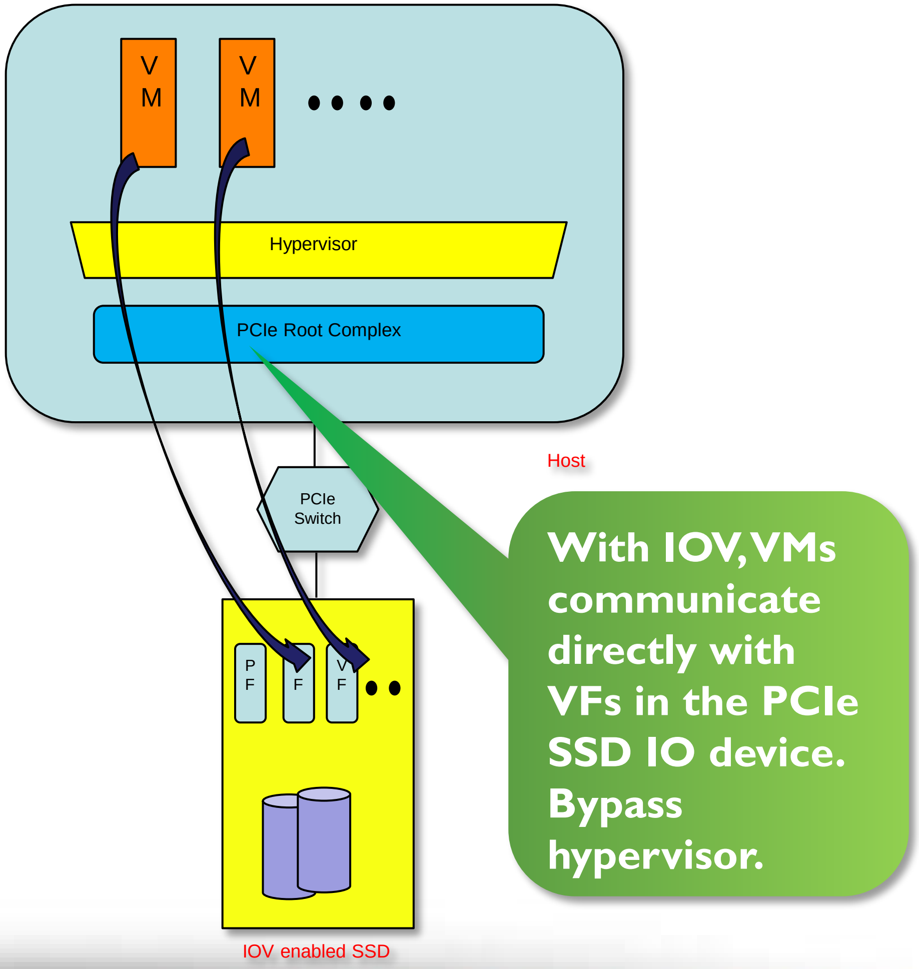
by Zhimin Ding
Toshiba America Electronic Components, Inc

Topics

- Why IO virtualization in eSSD
- Design objectives
- SR-IOV and NVMe
- Techniques and challenges
- Application benefits
- Summary and future work

Why I/O Virtualization in eSSD

- Server virtualization, especially growing number of virtual machines.
- High performance SSDs, e.g. PCIe-based SSD with >1M IOPS.
- Without IOV, the % overhead (especially latency) of Hypervisor to handle IOV becomes significant.



V M
V M
....
Hypervisor
PCIe Root Complex
Host
PCIe Switch
P F
F
V F
..
IOV enabled SSD
With IOV, VMs communicate directly with VFs in the PCIe SSD IO device. Bypass hypervisor.

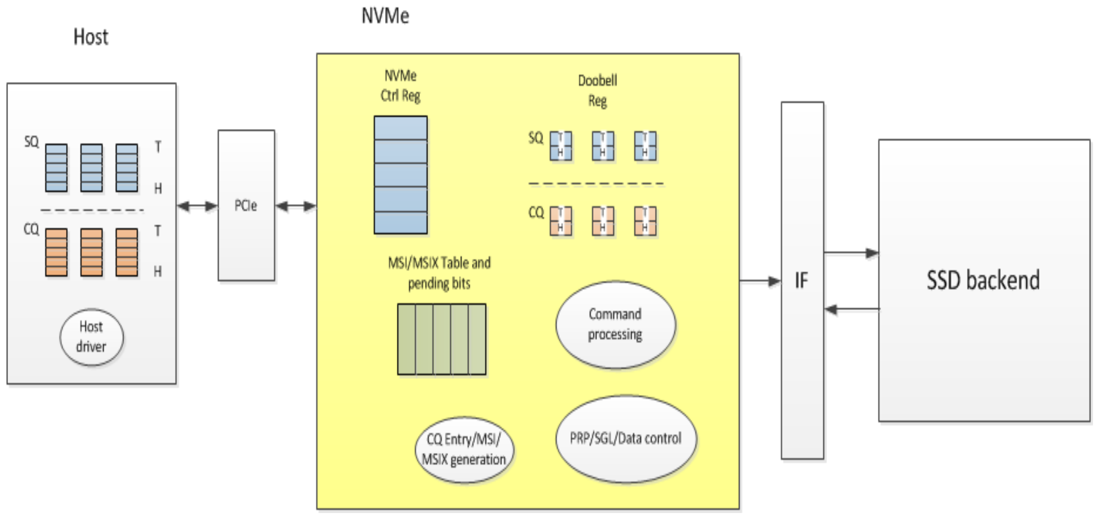
IOV eSSD Design Objective

- Efficient utilization of SSD resources to achieve maximum system performance for all VMs
- Matching (virtual) hosts' work load with SSD's virtual function capabilities
  - Each VM may have very different performance requirement
- Minimum HW/FW overhead in achieving the above.
- Flexible, can be programmed to adapt to very different user's VM profiles

One size fits all!

Implementing PCIe SR-IOV in NVMe

- NVMe: Scalable Command queueing interface
  - With support of Message Signaled Interrupts (MSI and MSIX)
- IOV is supported mostly in the NVMe, thus front end portion of the SSD system



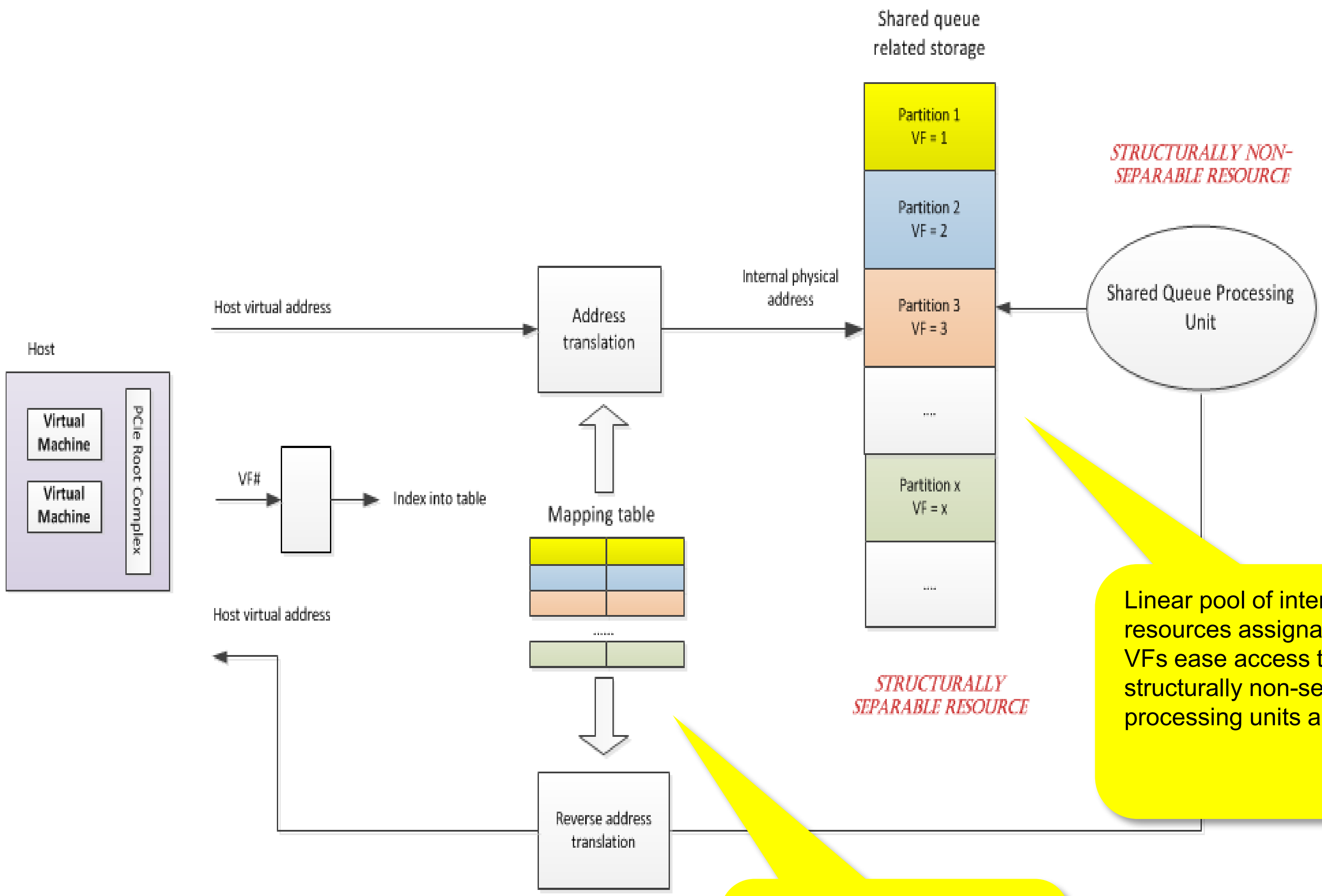
NVMe is very scalable, based on SQ/CQs and MSI vectors. These can be then partitioned for virtual functions.

Techniques and design challenges

- Techniques:
  - Queues and MSIs are partitioned among VFs
  - Processing capabilities and bandwidth of pipes are then provisioned with Queues and MSI vectors.
  - NVMe Ctrlr Regs and PCIe Config Regs partitioned for VFs as well.

- Challenges:
  - Structurally separable resources (e.g. memory) vs non-structurally separable resources (e.g. processing units, pipes)
  - What about Errors and exceptions? Especially (virtual) host errors.

Resources management in IOV-enabled SSD



Linear pool of internal resources assignable to VFs ease access to structurally non-separable processing units and pipes

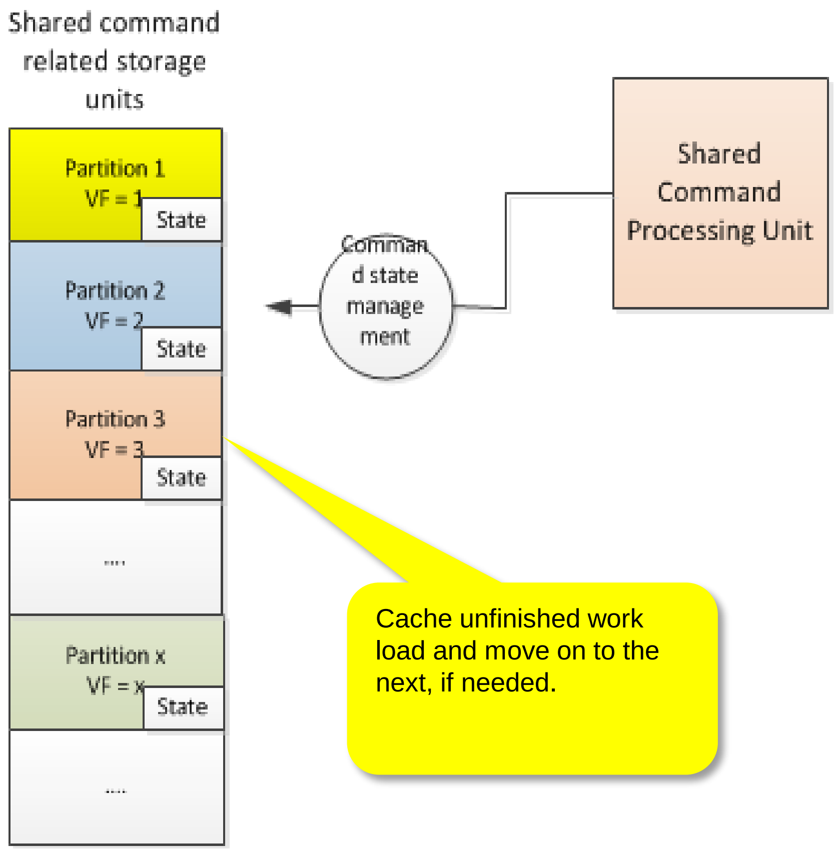
Mapping table defines translation between external and internal SQ/CQ/MSIX IDs

Resources management in IOV-enabled SSD (2)

- Linear pool of internal SQ/CQs/MSI(X) pointers/vectors provided.
  - For ease of access shared processing units and internal pipes
- Mapping table defines translation between internal and external queue IDs
  - Programmable
  - Translation is done at line speed to provide quick VF specific processing
- Translation logic
  - Forward translation: Host initiated transactions converted into internal transactions.
  - Reverse translation: Internal transactions converted back to Host communications

Non-blocking processing of VF traffic

- Real world SSD performance affected by host behavior.
  - In a non-IOV system, system performance suffer due to host error and that is considered fair game.
    - Example, host may not post CQ head update promptly, causing CQ full and slow down traffic.
- However, it is unreasonable for a mis-behaving virtual host(s) to bring down the performance of an IOV enabled system.


Shared command related storage units
Partition 1 VF = 1 State
Partition 2 VF = 2 State
Partition 3 VF = 3 State
....
Partition x VF = x State
....
Command state management
Shared Command Processing Unit
Cache unfinished work load and move on to the next, if needed.

STRUCTURALLY SEPARABLE RESOURCE

PCIe interrupts are Virtualized

- Provide a linear pool of vector storage, PBA bits and mask bits.
- Dynamically mapped to VF spaces according to a user-programmable mapping table.
- Processing unit to generate MSI or MSIX interrupt is time-shared.

Same concept as how queues are partitioned, mapped and processed for virtual functions.

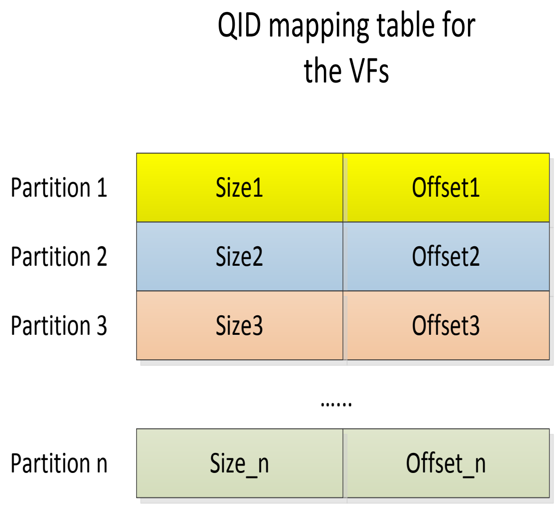
Error reporting and handling

- Errors and exceptions are detected on per VF bases.
- Error related interrupts generated on per VF basis for VF specific handling.
- Allow selective resets of states for specific VFs when needed. (e.g. function level reset or FLR)

Even though error handling is not as performance critical as normal IO flow, we have to make sure errors belongs to one VF do not disturbed IO flow of other VF(s).

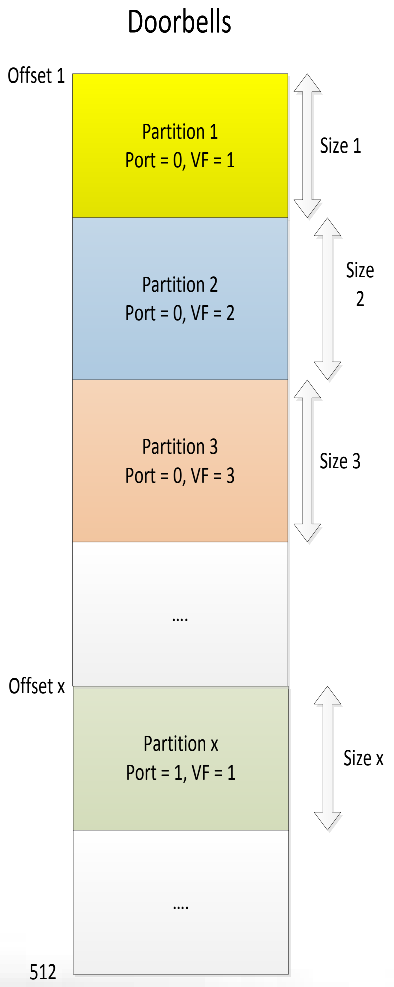
How to configure resource partition?

- Divide key resources, i.e. SQ/CQs into partitions.
- Each specified by its size and offset from the base.
- Build a partition mapping table.



QID mapping table for the VFs

Partition 1	Size1	Offset1
Partition 2	Size2	Offset2
Partition 3	Size3	Offset3
	……	
Partition n	Size_n	Offset_n



Application benefits

- Best utilization of SSD resource to achieve maximum server system performance.
- Tolerant to glitches and misbehaviors of some virtual hosts, while maintain maximum system level performance.
- Flexibility for matching virtual hosts' work load to VF capabilities
- Adaptive to a wide variety of use cases and application scenario

Summary and future work

- PCIe and NVMe being highly scalable, lend itself well to IO Virtualization Implementation.
- Many key considerations to make a good performance IOV system.
  - Needs to be performed at architectural stage. Cannot be after-thoughts.
- <u>Future work: </u>From dynamic to intelligent allocation of SSD performance potential to ever-changing VM performance requirement.

Thank you

Contact:
Zhimin Ding, PhD
Principle Engineer, Design
Toshiba America Electronic Components, Inc
2825 North 1st Street, CA 95134
zhimin.ding@taec.toshiba.com