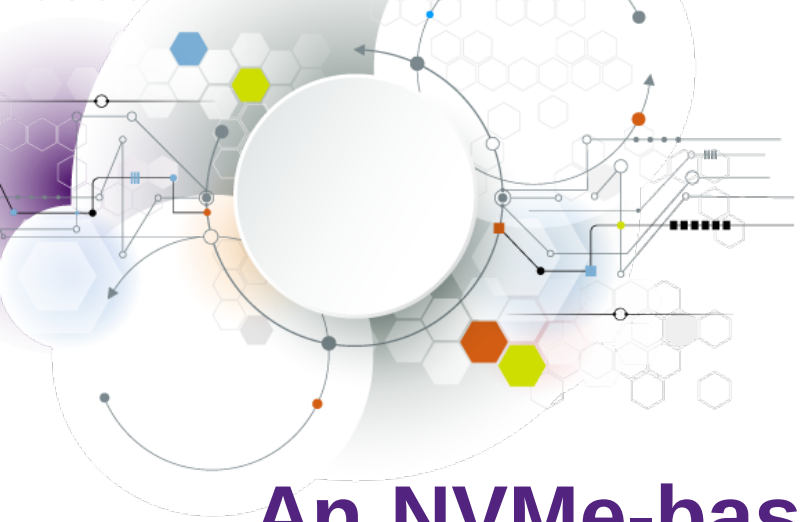


SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

An NVMe-based Offload Engine for Storage Acceleration

Sean Gibb, Eideticom

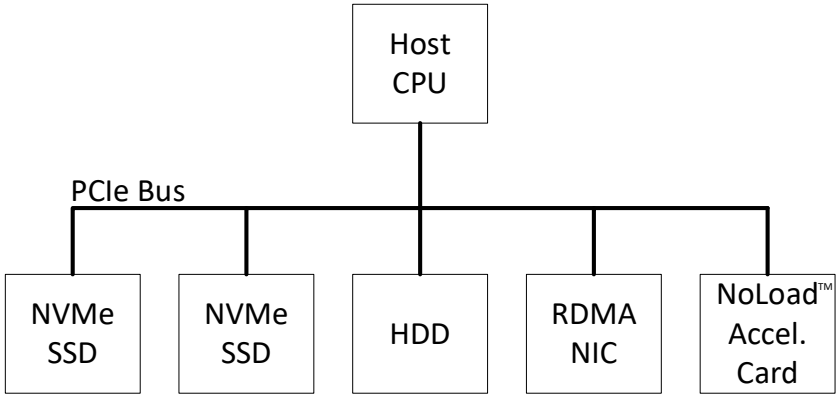
Stephen Bates, Raithlin

EIDETICOM

Overview

- Acceleration for Storage
- NVMe for Acceleration
  - How are we using (abusing ;-)) NVMe to support acceleration?
- Embedded NVMe Controller
  - RISC-V on FPGA
- Performance
- Fabrics, Peer-to-Peer, and CMB

Acceleration

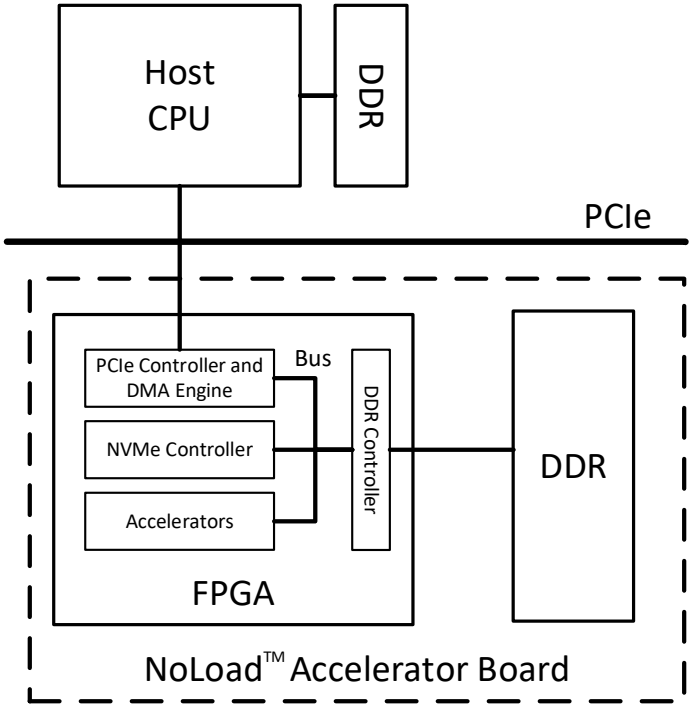


- Storage I/O bandwidth increasing rapidly
- Storage workloads can be taxing on host CPU
  - Hyperconverged storage exacerbates the problem
- Reconfigurable logic can provide compelling solution for storage workloads

Acceleration Over NVMe

- Using NVMe host controller interface to provide data and control to accelerator functions
  - No need for proprietary drivers
    - Avoid driver development and take advantage of improvements in NVMe standard, drivers and tools
  - Leverage industry-standard NVMe test tools
    - Assist with deployment and benchmarking
  - Test tools, software and ecosystem
    - Can tie into NVMe over Fabrics
    - Can leverage inbox drivers in all modern OS
    - Can leverage servers and storage systems developed for NVMe

Basic Architecture



- Presents as an NVMe 1.2 device with multiple namespaces
- Host CPU communicates with accelerators via NVMe commands to NVMe controller
- NVMe controller pulls commands and data via DMA engine
- Accelerators easily integrated on an internal bus
- Accelerators are mapped to NVMe namespaces to enable discovery and command and data routing

Acceleration Over NVMe Commands

- Identify Namespace is used for accelerator discovery
  - Vendor specific field used to provide accelerator specific information
- Write is used to provide data to accelerators
- Read is used to retrieve results from accelerators
- Writing and reading from specific namespace to communicate with specific accelerator
- Vendor specific commands available for accelerator specific control

Embedded NVMe Controller

- For flexibility developed an embedded controller
  - Faster turnaround on compliance debugging
  - Quickly implement new features
- Downside is that getting performance from an embedded controller on FPGA is more difficult
  - Requires coprocessors and offload

Processor Selection

- Which processor to use for controller?
- Requirements
  - Platform agnostic
  - Broad software ecosystem
- Soft Requirements
  - Extensible instruction set
  - 32-bit and 64-bit addressing available

RISC-V

- RISC-V is an instruction set architecture
  - Gaining momentum in academia and industry
- Originally developed in 2010 at UC Berkeley
- Several commercial and open-source processor implementations available
  - Can be autogenerated using open-source toolchain
- Meets our hard and soft requirements with a few caveats

RISC-V Software Ecosystem

- RISC-V includes software support for:
  - GCC toolchain with GDB support
  - LLVM toolchain
  - Spike simulator
  - QEMU model
- Includes OS support for GNU/Linux, FreeBSD, and NetBSD

RISC-V Core

- Original plan was to use Rocket core generator
  - Rocket designs are best suited to ASIC
  - Only achieved 50MHz on FPGA
- Alternative was to start from ORCA
  - BSD license FPGA-optimized RISC-V CPU
  - Using 32-bit instruction set to reduce size
  - Original design achieved 125MHz

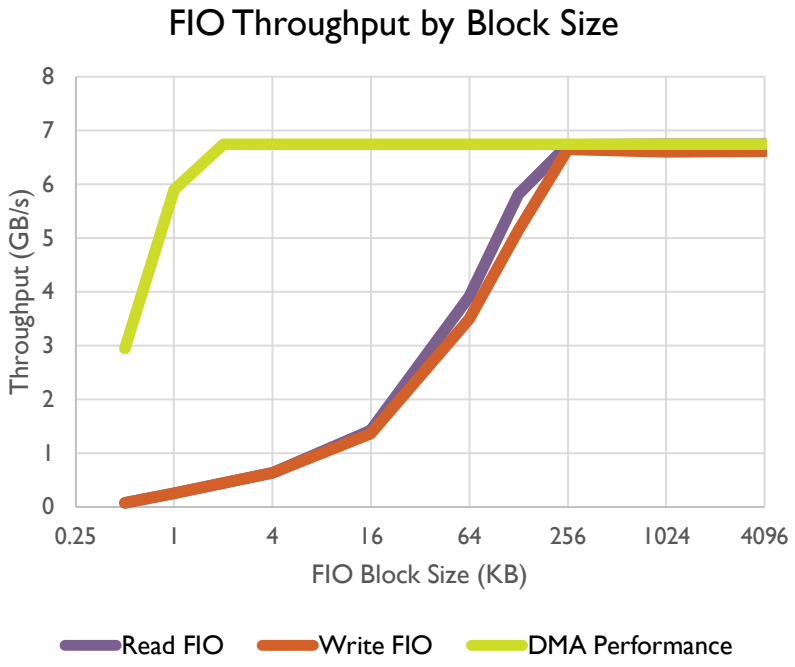
Development Process

- Wrote software for our controller and replaced NVMe QEMU model to verify functionality
- Ported to RISC-V
  - Porting DMA accounted for 90% of effort
- Verified controller against Linux and Windows drivers with a backing RAM drive
- Performance testing for the RAM drive

Test Setup

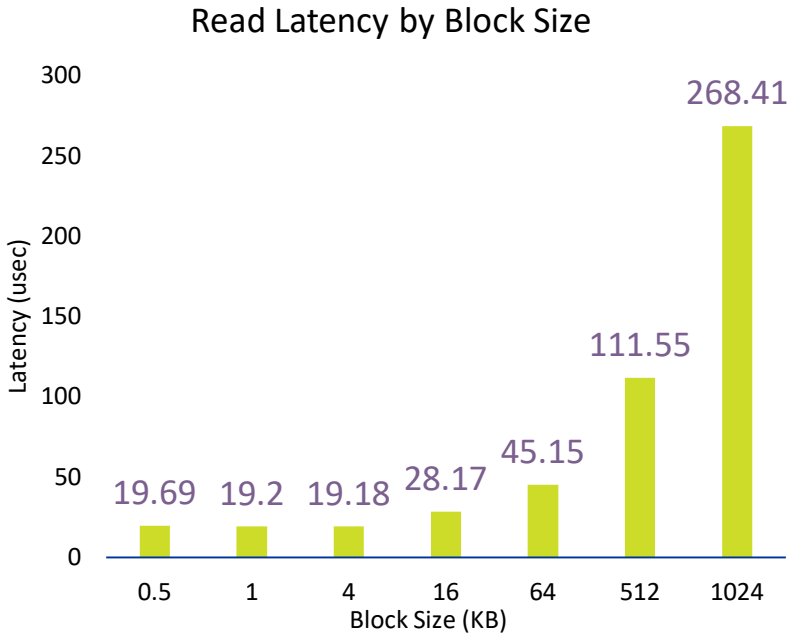
- Intel i5 6500
- PLX9797 PCIe switch
- Eideticom NoLoad Accelerator targeted to Xilinx XCVU095 Ultrascale
  - PCIe Gen3x8
  - 2 x 2.5GB DDR4
- Samsung 960 EVO 250GB M.2 SSD
- Intel SSDPEKKW256G7 256GB M.2 SSD
- Viavi PCIe Capture Card

FIO Performance



- Big block transfers saturate PCIe Gen3x8 throughput
- Small block transfers require further work by adding more RISC-V cores and command processing offload engines
- Capable of saturating PCIe Gen3x8 with current DMA engine for most block sizes
- Verified with PCIe capture card that we are saturating PCIe bandwidth

Latency



- NVMe latency is largely due to software path
- Accelerator use model will tend to focus more on throughput than latency
- Future improvements in NVMe command processing will improve latency

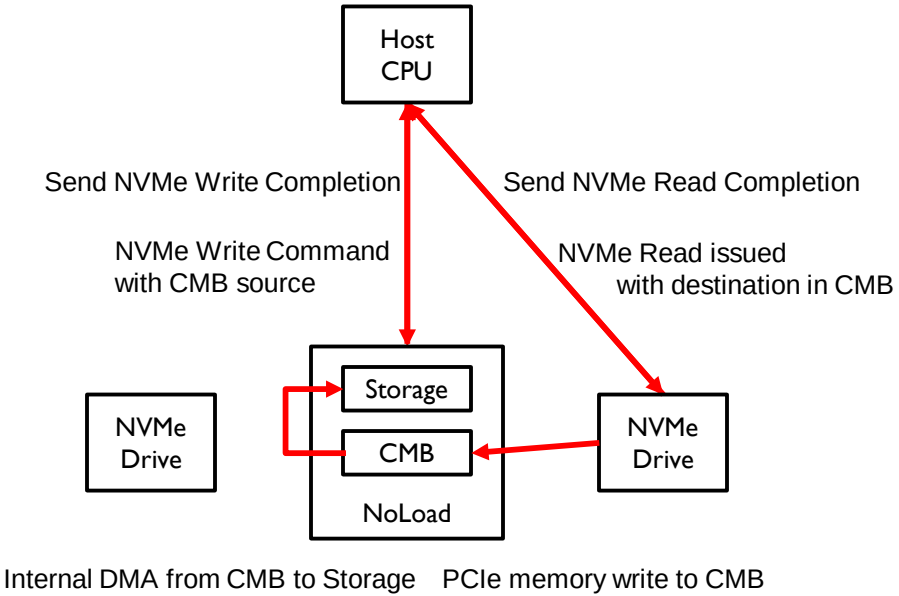
RISC-V Complications

- No external debugger for RISC-V yet
  - Difficult to track down bugs in embedded system without external debugger
  - Built our own internal primitive debugger
- Trade-offs between code size and clock rate in FPGA design are persistent
- Instruction bubbles in the processor were slowing us down
  - Fixing ORCA implementation improved performance
  - Turned up and fixed several ORCA bugs during this process
  - Managed to get ORCA to 190MHz on FPGA
- Built DMA offload to handle data transfers and Completion commands
- Underbaked or missing features in ORCA

Error Correction Accelerator

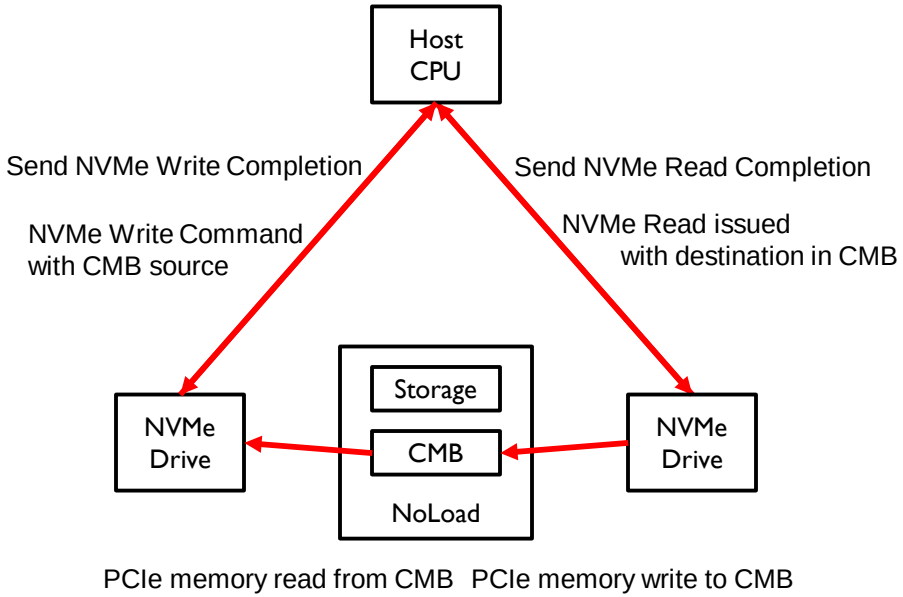
- Built an RS(32, 4) EC accelerator
  - ISA-L compatible
  - Utilizes 16KB block sizes
- Saturates PCIe Gen3x8 throughput (i.e. 8GB/s)
- Modified ISA-L perf test in less than an hour to use NoLoad NVMe Accelerator
- Using our host side API software integrates into host software with 10 lines of code
- Roadmap includes PCIe Gen3x16 (PCIe Gen4x8) dual namespace version capable of 16GB/s

Peer-to-Peer CMB



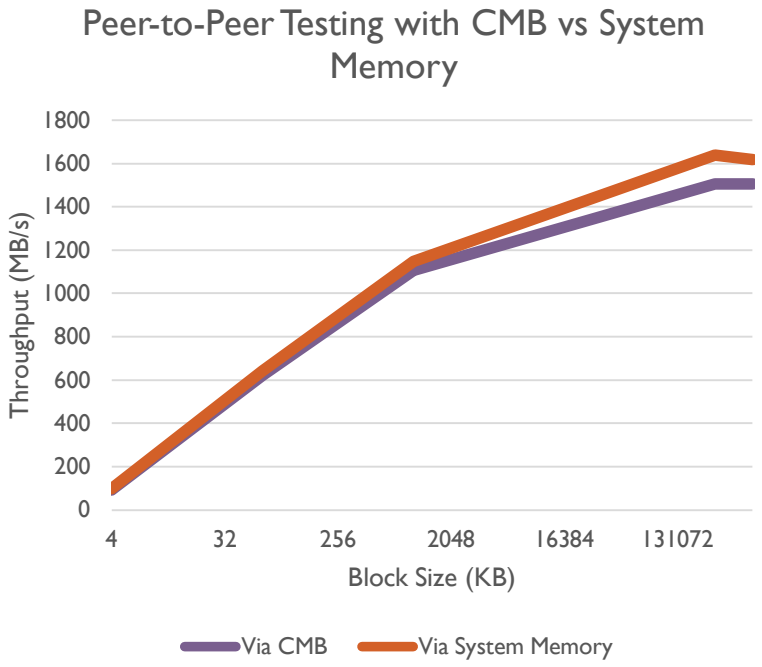
- Added full CMB support to accelerator
  - Took 1 day thanks to software controller
- With data CMB only one external DMA is required
  - Removes load on host CPU for memcpy

Peer-to-Peer CMB (Staging Buffer)



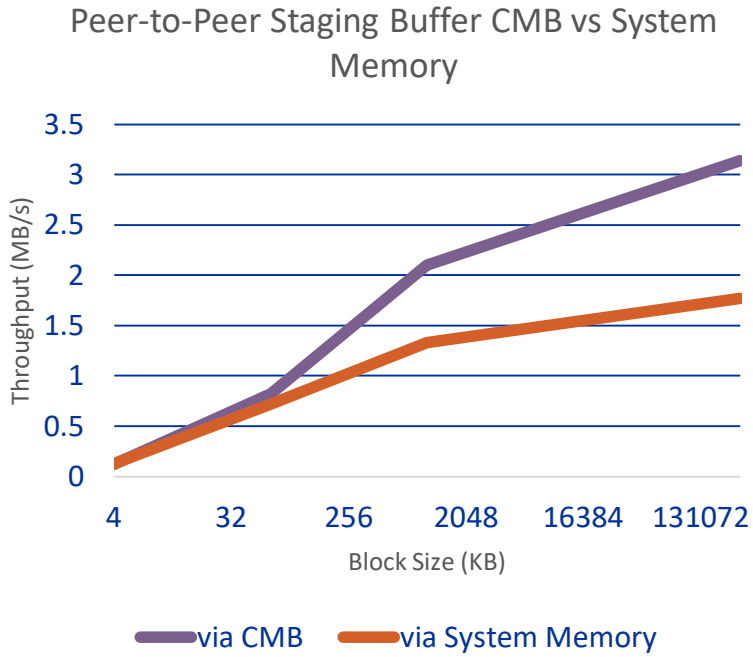
- Added full CMB support to accelerator
  - Took 1 day thanks to software controller
- With data CMB only one external DMA is required
  - Removes load on host CPU for memcpy
- CMB can be used as a staging buffer between two devices that do not support CMB

Peer-to-Peer CMB Results



- Current setup saturates due to insufficient sources in the test environment

Peer-to-Peer with CMB as Staging Buffer



- Our test setup has insufficient sources to demonstrate expected maximum performance