2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

NVMe Development and Debug for a 16 x Multicore System

Soummya Mallick, Microchip Inc, Sacramento, CA
Raj Mathur, Arindam Guha, Frank Schirrmeister, Cadence Design Systems, San Jose, CA

***Abstract* - Non-Volatile Memory Express (NVMe) is an open logical device interface specification for accessing non-volatile storage media attached via a PCI Express (PCIe) bus. As a logical device interface, NVMe has been designed from the ground up to capitalize on the low latency and internal parallelism of solid-state storage devices. This paper will introduce the verification challenges for NVMe and discuss innovative verification approaches, combining simulation and simulation acceleration to achieve verification completeness using Accelerated Verification IP for PCIe. This approach achieves 300x-700x acceleration over simulation for 50-100MGate designs transacting 10 IOPS.**

I. Introduction

Non-Volatile Memory Express (NVMe) is an open logical device interface specification for accessing non-volatile storage media attached via a PCI Express (PCIe) bus (see [1]). As a logical device interface, NVMe has been designed from the ground up to capitalize on the low latency and internal parallelism of solid-state storage devices. Verification of NVMe and its hardware and software integration is facing numerous challenges, including concurrency when it comes to communication between firmware and hardware as illustrated in Figure 2. Hundreds of concurrent engines saturate the PCIe link, and interaction between firmware is controlled by semaphores. For a typical data-path managed by firmware, simulation of all cores in a multi-core system is not feasible as simulation would take too long to execute the required firmware. The stimulus for NVMe commands and the data-path is slow to simulate each Transaction Layer Packet (TLP). Due to state explosion, System-on-Chip (SoC) test benches would require days or even weeks to run, with the PCIe simulation itself taking several days.

II. Design under Verification and Objective

The design under verification is a NVMe endpoint with 16 PCI Gen 4 lanes that terminates the flash drive and provides a flash controller for users. Key verification challenges are driven by 16 parallel lanes creating many channels talking to the host, and 16 processor cores running at 1.2 to 1.6 GHz. As a result, the software workload has very high impact. The verification of a software-controlled data path poses unique challenges, verifying a tremendous amount of software interacting on 16 cores via a cache coherent interconnect network. Software based message traffic from the CPU controls all of the hardware functional units. Due to the simulation speeds, block-level test-bench only allows a small portion of software execution and is focused on randomization, with fixed software. In addition, verification needs to include the environment with PCI cores as peripheral, custom accelerators, DMA engines that are moving traffic, for which ultimately software is in control. The verification complexity is illustrated in Figure 1.

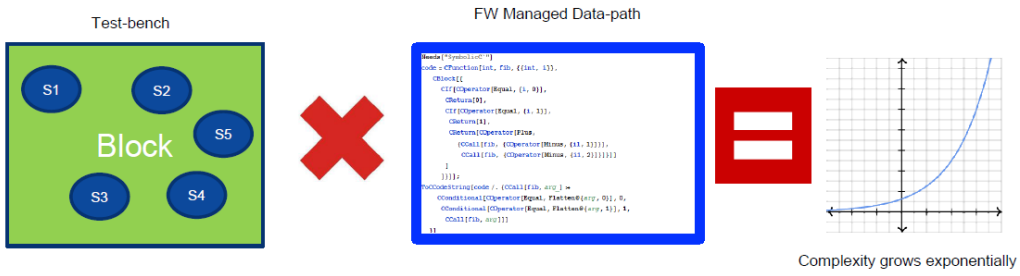

Figure 1: NVMe Verification Challenge

For verification, a NVMe bring-up platform will be required that allows stimulation of a design with randomized NVMe traffic with an UVM-SV test bench to increase coverage. It should be able to leverage UVM logging to extract NVMe grammar contained in Transaction Layer Packet (TLP) traffic and allow the debug of C code during run-time via virtual UARTs or virtual JTAGs in an Integrated Development Environments (IDEs). A productive NVMe bring-up platform will require the hooks necessary to implement protocol analyzers and needs to offer the flexibility to debug C code at run-time. It also should contain the necessary hooks to ease randomization, creating UVM sequences to stress the DUT and to leverage sequences from simulation.

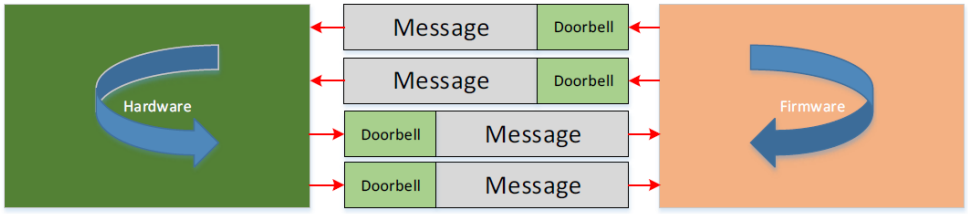

Figure 2: Verification challenges introduced by concurrency

III. Verification Approach

Microchip (MicroSemi) has traditionally been using classic RTL simulation and In-Circuit Emulation (ICE) for verification. The combination has been very successful in finding bugs. RTL simulation for block level verification using randomization, and ICE for firmware validation and bring up of flash memory, can be combined more efficiently, specifically when it comes to randomization. While the message exchange between hardware and software is fully synchronized and cache coherent, the data path is under software control. Traditional techniques cannot randomize more than a module, and due to simulation speed only a few TLPs can be simulated per week, rendering UVM simulation as in-sufficient.

Simulation acceleration was introduced as a solution, keeping the testbench on the host while compiling the DUT into an emulator with a fast inter-connect between them as shown in Figure 3. Accelerated Verification IP (AVIP) for PCIe was used to generate transactions. Layered on top of the PCIe connection is the NVMe protocol layer to provide NVMe randomized traffic. Starting from a pure ICE environment, a simulation acceleration environment allows the use of randomized tests that can be re-used from simulation and can run on various engines.

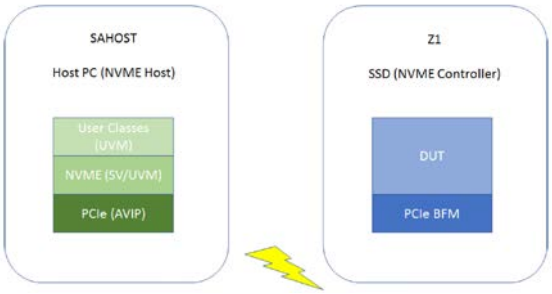

Figure 3: Connecting Simulation and Emulation using AVIP

While In-Circuit-Emulation (ICE) is very efficient and necessary to apply real network traffic to the design under test, for example with physical testers, and RTL simulation is very efficient with randomization for block-level test, augmentation of ICE with virtual environment allows capabilities that are more difficult to achieve in pure ICE or RTL simulation mode:

- As an interoperability platform, simulation acceleration offers more control and specific capabilities of injection of data into the tests. For instance, specific reads and writes can be performed, independent from real world traffic. This fact lends itself to a great bring-up platform for ICE.

- Simulation acceleration allows the assessment of race conditions that are hard to assess in ICE as the clock is always present and cannot be stopped or varied in timing when a TLP arrives. As a result, race conditions can be missed in pure ICE. As RTL simulation is often too slow to fully express race conditions, simulation acceleration offers an elegant alternative.
- With its execution speed in the sub Hz range, RTL simulation does not allow one to efficiently execute credit checks and link events. For instance, it may take a week or more to get to the PCIe link up in simulation, and it doesn't even account for any real traffic being transmitted.
- Some bugs simply require lots of traffic and happen later, when link up has happened and many TLPs are executed. Examples like tests that measure queue fairness are obvious examples.
- NVMe traffic requires fixed enumeration on an OS in ICE execution. To modify enumeration, users would have to modify the drivers which is inherently hard to do. Intent is to do less driver development, but instead to leverage NVMe sequences to test firmware.
- In ICE environment, an actual OS with stock drivers running on a target server, can only have a single queue per core. Virtualization allows more flexibility, for instance with 8 submission queues and one completion queue.
- Users would like to utilize Single Root I/O Virtualization (SR-IOV), allowing a single peripheral component to appear as multiple, separate physical devices to a hypervisor that runs the OS. Virtualization provides easier control than individual installation on SpeedBridges in ICE configurations.

There are several additional capabilities that users are looking for in a NVMe bring-up platform:

- Users require high-level debug including NVMe and PCIe traffic. Typical questions include "What is happening on the PCI bus as I am receiving this traffic?". While analysis in ICE can be done by attaching a traffic analyzer, high level TLP logging are inherently easier to understand and help in faster root-causing of issues.
- ICE setups allow efficient C debug by attaching classic embedded software debuggers. However, more flexibility to debug C code is desirable, using for instance virtual UART and virtual JTAG connections into the debug IDEs. Sometimes compromises compared to ICE speed may be needed, but are left to the user offering both ICE and virtual setups.
- Reuse of randomization from simulation environments allows users to simply stay with UVM sequences that already (need to) work in simulation. They can be brought into a simulation acceleration platform in order to execute faster. In addition, UVM logging that is used to look at TLP traffic can be done using post-processing without having to develop additional monitors. Protocol analyzer, can be focused on NVMe traffic only as indicated in Figure 4.

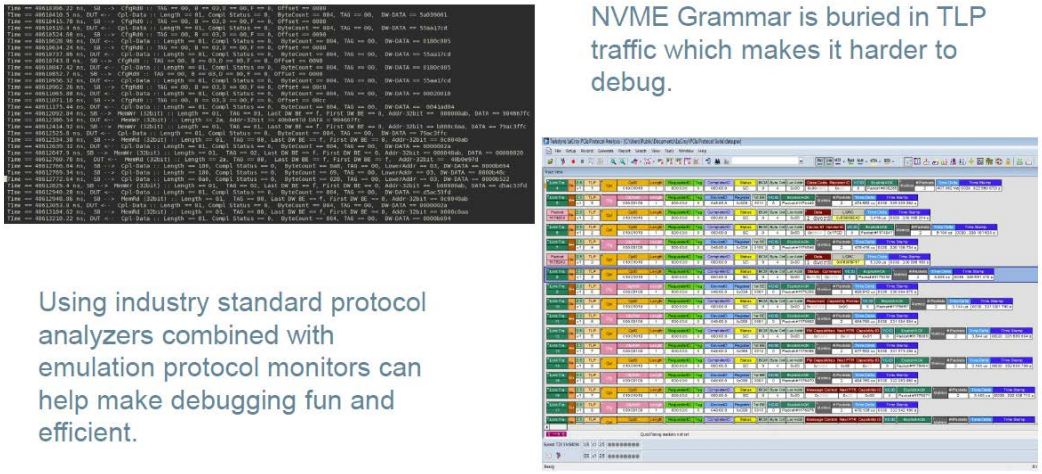

Figure 4: NVMe Traffic Debug

IV. Implementation of Verification Approach

As a first step, the team modified the design that was talking to a physical Speedbridge using a wrapper via pipe connect, and replaced it with an AVIP wrapper, pipe to pipe. The PCIe UVM testbench that was writing low level PCIe commands, was extended with NVMe that is using PCIe as a transport layer. As a result, the team was able to do randomization and sequences from PCIe and NVMe world. Figure 4 illustrates the resulting setup.

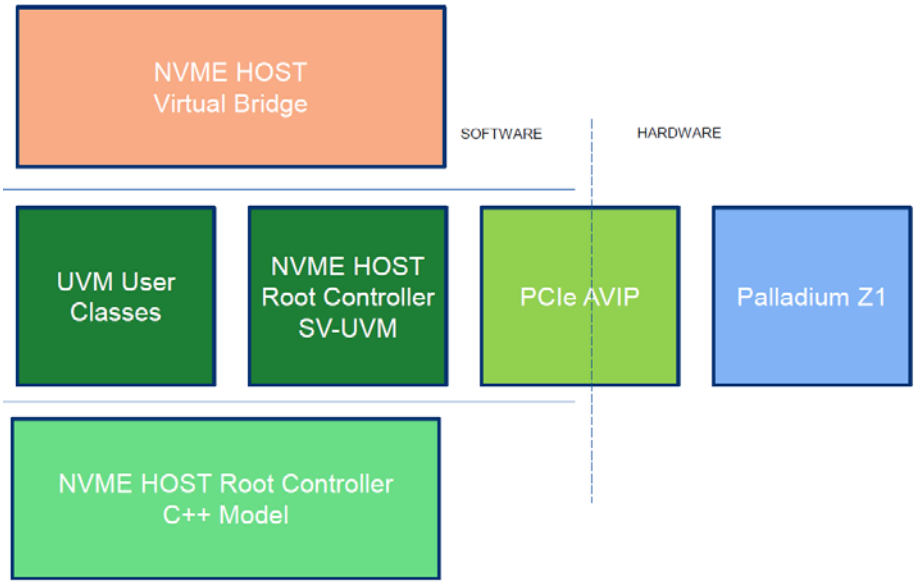

Figure 5: Verification Setup with PCIe and NVMe Debug

As a second step, the physical connection between emulator and host will be replaced with VirtualBridge. Using QEMU to abstract the actual workstation, teams do not have to worry about the impact of physical delays and provides line of sight to debug drivers on multiple operating systems at the same time, for instance allowing Ubuntu, CentOS and others talking to the same design, as a benefit of VirtualBridge and SR-IOV (from more on VirtualBridge see also [2] and [3])

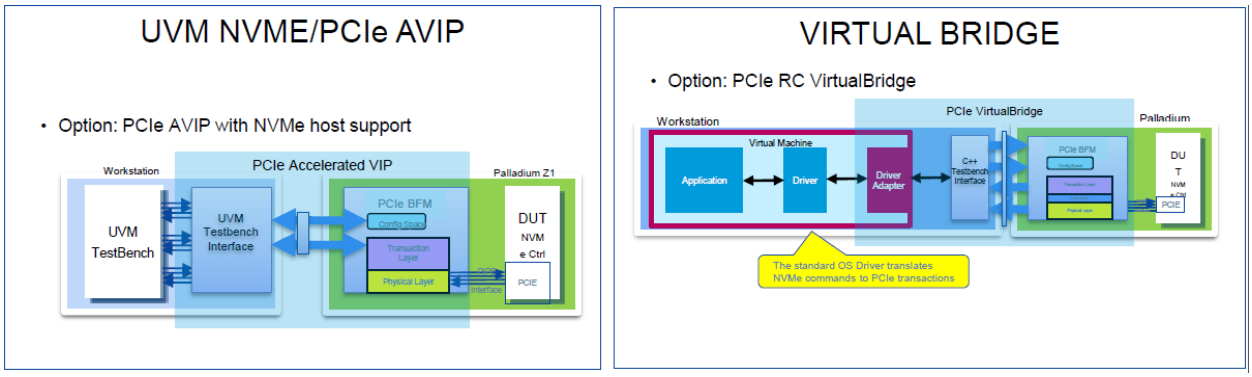

Figure 6: UVM PCIe AVIP and VirtualBridge Setup

V. Results

Modification of the design turned out to be relatively straight forward. The team was able to get the first sequences of TLP traffic within a couple of weeks. By including AVIP into the flow, they can now generate traffic

at performance in the MHz range. With RTL simulation taking a week for the link up and a couple of TLPs, it would have taken the team months per issue. The.se timeframes render the full chip simulation efforts infeasible.
Example bugs discovered in minutes given the speed of the setup included:

- BOOT-ROM bug - In an ICE environment timing of incoming TLP is not easy changed, in pure simulation, reading a different timing takes a week or more to reproduce. In the acceleration setup the team was able to change the timing of the TLP, and immediately found a bug in the firmware that had assumed a specific TLP delay when setting a CRS disable flag, leading to a race condition.
- Link Events – the NVMe stack up was full, during link events, after NVMe enumeration happens, the hardware did not reset the thread count for the 2nd or 3rd linkup. As a result, the software expected threads to happen, registers were updated, but did not see any threads. This could only happen under lots of traffic.
- Engine Initialization Sequencing – commands were swapped during initialization and it would have taken too long in simulation to identify this issue.

Using the NVMe bring-up platform using simulation acceleration we achieved 300x-700x acceleration compared to pure simulation for 50-100MGate design transacting 10 IOPS.60 hour simulation runs were reduced to minutes and multiple bugs discovered using simulation acceleration in the ROM, link event and NVMe implementation. The best of both worlds – UVM sequences and firmware execution at high speed – was combined.

VI. Conclusion

In conclusion the authors would like to show the readers that when the simulation space gets too large and time consuming to be handled in pure software, especially in a case of heavy firmware workload, and true randomization is desired, Simulation Acceleration, when deployed strategically, lends itself as a very powerful tool. True best of both world advantages can be harvested by getting almost 25-50% of ICE like emulation speeds and full UVM testcase randomness. The cost of going from uncontrolled to controlled clocks is not substantial when the randomized stimulus is state-full. In other words, simulation acceleration benefits for a NVMe Drive controller are multiple orders of magnitude higher than state-less designs like PCIe Bridge SOC for example. The amount of TLP processing that happens in a PCIe Based drive controller are tremendous. Heavy duty firmware is deployed to track the PCIe Transaction and as the reader can imagine, is highly state-full.

When the NVMe Messages are randomized in UVM, they generate hundreds of TLP transactions that are correct and need a lot of processing by the device under verification and hence opportunity to find more and menaingfull bugs.

VII. References

[1] NVMe Specification, https://nvmexpress.org/, https://nvmexpress.org/wp-content/uploads/NVM-Express-1_3c-2018.05.24-Ratified.pdf

[2] “Cadence Announces VirtualBridge Adapter for Palladium Z1 Emulator to Accelerate Software Bring-Up Time by Up to Three Months”, http://bit.ly/2GsWfBL, downloaded 12/10/2018

[3] VirtualBridge Adapters, http://bit.ly/2T4HoPx, downloaded 12/10/2018