# Bridging the gap between system-level and chip-level performance optimization

Soniya Gupta, Synopsys Inc., Noida, India (*soniya.gupta@synopsys.com*)

Vikrant Kapila, Intel Inc., Singapore (*vikrant.kapila@intel.com*)

Holger Keding, Tim Kogel, Synopsys Inc., Aachen, Germany

***Abstract*** **The growing demand for AI and cloud enabled applications fuels an insatiable demand for data center resources. To minimize the energy consumption of data center processing and communication equipment, the semiconductor devices must be thoroughly optimized for the target application requirements and characteristics. We present a unique modeling methodology which enables the joint analysis of system-level applications and semiconductor architectures to optimize the overall computational efficiency. The key idea is to combine traditional system-level analysis based on queuing networks with state-of-the-art chip-level architecture design based on Transaction-Level Modeling (TLM). This new approach has been applied to the real-world design of a next generation SmartNIC System-on-Chip (SoC). Compared to RTL based methods, the Data Plane Development Kit (DPDK) [1] application and underlying VirtIO protocol have been quantitatively analyzed 6+ months earlier and the processing efficiency has been improved by up to 30 % using the given methodology.**

***Keywords: Level Modeling, Chip-to-Chip communication, Architecture Analysis, Performance Exploration***

## I. INTRODUCTION

Traditionally, architecture specification and exploration are done using spread-sheets analysis, using static formulas to calculate the Key Performance Indicators (KPIs) like packet throughput, end-to-end latency, energy efficiency, etc. Today, static spreadsheets are still helpful for analysis of theoretical boundary conditions. However, this approach does not allow the modeling of representative usage scenarios and the prediction of realistic KPIs for heterogeneous distributed multi-processor system. As a result, specifications based only on static analysis tend to be either overly pessimistic, leading to over-design and excessive cost, or overly optimistic, leading to under-design and missed performance requirements.

Early performance analysis and architecture exploration of System on Chip (SoC) architectures using TLM based SystemC Modelling has been an established methodology [2],[3] to mitigate the limitations of static spreadsheet analysis. Enabling early detection of performance issues and optimization of shared interconnect and memory subsystem configuration helps semiconductor companies to save cost and shorten time to market for complex SoC designs. The growth of distributed computing and cloud enabled devices are driving the need for performance analysis of multi-chip "Systems of Systems" at a very early stage. The overall efficiency of such distributed systems very much depends on matching end-to-end application data flows to the underlying SoC architecture and chip-to-chip communication resources. The methodology presented here combines the benefits of accurate SoC performance analysis using TLM based modelling with packet level modeling of chip-to-chip protocols and system-level application data flows. To enable this, we propose the Queuing Task Library (QTL) that provides a flexible, fast, and sufficiently accurate packet-based modelling technique to take design decisions during the early architecture specification phase. QTL allows the efficient modelling of system-level data flows and chip-to-chip communication protocols independent of the hardware platforms. The flexible and highly configurable mapping of the QTL application dataflows to TLM-based architecture models enables fast turnaround for early design space exploration and optimization. Compared to traditional RTL based profiling methods, the architecture can be optimized for the target application 4-6 months earlier, enabling much more impactful design decisions.

In this paper, we will first review related work in system level and chip level performance analysis, followed by an introduction of our methodology for unified architecture modeling. The feasibility of this approach is demonstrated by the results of a SmartNIC design project, which has been optimized for processing of DPDK [1].

## II. Related Work

Over the last 15 years, the semiconductor industry has adopted SystemC based Approximately Timed Transaction Level Modelling (AT-TLM) for pre-RTL performance analysis of complex SoC architectures [2],[3]. The established use-cases are hardware/software partitioning [4],[5] as well as the dimensioning and optimization of the shared interconnect and memory architecture [6]. On the other hand, discrete event network simulators are widely used for the performance analysis of distributed communication networks, like Ethernet, TCP/IP, LTE, etc. [7],[8].

There is a gap between these two domains, where Approximately Timed TLM is too detailed for modeling multi-chip systems, while network simulators are too abstract for modeling SoC-level performance effects like pipelined bus transactions and DDR memory timings. Hence, there is currently no solution for SoC-level performance analysis of chip-to-chip protocols, like e.g. PCIe connectivity of Smart-NIC, Ethernet routers of automotive domain controllers, or high-speed SerDes for data-center grade AI accelerators.

We propose to combine packet-level modeling of chip-to-chip protocols with the benefits of accurate SoC performance analysis using SystemC based AT-TLM. We leverage the concepts of queuing theory [9], which is a well-established analytical framework for the analysis and optimization of architectures [10]. In our case, we provide a library of components for modeling traffic sources, protocol layers, servers, and physical communication links that can be used to build and simulate queueing networks as well as interface with SystemC AT-TLM based architecture models. Our approach is similar to SystemQ [12], which also combines queuing network concepts with SystemC for SoC performance analysis based on stepwise refinement. As opposed to SystemQ, we follow the y-chart approach [13] and separate application dataflows from hardware resource models. This orthogonalization of concerns enables a more flexible exploration of architecture alternatives [14].

## III. METHODOLOGY

We start by describing the baseline methodology for early architecture analysis and exploration of SoC architectures, which is based on application workload models in combination with TLM Virtual Performance models [4], [5]. As illustrated in Figure 1, the idea of the y-chart approach [13] is to capture the processing and bandwidth requirements of the application workload separately from the underlying resources in the hardware platform. By mapping the application workload onto the hardware resource model, we obtain a virtual prototype that allows the quantitative analysis of KPIs like throughput, latency, utilization, efficiency, etc.

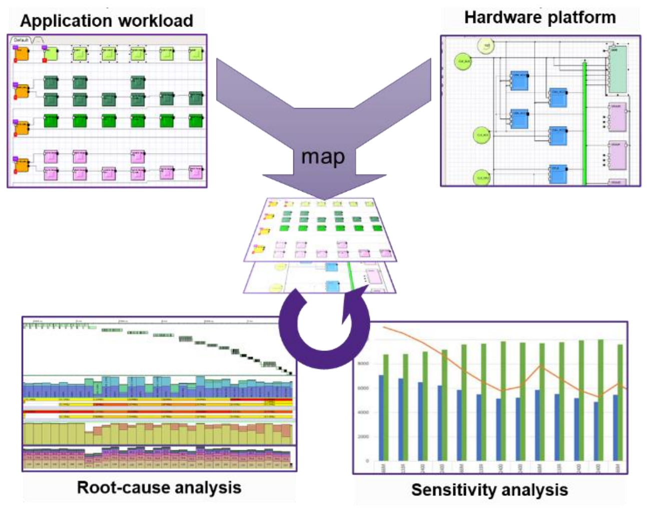

Figure 1: Y-chart approach for early architecture analysis

The workload model captures task-level parallelism and dependencies as well as processing and communication requirements per task in an architecture independent way. The HW platform models the available processing, interconnect, and memory resources of the envisioned SoC. The task-based application workload model can be mapped to Virtual Processing Units (VPU), which model the execution resources, e.g. a CPU, GPU, DSP, HW accelerator, or DMA. The VPU translates the communication requirements from the task graph into transactions, which stimulate the transaction-level models (TLM) of the shared interconnect and memory subsystem. At this level of abstraction, the detailed timing of on-chip bus protocols like AMBA [15] are modeled to accurately simulate the dynamic effects of bus arbitration, DDR latencies, Quality-of-Services (QoS) regulators, etc.

The overall time is modelled by combining stochastic characterization of processing time per individual tasks and the simulation of dynamic effects like e.g. task scheduling, interconnect arbitration, and memory latencies. This type of workload-based virtual prototype meets the requirements for macro-architecture specification. It is purely based on TLM models and hence does not depend on any software or hardware implementation. With the right modeling libraries, the virtual prototype is sufficiently easy to create, accurate, and fast. The flexible allocation of tasks to resources enables fast exploration of application mapping options. Under the umbrella of macro-architecture specification, a workload-based Virtual Prototype can be used for the following specific tasks:

- HW/SW partitioning by exploring the mapping of tasks to different number and types of processing resources.
- Bus/memory optimization by using the traffic generated by VPUs as stimuli for configurable TLM models of the interconnect and memory sub-system.
- System level power analysis by combining the workload-based performance model with a power analysis.

## A. *Queuing Task Library*

While the high timing fidelity of TLM is ideally suited for chip-level architecture models, it is too detailed for chip-to-chip protocols, like e.g. PCIe, Ethernet, SerDes, CAN, etc. Therefore, we define a special task library targeting high-level modelling of system-level application data flows and chip-to-chip communication protocol. It is also built upon the Y-chart based approach described above. This way we extend the state-of-the-art methodology from chip-level architecture analysis to the end-to-end analysis of multi-chip systems. The new Queuing Task Library (QTL) is inspired by the concepts of queuing theory, which is a well-known tool for analytical system performance estimation [9], [10]. As depicted in Figure 2, in Queuing networks, the timing of distributed systems is modeled as network of job queues and servers. Queues represent the service requests or jobs while they wait for their turn and servers represent processing resources or service stations.

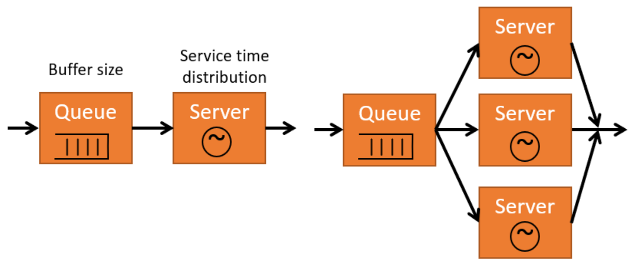

Figure 2: Single Server and Multi-Server Queuing System

Servers sequentially select jobs from the input queue according to a queuing discipline like e.g. priority-based or round-robin, and then consume an amount of service time while processing jobs. The service time can be constant for all jobs, but can also be dependent on the specific job type. For a multi-server system, several queues and servers are connected with each other. Queuing networks are very suitable to detect performance bottlenecks in case of shared resources on distributed systems. Several parameters like queue size, queuing discipline, size of requests, and service time influence the performance of such a system and can be explored to get best configuration.

As illustrated in Figure 3, we generalize the concept of queuing networks by defining a set of building blocks for modelling application data flows and chip-to-chip communication protocols. As stated earlier, QTL models data flow at the packet level as opposed to the detailed TLM approach for modeling on-chip protocols. This is better suited for modelling of network protocols, which follow the OSI reference model [16]. QTL inherits the flexibility of the y-chart based task graph methodology, by separating application workload models from hardware resources. Also, QTL can be combined with TLM models, so you can mix QTL based system-level models with TLM based SoC models. This link between QTL and TLM is of key importance to accurately model the impact of chip-to-chip communication onto SoC hardware resources.

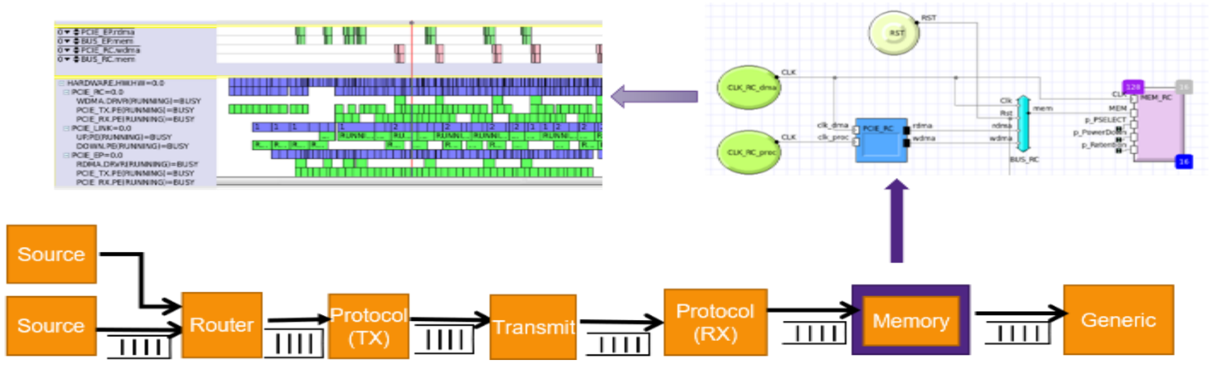

Figure 3: Example data flow with QTL elements (bottom). The QTL Memory task is mapped to the TLM DMA controller in the hardware platform (upper right corner) to generate AXI transactions as shown in transaction trace in upper left corner

### *B. QTL Library Components*

This paragraph presents the key components of the Queueing Task Library in more detail.

1) QTL Packet - In a queuing network as shown in Fig. 3, the QTL packet is analogous to the job requesting service from the server. Represented by an abstract data structure, the QTL base packet carries common attributes of protocol data units (PDUs), for example payload size, source and destination address, command type, priority, etc. The protocol agnostic generic QTL packet can be extended to carry protocol specific attributes like sequence id, checksum, header length etc.

2) QTL Source Task - The QTL source generates QTL packets and is configurable to mimic typical traffic scenarios. The randomization of packets interarrival time, along with different packet attributes like size, priority etc. enables system architects to specify realistic system use cases. These traffic flows represent the application requirements that drive the architecture optimization process. When modelling dependent traffic flows, the source task can also act as externally triggered QTL packet generator.

3) QTL Queue - The QTL queue is the eponymous element in a queuing network and connects all components in the task graph. It buffers jobs before they are processed by a server. By analyzing the queue fill levels and queuing times, the system architect can find bottlenecks in a queuing network. A certain amount queueing is required to avoid backpressure and stalls due to variations in the processing delay of individual task, but growing queues indicate insufficient processing capabilities in the server. This leads to longer end-to-end service latencies and should be addressed by properly dimensioning the corresponding processing resources.

4) QTL Router Task - The router task can multiplex separate network flows into one flow and vice versa also demultiplex one flow into separate flows. It offers scheduling policies like round robin and priority based for combining flows. The demultiplexing can be configured based on packet attributes like priority, flow id, etc. This way the router is a key element for modeling the complex dataflow topologies, where different network flows are funneled through shared network links and are then processed separately by the processing nodes.

5) QTL Generic Protocol - The generic protocol task represents protocol layers of a networking protocol according to the OSI reference model [16]. The generic protocol models packet processing latencies and represents one of the servers in a queueing network. It can act both as transmitter (TX) and receiver (RX) in the data flow. If configured in TX mode, it wraps incoming packets as payload in a new PDU, adding a configurable size header specific to the protocol being modeled. If configured in RX mode, it unwraps the payload and strips the header to pass the packet to the outgoing interface. The protocol task also supports segmentation and reassembling of the packets as required by the maximum PDU size of the lower protocol layer.

6) QTL Transmit Task - QTL Transmit task models the network link in the data flow of the communication network. It also represents a server in Queuing terminology. It models link transfer delays based on packet sizes and can be mapped to a physical communication medium, e.g. Ethernet cable, in which case the latencies incurred are dependent on link configuration.

7) QTL Memory Task - The QTL memory task acts as the link between the abstract QTL packet-based flow for chip protocol modelling, and the more accurate TLM flow for device architecture exploration. Based on the QTL packet attributes, it converts incoming QTL packets into on-chip bus transaction. As shown in Fig. 3, a QTL memory task is mapped to Virtual Processing Unit in the hardware platform, which generates TLM traffic based on request issued by QTL memory task. The bus ports of the VPU are connected to the interconnect and memory subsystem and together they represent the SoC architecture. This way the memory task models the dynamic or workload dependent performance effects like interconnect arbitration, and shared memory latencies.

Taken together, the components in the QTL library provide a modular framework for the construction of application workload models. To model protocol specific functionality or timing, the QTL infrastructure allows to extend the basic packets, tasks, or queues. The customization interface provides means to model any network communication protocol like Ethernet, PCIe etc. The following case study will highlight this in more detail.

## IV. CASE STUDY

We have applied the QTL methodology to the design of a next generation SmartNIC [17] device. This section describes how QTL based modeling has been used for performance exploration of a SmartNIC device connected to a server CPU over a PCIe interface.

### A. *Introduction*

Smart Network Interface Cards (SmartNICs) provide the opportunity to increase performance and reduce power consumption of data centers by offloading generic server CPUs from network processing tasks. However, designing SmartNIC SoCs is very challenging, as they need to cope with exponentially growing network speeds as well as with rapidly changing Software Defined Network (SDN) application stacks. FPGA-based SmartNICs are well suited to provide software-like programmability while providing hardware-like performance [18]. To combine these conflicting goals, FPGA-based SmartNICs integrate dedicated ASIC blocks, such as Network on Chips (NoC), embedded SRAMs, transceivers, accelerators, and I/O protocol blocks into a heterogenous architecture.

It is a challenge to architect FPGA based SmartNICs, as semiconductor companies do not have insights to all data flows that Cloud Service Providers (CSPs) map onto the FPGA. Pre-RTL architecture modeling design space exploration is required to analyze and optimize how the network data flows map onto a mix of FPGA and ASIC resources. In the following section we describe the construction of a QTL based model of the network data flow in combination with a TLM based model of the SmartNIC SoC platform.

### B. *Development and Validation of HW-SW Models*

Figure 4 shows the system-level data-flows, where the left side represents the host CPU and the right side is the FPGA based SmartNIC device. The lower right corner shows the core of the SmartNIC pipeline that is running on the FPGA soft logic. The host uses the VirtIO 1.0 [19] protocol to initiate requests to the SmartNIC device over the PCIe interface. The sending (TX) and receiving (RX) of PCIe transfers is modeled using customized protocol tasks from the QTL library.

The SmartNIC SoC is the focus of the architecture study, so the hardware platform is modeled with accurate TLM components (hatched box in Figure 4). The QTL memory task acts as adaptor that converts QTL packets into AXI transactions and vice versa. This adaptor layer is not explicitly drawn out in simplified block diagram shown in Figure 4. The Memory Rd/Wr Operation block represents the detailed TLM models of the interconnect and memory sub-system with NoC, DDR controller, etc. This accurately models the latencies for exchanging packets between the PCIe subsystem and the FPGA fabric. These latencies depend on application properties like packet size and priorit y as well as hardware parameters, like e.g., bus width and bus frequency. The packet router block routes host requests and responses to the corresponding VirtIO protocol stack, which issue the corresponding commands to the SmartNIC pipeline.

All the above architecture models are instrumented with analysis monitors to record power and performance metrics like throughput, latency, contention, and utilization. The following paragraphs describe the modeling of the PCIe and VirtIO protocol stacks in more detail.

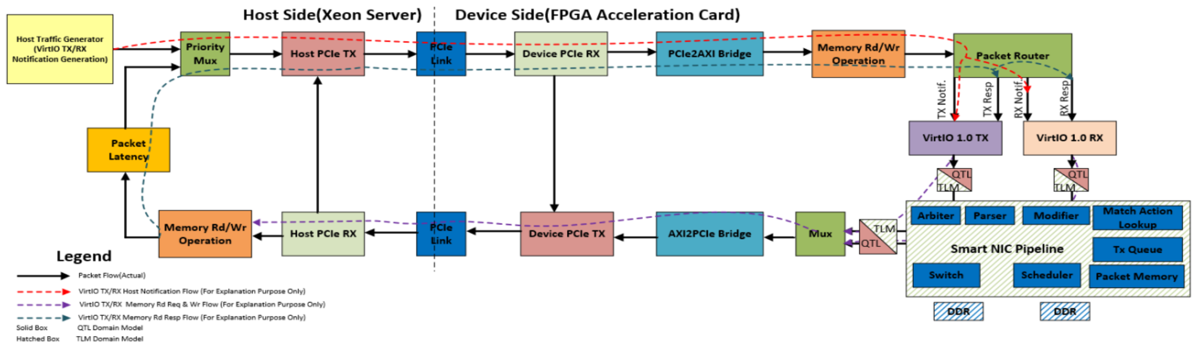

Figure 4: End-to-end system performance model of an CPU host (left) and FPGA based SmartNIC (right).

**QTL PCIe Modelling** – The PCIe protocol layers and physical transmission are abstract models, where only the relevant attributes as well as the size of headers and payload are modelled. This allows the modelling of link delays based on the packet size. The model of the QTL PCIe stack follows the PCIe protocol specification [20] with transaction layer, data link layer and physical layer. A full PCIe system includes protocol stacks for transmitting (TX) and receiving (RX) packets.

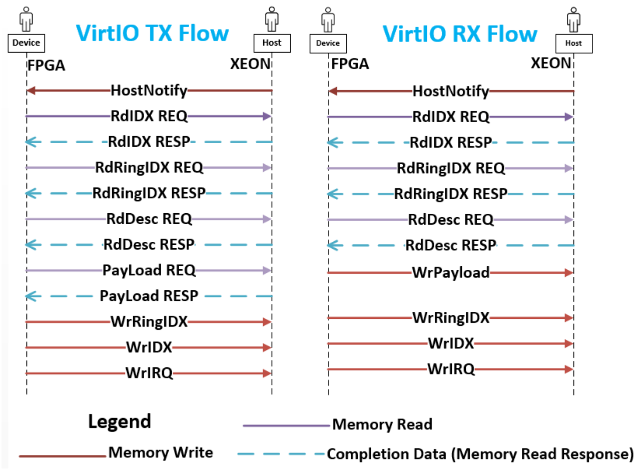

Figure 5: VirtIO 1.0 sequence chart of TX-RX flows

QTL VirtIO 1.0 Model – The VirtIO 1.0 protocol defines two data transfer flows: host to device (H2D, VirtIO TX) and device to host (D2H, VirtIO RX). Fig. 5 shows the message sequence chart of both VirtIO TX and RX flows. Both are required to execute one full duplex packet transfer from the host to the device and back.

VirtIO based transfers between host and NIC device involve multiple steps of memory writes (e.g. host notification, ring update, etc.) and memory reads (e.g. ring index fetch, descriptor fetch, etc.) over the PCIe link. These data flows between host and FPGA based SmartNIC are modelled using QTL library elements.

Fig. 6 shows the implementation detail of the VirtIO 1.0 TX block in Fig 4. Each step of the VirtIO protocol sequence is modelled as a node in a QTL based data flow graph. The starting node of each data flow is a QTL source task which generates either a memory read or write request towards the host. The next memory operation is triggered either by the completion of previous read operation or write request. The first read operation is triggered by the host doorbell notification. The responses from the read requests arrive at the QTL router which routes it to the starting node of next VirtIO step based on the task id. The Mux in Fig. 6 is again a QTL Router block which multiplexes request coming from different source tasks and forwards them to host over the PCIe link. The VirtIO RX protocol stack is modelled accordingly.

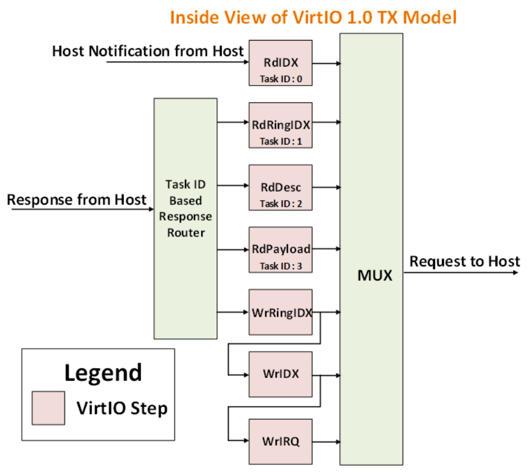

Figure 6: VirtIO 1.0 TX Model (Workload Domain)

## V. Results

This section summarizes the results of the Smart NIC SoC architecture study using the QTL based modeling methodology. We first show the accuracy of the abstract QTL model and then illustrate the performance gains that have been achieved.

To measure the accuracy of the QTL PCIe model, we compared the raw read and write throughput of a PCIe Gen4 x16 connection between the QTL PCIe model and an accurate RTL simulation. Figure 7 shows the average relative error % plot between model and RTL result across various microbenchmarks and packets sizes. It shows that the overall relative error is less than 1%, reflecting a very good correlation between model and RTL results.

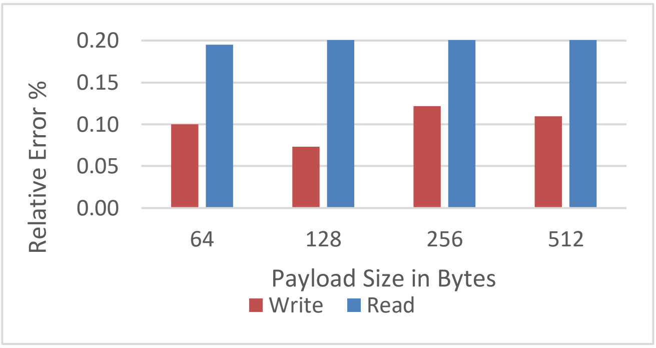

Figure 7: Accuracy of QTL PCIe model for Gen4x16 configuration

To measure the accuracy of the QTL VirtIO 1.0 model, we measured the VirtIO1.0 duplex throughput using the Gen3 x16 DPDK loopback test **Error! Reference source not found.**. We compared the model result with DPDK loopback hardware setup result to measure the relative error. The DPDK loopback test show less than 8 % error against silicon measurements, which is very acceptable for early architecture exploration. We used the DPDK loopback testbench model for KPI projection and design space exploration with respect to various PCIe, VirtIO and AXI parameters for a next generation Gen4 x16 SmartNIC device. The VirtIO 1.0 performance improved by 80% between the previous Gen3 x16 and next generation Gen4 x16 SmartNIC.

Further investigations are done to study how the PCIe link bandwidth (H2D, D2H) is utilized among different VirtIO steps, like e.g. descriptor fetch, payload fetch etc. We observed that H2D RdPayld downlink utilization is just ~20% out of total PCIe H2D downlink utilization. The remaining 80% are consumed due to VirtIO overheads, like e.g. Ring Fetch, Descriptor Fetch etc. Upon further investigations, we found out that these VirtIO overhead steps, which require small size data fetch or write i.e., RdRingIDX (2Bytes), cause significant inefficiency on the bus interface between the PCIe controller and the VirtIO processing in the FPGA fabric. Due to the inefficiencies of these overhead transfers with small data size, the bus interface gets saturated. To improve the performance, we suggest combining multiple descriptor fetches in VirtIO 1.1 in order to reduce the inefficiency due to VirtIO overhead transfers. The modelling and analysis of the enhanced VirtIO 1.1 protocol took 2 weeks of efforts. This illustrates the great modeling efficiency of the QTL modelling paradigm and the high impact of system level analysis. We also studied the sensitivity of the throughput with respect to different server configurations. Each server configuration represents increased loading of server with multiple workloads gradually (Load increases from 1 to 6) which causes higher packet latency and lower throughput as shown in Figure 8. Such sensitivity analysis enabled us to choose the optimal configuration of PCIe tags needed to pipeline transfers and this way hide the server latency.

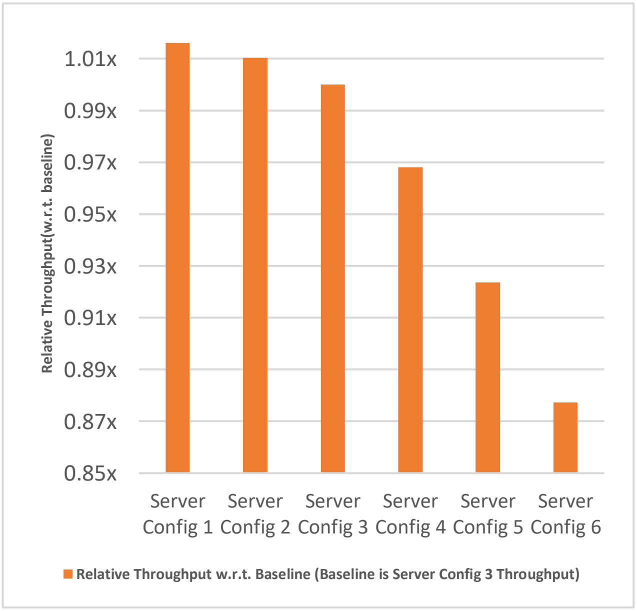

Figure 8: Server Configuration Sensitivity Analysis, 64B Packet Relative Throughput vs Different Server Workload Config

## VI. Conclusion

This Paper shows how QTL based data flow modeling of communication networks can help with the early performance exploration of multi-chip systems. The effectiveness of this approach is demonstrated by a SmartNIC case study, where the QTL based model enabled to left-shift performance analysis and optimization by 6 months. The methodology is accurate enough for reliable KPI projections of next gen architecture as proven by accuracy experiments which showed 99% accuracy for PCIe and 93% accuracy for end-to-end VirtIO when compared to RTL testbenches. Deployment of this methodology allowed us to uncover up to 30 % performance improvements for next-generation architectures for mix of networking and storage workloads, which otherwise we would have discovered late in design cycle causing schedule delays. Compared to hardware-based modelling, the task graph-based modelling paradigm enables quick turnarounds as obvious from the case study, where migrating model from VirtIO1.0 to VirtIO1.1 took only two weeks. It has also proved to be helpful in studying the impact of workload changes and finding optimal hardware configuration (PCIe Tags) as per application requirements.

It is important to highlight that in the past such high-level analysis and detailed debugging of performance issues has been either missed out or done too late in design cycle. The combination of modeling system-level application use-case with SoC-level hardware properties provides a level of accuracy and turnaround time that enables us to do end-end design space exploration for multiple chipsets communicating over PCIe.

## References

[1] Data Plane Development Kit, https://www.dpdk.org

[2] IEEE Standard for Standard SystemC Language Reference Manual. IEEE Std 1666-2011, 2012.

[3] Ghenassia, Frank. “Transaction-level modeling with SystemC”. Springer, 2005.

[4] Kempf, Torsten, Dörper, Malte, Leupers, Rainer, Ascheid, Gert, Meyr, Heinrich, Kogel, Tim, Vanthournout, Bart. “A modular simulation framework for spatial and temporal task mapping onto multi-processor soc platforms” In: Proceedings of the Conference on Design, Automation & Test in Europe (DATE). Munich, Germany, 2005.

[5] Kogel, Tim. “Synopsys Virtual Prototyping for Software Development and Early Architecture Analysis”. In: S. Ha, J. Teich (eds.) Handbook of Hardware/Software Codesign, pp. 1127–1159. Springer, 2017.

[6] Lecler Jean-Jacques, Baillieu Gilles. “Application driven network-on-chip architecture exploration& refinement for a complex SoC”. Design Automation of Embedded System 15(2):133–158, 2011.

[7] Riley, George F., Henderson Thomas R. “The ns-3 Network Simulator”. In: Wehrle K., Güneş M., Gross J. (eds) Modeling and Tools for Network Simulation. Springer, 2010

[8] Varga, András, Hornig, Rudolf. "An overview of the OMNeT++ simulation environment", 1 st Int. Conf. Simulation tools and techniques for Communications Networks and Systems, pp. 1-10, 2008.

[9] Kleinrock, Leonard. “Queueing systems: theory” John Wiley, 1975.

[10] Harchol-Balter, Mor. “Performance modeling and design of computer systems.” Vol. 576. Cambridge University Press, 2013.

[11] Synopsys Platform Architect, https://www.synopsys.com/verification/virtualprototyping/platform-architect.html

[12] Sonntag, Sören, Gries, Matthias, Sauer, Christian. “SystemQ: A Queuing-Based Approach to Architecture Performance Evaluation with SystemC.” In Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS V), volume 3553 of Lecture Notes in Computer Science, 434-444, Springer, July 2005.

[13] Kienhuis, Bart, Deprettere, Ed,Vissers, Kees,Van DerWolf, Pieter “An approach for quantitative analysis of application-specific dataflow architectures” In: Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 338–349, 1997

[14] Keutzer, Kurt, Newton, Richard, Rabaey, Jan, Sangiovanni-Vincentelli, Alberto “System-level design: orthogonalization of concerns and platform-based design”. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19(12), 1523–1543

[15] Advanced Microcontroller Bus Architecture (AMBA) Specifications: https://developer.arm.com/architectures/system-architectures/amba/specifications

[16] Zimmermann, Hubert, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," in IEEE Transactions on Communications, vol. 28, no. 4, pp. 425-432, April 1980

[17] Scott Schweitzer, SmartNIC Architectures: A Shift to Accelerators and Why FPGAs are Poised to Dominate, in Electr. Design, Oct 2020

[18] Firestone, Daniel, Putnam, Andrew, Mundkur, Sambhrama, Chiou Derek, Dabagh, Alireza et al. "Azure Accelerated Networking: SmartNICs in the Public Cloud" in Microsoft Research,2018

[19] Virtual I/O Device (VIRTIO) Version 1.0, http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.html

[20] PCIe specifications, https://pcisig.com/specifications