Designing for FPGAs in the Cloud

Naif Tarafdar, Nariman Eskandari, Thomas Lin, and Paul Chow
University of Toronto

Editor's note:
This article proposes a flow to provision FPGAs from a pool of cloud resources. The proposed flow can lead to more efficient sharing of limited FPGA resources by enabling FPGA development and simulation in virtual machines.

—Mustafa Ozdal, Bilkent University

A CLOUD NETWORK can be seen as a large collection of shared resources, which are then provisioned to users by multiplexing them in time and space. This provisioning is known as infrastructure as a service where users can build large computing infrastructures without the investment of purchasing physical clusters of computing devices. The sharing of these computational resources is done through virtualization, in which a layer of abstraction hides the physical details of the shared resources from the user. These physical details include other users that are sharing the resource, the actual physical location of the resource, and possibly even the physical specifications of the device. This gives the illusion of a complete physical resource to the user. The privacy, performance, and other complications that arise due to the sharing of devices are popular areas of research. While this is still a growing area of research, there exists commercial services such as Amazon EC2 [1], along with academic services such as the SAVI [2] testbed.

Field-programmable gate arrays (FPGAs) have recently been shown to be able to address the power and performance issues being faced by data-centers. The best example with published details is the Microsoft Catapult project, where FPGAs were deployed in the Bing search engine [3]. With only a 10% increase in power and 30% increase in cost, a 95% increase in performance was achieved. The FPGAs were used to implement a part of Bing's ranking engine in custom hardware. The performance and power savings multiply significantly at the scale of a data center.

The challenge of using FPGAs in a cloud is that there has been little infrastructure developed to provision FPGA resources in a way that allows many users to create and interact with their own virtual FPGA compute cluster. In contrast, this problem is much better understood for software-based virtual machines (VMs). What is needed is a complete implementation of a mechanism for provisioning an FPGA cluster within a fully heterogenous environment, where the cluster can communicate with any other network device (be it CPU, another FPGA cluster, or Internet-of-Things device).

Our overall work explores the provisioning of FPGAs from a pool of cloud resources. The FPGAs are provisioned to the user as PCIe devices connected to a virtual CPU in our cloud. These resources are managed with OpenStack [4], which is an open source cloud management tool. Once the user receives a VM with an FPGA, the user programs the FPGA with the Xilinx SDAccel Tool [5]. The SDAccel framework abstracts away much of the FPGA I/O such as the PCIe interface to the host and off-chip memory. The user manages the application region with an OpenCL host application running on the CPU that communicates to the FPGA application (implemented in OpenCL, C++, C, or even HDL).

Digital Object Identifier 10.1109/MDAT.2017.2748393
Date of publication: 8 September 2017; date of current version: 2 February 2018.

This paper focuses on the design and test aspect of our infrastructure, which allows users to develop and simulate an FPGA in a VM, and then migrate their application to a VM with a connected physical FPGA. This tool flow enables more efficient sharing of limited FPGA resources. Previous work has explored the infrastructure to provision large network FPGA clusters within the cloud [6]. We are currently in the process of implementing network function virtualization (NFV) by chaining network functions implemented in FPGAs and CPUs. In this paper, we will explore how the design flow for chaining FPGA functions is done, particularly an incremental design flow done by incorporating chains or part of the chain in software simulation before migrating the functions to FPGAs.

Background

This section reviews the background about OpenStack, FPGAs, software-defined networking (SDN), and NFV.

OpenStack

OpenStack [4] is the cloud management platform used in our cloud data center, and the two main OpenStack services that we employ in our design are Nova and Neutron. Nova is responsible for the deployment of compute resources from the infrastructure, which involves the generation of VMs on physical machines. When a user client requests a VM, they are required to specify a software image and a flavor. The software image refers to the disk image used to generate the VM, which includes the operating system and any other software applications that are required to be installed on the VM. These images are typically kept in a repository and can be updated by users of the data center. The flavor refers to the physical specifications of the VM, such as the number of CPU cores, RAM, and hard drive space.

Neutron is responsible for the provisioning of network resources. We can create network ports within our cluster, and these ports are assigned MAC addresses and IP addresses that will be valid within the cluster. When creating VMs these ports are created implicitly, but we can explicitly create additional ports for nonvirtual devices or nonCPU compute devices.

Field programmable gate arrays

FPGA is a silicon chip with a programmable switching fabric that can be used to implement customized digital hardware circuits. In contrast to the standard CPU environment where the circuitry stays constant and the circuitry performs actions based on instructions, an FPGA changes its circuitry depending on the application.

Logic functions are implemented with the use of lookup tables (LUTs), which essentially implements the truth tables for arbitrary functions. In addition to the LUTs are hardwired flip flops, DSP blocks, memory blocks, and input/output interfaces such as Ethernet and PCIe. FPGA CAD tools synthesize the user-specified design and map it into the physical resources.

FPGAs have traditionally been programmed with hardware description languages that describe the hardware at a very low level, which makes FPGA design inaccessible to software developers. Recently, high-level synthesis has emerged as a new technology that can translate high-level languages such as C, C++, and OpenCL into physical circuit descriptions. Furthermore, FPGA-based platform architectures have made FPGA programming even easier by abstracting many of the interfaces such as PCIe, Ethernet, and off-chip DRAM.

Software-defined networking and OpenFlow

SDN is a concept that enables programmatic control of entire networks via an underlying software abstraction. This is achieved by the separation of the network control plane from the data plane as shown in Figure 1. SDN opens the door for users to test custom network protocol and routing algorithms, and furthermore, it allows the creation, deletion, and configuration of network connections to be dynamic. The current de facto standard protocol for enabling SDN is OpenFlow [7].

In OpenFlow, the control plane is managed by a user program running on a CPU that leverages APIs exposed by an SDN Controller. The SDN controller, often referred to as the "network operating system," abstracts away network details from the user programs. The controller manages the data plane and creates configurations in the form of flows. These flows describe the overall behavior of the network, and can be used to specify custom paths through the network based on packet headers, or even specify operations on the packets themselves (e.g., drop packets, modify headers, and so on). While the switches in the data plane can handle simple header matching and modification of

header fields, more complicated features, such as pattern-matching within the payload or modifying the payload data, require the packets to be forwarded up to the control plane for processing in software. Per-packet software-based processing often incurs significant latencies and reduces line-rate.

This creates an opportunity for FPGAs: FPGAs can combine the best of both worlds with the reconfigurable nature of software programs in the control plane, and the low-latency of the switches in the data plane. An example of a project using FPGAs in SDN can be seen in [8]. This project was implemented with virtualized FPGAs in a data center, where two virtualized FPGAs were inserted into the data path of a network flow. Packets that normally would have been sent to the control plane for custom processing were instead redirected to the FPGAs for processing. Using this approach, the throughput of the packets is the same as a direct path through a switch; whereas when the packets were handled by software running in the control plane, only half the expected throughput was observed.

Xilinx SDAccel

In our design, we use the Xilinx SDAccel [5] platform as an FPGA hypervisor, where the hypervisor is used to provide some basic services. The FPGA in this model is a PCIe-connected device and the platform first provides a driver to communicate to the FPGA. This is done through OpenCL, which provides the API to communicate to and manage computing devices implemented in the FPGA.

OpenCL is both a programming language for heterogeneous devices and a programming API for a host application (conventionally run on a CPU) to manage and communicate to OpenCL compatible devices [9] often connected to the processor via PCIe. In the SDAccel Platform, as shown in Figure 2, the OpenCL API communicates to a driver provided by Xilinx called the hardware abstraction layer (HAL) that provides driver calls to send or receive data from the FPGA and program the application region in the FPGA. The application region is programmed using partial reconfiguration, and the region around the application region is the hypervisor in our model. In this platform, the computing circuit implemented in the application region can be created using high-level synthesis of OpenCL, C, or C++ code or hand-coded Verilog/VHDL. Partial reconfiguration allows the programming of a certain

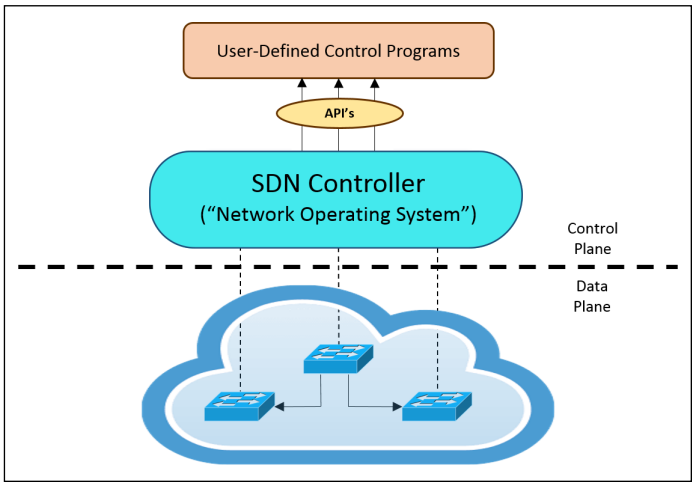

Figure 1. System diagram of an SDN, where user-defined control programs manage network switches.

portion of the FPGA (the application region) without programming the surrounding portions (Ethernet, PCIe, and off-chip memory).

The PCIe Module is a master to a DMA engine to read or write to off-chip DRAM. This is used to communicate data to the application region. The PCIe Module is also a master to a module (not shown) responsible for programming the partially reconfigurable region with a bitstream sent from the user in software. The HAL driver provides an API that abstracts away the addresses required to control the various slaves of the PCIe master.

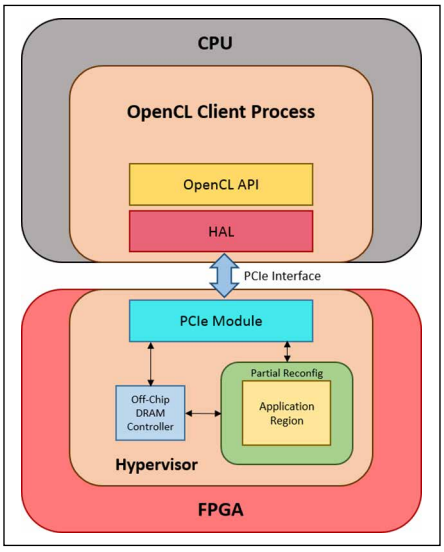

Figure 2. System diagram of the SDAccel platform.

Previous work on FPGA cloud deployment

After our initial deployment, there have been several works implementing FPGAs in cloud environments but details are limited. IBM's SuperVessel looks at providing an FPGA as a cloud resource which shares memory (through CAPI) with a CPU, also provisioned with OpenStack [10]. In this model, a single FPGA is provisioned to the user as an accelerator to which the user can upload FPGA code to be run and compiled onto the FPGA. This simplifies the process of provisioning an FPGA and running code to be accelerated on the FPGA but works with a single FPGA. Microsoft has also continued their work with data-center FPGAs with the second iteration of Catapult [11]. The model here looks at providing a backbone infrastructure for multiple FPGAs to be connected together through a high performance network switch. CPUs are tightly coupled with FPGAs, and the FPGAs are connected to the switch. FPGAs communicate amongst each other through a low-overhead custom transport layer. Finally, Amazon AWS has recently announced that they are introducing Xilinx UltraScale+ VU9P FPGAs connected to VMs via a virtual JTAG connection to their cloud resource pool [12].

Network function virtualization

NFV is a concept for creating novel network services by chaining together individual network functions realized using programmable resources in the cloud. These chained functions can range from standard networking services such as firewalls and load-balancing to more complex features such as deep-packet inspection and intrusion detection. The virtualization of these functions, which are often implemented in software, enables targeted placement of functions closer to end-users, as well as dynamic scaling based on load.

When NFV is deployed in an SDN-enabled network, traffic can be steered through a series of virtualized network functions on-demand. This operation is called service chaining, and has the potential to provide network operators with greater flexibility at reduced operational costs. An example is shown in Figure 3, where an operator forms a chain of virtualized network functions.

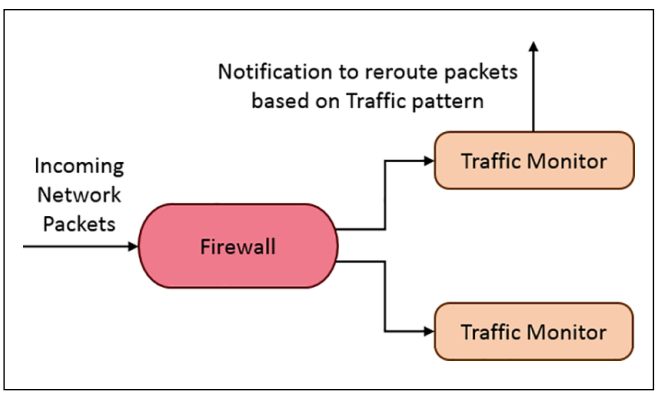

Figure 3. An example of an NFV chain consisting of a firewall and traffic monitors implemented using virtual machines (VMs). In this example, one of the traffic monitors, upon observing a specified pattern, will notify the SDN controller to reroute the network flows accordingly.

FPGAs deployed in our heterogeneous cloud platform

This section describes our infrastructure used to provision FPGAs from a pool of cloud resources.

PCIe passthrough and OpenStack image

First, we provide the FPGA as part of a VM using PCIe passthrough, which is when the VM is given full access to a PCIe device on the physical server. OpenStack notifies the software hypervisor on the physical server of the VM parameters using the flavor discussed in OpenStack. These parameters also include information about any PCIe devices required by the user. This involves configuring the hypervisor to pass control of the PCIe device to a specific VM by adding the PCIe vendor and device ID of the FPGA to the OpenStack configuration script on the physical server. The cloud management system then provisions the VM including the requested PCIe device(s). Figure 4 shows two example VMs with PCIe-connected FPGAs.

Second, we have created multiple OpenStack flavors corresponding to the PCIe devices. Each flavor describes the configurations of the desired VM. These configurations include the number and type (specified by the device ID and vendor ID) of PCIe devices. We made two flavors, one lightweight flavor and another for a full development environment. The lightweight flavor, which consists of only two CPU cores and 2 GB of memory, is intended for the CPU on the VM to act as a mere controller for the FPGA. The full development environment, which consists of four CPU cores and 8 GB of memory, provides a complete environment to create and test FPGA designs as well as control the FPGA.

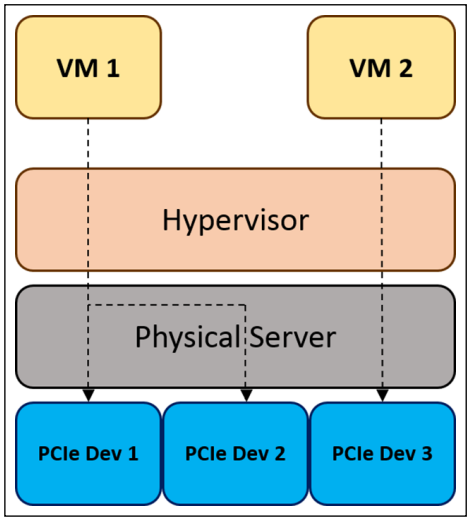

Figure 4. An example of two VMs on a single server. One VM with one PCIe FPGA and the other one has two PCIe FPGAs.

We then create a software image containing the software tools required to communicate to the FPGA. We created two different software images as well. One contains the complete development tool-chain of Xilinx SDAccel Vivado 2015.1. This contains the software required to design, debug, and program FPGAs. We also created a stripped down version of the tool chain to only include the PCIe driver that is used by the CPU to program the application region of the FPGA and to send data to the FPGA.

Design and test environment

We deployed our FPGA cloud service in May 2015. Since then it has been used by students within the University of Toronto as part of their own FPGA development environment. Our infrastructure lays the groundwork for a new design flow that helps utilize and share the FPGAs effectively. This is done through the use of software simulation of FPGAs. The software tools provided within the SDAccel environment allow for simulating the application region completely in software, with no change to the user software application that is calling the application. The simulated application region is wrapped to provide the exact same interface for the HAL as is done in the actual hardware. In this way, the same HAL can be used during software simulation to transfer data to and from the simulated application region.

Our environment gives the user flexibility to provision a VM containing the FPGA development tools with and without a physical FPGA. This creates a new design flow as follows:

- User develops their application on a VM without an FPGA. The user requests a VM with a flavor that does not have the FPGA and the software image containing the FPGA software tools. The user tests their design using the software-simulated FPGA.
- Once the user is ready to migrate their work to a physical FPGA, they save a snapshot of their VM. This is done through an OpenStack API to save the state of a VM.
- The snapshot is then uploaded to the OpenStack software image repository. The user then requests a new VM with a flavor that has the FPGA and the software image snapshot saved in Step 2.
- Now the user can test their application on a physical FPGA. After testing, they can migrate their application back to a VM without an FPGA. They once again will save a snapshot of their VM but this time migrate to a machine without an FPGA.

This design flow allows for easy sharing of the FPGA. Cloud managers can track usage of the physical FPGAs by using monitoring functions provided by OpenStack.

This also has further implications toward reusability of FPGA applications as functions. Similar to software applications such as NFV applications, we can create FPGA applications as virtualized resources, upload the application to OpenStack, and have them available as a software image to be readily available to everyone. We will explore this in the following section.

NFVs using FPGAs

Software-defined networking and OpenFlow describes how FPGAs can be useful in the area of SDN. Furthermore, we can easily implement NFV chains using virtualized network functions and SDN as described in network function virtualization. However, as the implementation of these functions are typically done in software, they have the potential to cause bottlenecks and reduce network line-rate. Our infrastructure allows us to create virtualized network functions with FPGAs that can be extremely beneficial in networking applications and services. An example of NFV application is shown in Figure 3. We can update this configuration as shown in Figure 5.

In Figure 5, each VM is connected to an FPGA using PCIe passthrough. Each VM listens for network packets on their network interface, offloads the processing and computations to the FPGA, and if necessary outputs them back onto the network. We are currently in the process of enabling 10-Gb network connections directly on the FPGAs so they can receive and send packets from the network without requiring any interaction from the VM other than configuration. Eliminating the VM from the data path will reduce latency and allow full network line rate to be maintained. Incorporating the FPGA design within the OpenStack environment allows for an incremental design flow, similar to that described in design and test environment. The design flow of NFVs with FPGAs in our environment is as follows:

- Implement all parts of the NFV design as a software NFV as shown in network function virtualization. Each function is an OpenStack image that contains a software application listening to the network port, performing a function, and outputting to the port.
- Implement and test each individual network function as an FPGA-offloaded design. The design flow for each individual component is highlighted in design and test environment.
- Incrementally, as we complete each part of the NFV chain, swap the software-based function with the FPGA-based implementation.
- If the NFV chain remains functionally correct, then repeat Steps 2 and 3 for the next part of the chain. Repeat until the whole chain is implemented using FPGAs.

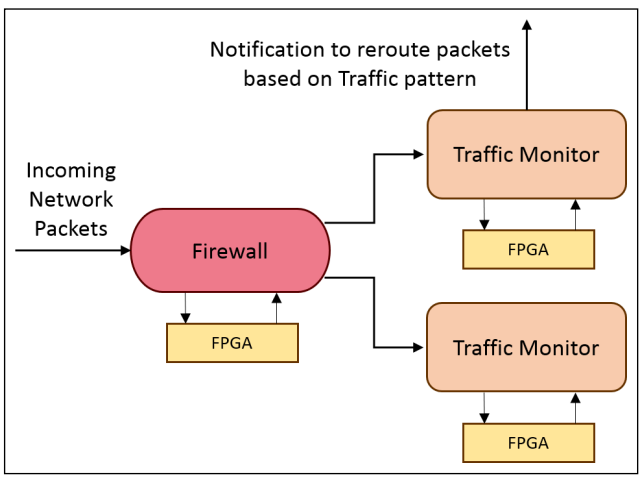

Figure 5. An example of the NFV chain implemented with FPGAs. In this example, all the NFVs in the chain are implemented on an FPGA, but this can be chained with other NFVs implemented in software and other implementations.

Results and case study

Our infrastructure uses partial reconfiguration to program the application region managed by the SDAccel Hypervisor on the Alpha Data 7v3 board which uses a Virtex 7 FPGA. The hypervisor occupies 14.4% of the LUTs, 8.79% of the Flip-Flops, and 15.5% of the BRAM on the FPGA, leaving the rest of the FPGA to be used by the application. Our hypervisor provides the user an interface to memory, PCIe, and 10 Gb/s Ethernet.

The FPGAs are provisioned using PCIe passthrough. The time to provision a VM with an FPGA and without an FPGA is the same in our data center which ranges from 1 to 3 min (as long as there exists a server with an FPGA that has not been provisioned to a VM).

FPGAs can perform better in applications in which require stream processing. This is an application model popular in networking and multimedia applications. For demonstrating an FPGA network function, we use a string-matching application. The string-matching application traverses through an entire packet searching for a collection of strings, if all the strings are found the packet is forwarded to the output. On the FPGA, this is handled at line-rate as this is implemented with a simple shift-register. Our software implementation of this uses the Intel DPDK [13] library which provides a low-latency networking interface to the network packet allowing us to bypass the traditional network stack. Until 30 strings the software implementation can keep up at line-rate, after which we become CPU bound and we see performance dropping to 30% of the packets which continues to exponentially decrease as we increase the number of strings to match, dropping about 80% of the packets with 140 strings. The FPGA experiences no packet drop as we increase the number of strings.

This shows that we can implement a more efficient version of the application on an FPGA. This is a perfect use-case for our incremental design flow where we can first create a version in software and migrate to an FPGA implementation.

Future work

We are also looking into creating a library of NFV blocks to be used within our data center. Other

areas of future exploration include: debugging platforms, which we can investigate by rerouting packets between chains and feeding them to a global debugger, partitioning of a large circuit into multiple FPGAs, and the complete virtualization of the FPGA.

In this paper, we have described how we have used OpenStack to provision FPGAs that are available as a cloud-based resource. By leveraging the VM model and features within OpenStack, we show that it is possible to design and test an FPGA application entirely in software before committing to the actual hardware implementation. This is particularly useful when many FPGAs are chained in an NFV application as the FPGAs can be individually tested in hardware, while the rest of the chain is still running in software. While FPGAs must be handled very differently from processors running software, it is still possible to leverage existing platforms, such as OpenStack, to accommodate the inclusion of FPGAs into the cloud. ■

Acknowledgments

We would like to thank the SAVI testbed for providing the infrastructure, NSERC, Xilinx, and CMC/emSYSCAN for providing the equipment and funding for this project.

■ References

[1] Amazon Web Services Inc., *Amazon Web Services*, 2014. [Online]. Accessed: Nov. 16, 2016. Available: http://aws.amazon.com

[2] J.-M. Kang et al., "SAVI testbed: Control and management of converged virtual ICT resources," in *Proc. IFIP/IEEE Int. Symp. Integr. Network Management*, 2013, pp. 664–667.

[3] A. Putnum et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. 2014 ACM/IEEE 41st Int. Symp. Comp. Architecture*, 2014, pp. 13–24.

[4] O. Sefraoui et al., "OpenStack: Toward an open-source solution for cloud computing," *Int. J. Comp. Appl.*, vol. 55, no. 3, pp. 38–42, 2012.

[5] Xilinx Inc., *SDAccel Development Environment*, 2016. [Online]. Accessed: Nov. 12, 2016. Available: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[6] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network FPGA clusters in a heterogenous cloud data center," in *Proc. Int. Symp. Field-Programmable Gate Arrays,* 2017, pp. 237–246.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford et al., "OpenFlow: Enabling innovation in campus networks," in *Proc. ACM SIGCOMM Comp. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[8] S. Byma, N. Tarafdar, T. Xu, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Expanding OpenFlow capabilities with virtualized reconfigurable hardware," in *Proc. FPGA ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays,* 2015, pp. 94–97.

[9] The Khronos Group, *OpenCL Standard*, 2015. [Online]. Available: https://www.khronos.org/opencl/

[10] IBM Research, *OpenPOWER Cloud: Accelerating Cloud Computting*, 2016. [Online]. Available: https://www.research.ibm.com/labs/china/supervessel.html

[11] A. Caulfield et al., "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–13.

[12] Amazon, *Amazon EC2 F1 Instances*, 2016. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[13] DPDK Intel, *Data Plane Development Kit*, 2014. [Online]. Accessed October 12, 2016. Available: http://dpdk.org

Naif Tarafdar is currently investigating ways to provide FPGAs as a sharable resource within cloud environments. He is a first year PhD candidate at the University of Toronto.

Nariman Eskandari is a first year MASc candidate at the University of Toronto. His current research is focused on providing easy ways to use FPGAs in cloud environments.

Thomas Lin has been part of the Smart Applications on Virtualized Infrastructure project helping to develop a future application-platform testbed since 2012. He is a third year PhD candidate at the University of Toronto with a focus on software-based infrastructure management for heterogeneous clouds.

Paul Chow holds the Dusan and Anne Miklas Chair in Engineering Design at the Department of Electrical and Computer Engineering, University of Toronto. His primary research interests are focused around making FPGAs easily usable as computing devices.

■ Direct questions and comments about this article to Naif Tarafdar, University of Toronto, Toronto, ON, M5S, Canada; e-mail: naif.tarafdar@mail.utoronto.ca.