A Framework to Enable Runtime Programmable P4-enabled FPGAs in the Open Cloud Testbed

Zhaoyang Han*, Suranga Handagala‡, Kalyani Patle†, Michael Zink†, Miriam Leeser*
Northeastern University *‡, University of Massachusetts Amherst †
Email: *zhhan,mel@coe.neu.edu, ‡s.handagala@northeastern.edu
†kpatle,mzink@umass.edu,

***Abstract*—This paper presents a framework for cloud users who wish to specify their experiments in the P4 language and map them to FPGAs in the Open Cloud Testbed (OCT). OCT consists of P4-enabled FPGA nodes that are directly connected to the network via 100 gigabit Ethernet connections, and which support runtime reconfiguration. Cloud users can quickly prototype and deploy their P4 applications through our framework, which provides the necessary infrastructure including a network interface shell for the P4 logic.**

We have provided several examples using this framework that demonstrate designs running at the 100 GbE line rate with the support of runtime reconfiguration for P4 functions. By combining an existing network interface shell and P4 toolchain on FPGAs, we offer a framework that enables users to rapidly execute their P4 experiments in real time on FPGAs.

***Index Terms*—FPGA, Partial Reconfiguration, P4, Networks Systems**

I. Introduction

P4 is a rapidly developing language for network data plane programming that had a large impact on recent networking research. It allows programmers, as opposed to network device vendors, to define and program the network. Significant progress on programmable networks has been made based on the success of the P4 language.

To fully utilize the power of P4 and exploit its performance in the data plane, network researchers need a hardware testbed for P4 related experiments. FPGAs present an attractive hardware solution for programmable network infrastructure as they are highly programmable and relatively inexpensive. Enabling P4 on FPGAs brings a lot of opportunities in different areas of network research, such as in-band telemetry, security and in-network computing. Moreover, enabling partial reconfiguration on FPGAs supports runtime programmable networking, a topic which has recently attracted the interest of network researchers.

While several tool flows exist for enabling P4 on FPGAs, there is no unified framework and open testbed for P4 on FPGAs. Network researchers who are unfamiliar with hardware programming thus do not have an easy path to hardware development. This paper introduces a P4-enabled FPGA network testbed under the Open Cloud Testbed (OCT) project. We present a small but evolving network testbed with multiple nodes with FPGA as P4-enabled SmartNICs. To enable P4 in the OCT testbed, a runtime-reconfigurable framework and toolchain have been explored and tested.

This work is supported in part by NSF grant CNS-2130891, CNS-2130907, CNS-1925658, and CNS-1925464.

The contributions of the work presented in this paper are:

- Building FPGA-enabled nodes with network connections in OCT for a P4-enabled testbed;
- Exploring and creating FPGA designs for **P4-enabled** FPGA SmartNICs;
- Using FPGA Partial Reconfiguration for **runtime-reconfigurable** P4 functions;
- Providing multiple examples as tutorials and demonstrations for the cloud and networking research community.

II. Background and Related Work

A. Open Cloud Testbed

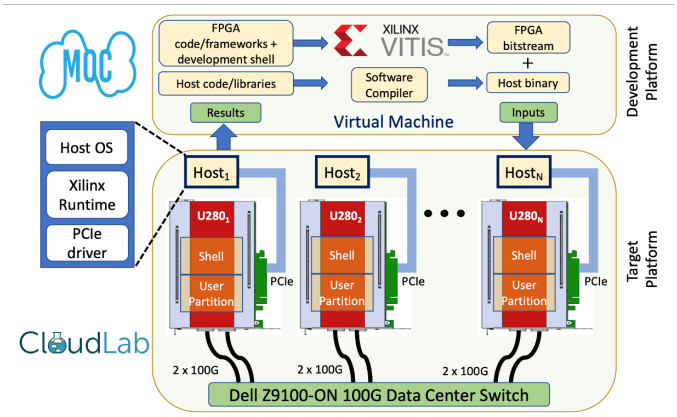

Fig. 1. Overview of OCT FPGA development and test flow [28]

The Open Cloud Testbed [28] provides a research-oriented experimentation testbed for systems researchers who focus on cloud platforms. Testbeds like OCT deliver the necessary hardware and software with cloud and HPC services to researchers in both the cloud and system communities, enabling more experimental-based research opportunities and directions.

OCT currently consists of a total of 5,172 cores and 63TB of RAM distributed over 237 servers, thanks to the generous donations by industry partner Two Sigma among others. 16 FPGAs, namely AMD/Xilinx's Alveo U280 data center cards with High Bandwidth Memory to support data-heavy tasks,

are currently provided in OCT along with a toolchain that supports the development of bitstreams [14].

Fig. 1 illustrates the development and the target platforms we have created in OCT. For the development platform, AMD/Xilinx development tools are instantiated as part of a virtual machine image in the Massachusetts Open Cloud (MoC). An experimenter specifies their design and uses the provided tools to generate an FPGA bitstream; the output corresponds to the hardware logic along with necessary runtime drivers. This FPGA bitstream with drivers is copied to OCT nodes and loaded onto the FPGAs. The final results are computed on the FPGA nodes and may be sent back to the development nodes. A detailed description of the development workflow and step-by-step tutorials are available [18], [28].

In 2022, support for direct network connections from the FPGAs was added [12]. Traditionally, an FPGA connects to a host processor via PCIe buses. In the network attached configuration, the FPGA is also directly connected to the network through two independent 100GbE ports. Several applications with direct FPGA-to-FPGA communications via TCP and UDP FPGA stacks have been demonstrated [12]. In this paper, we explore the potential research opportunities in providing more complex and flexible network services by enabling P4 specification of designs for the OCT FPGAs.

B. FPGAs in Runtime Programmable Networks

With the development of the P4 language [5] and programmable network devices, progress has been made in granting users direct access to the control and data planes. Various data plane programming researches like data plane disaggregation, cryptography and data plane machine learning have been conducted using FPGAs [19], [20], [26]. Recently, the network research community has begun to explore not only programmable networking, but runtime programmable networking as well [24]. As summarized by Xing et al., runtime programmable networks can create multiple novel use cases: dynamic applications, real-time security, live infrastructure customization, and tenant extensions. Enabling reconfigurable P4 programming on FPGAs in OCT encourages this current trend in network research.

Compared to a dedicated P4 SmartNIC, an FPGA is highly programmable and can be more easily customized and scaled. For example, a dedicated P4 SmartNIC with rigid data flow, like AMD's Pensando Distributed Services Card (DSC-200) does not provide the ability for data path customization. In contrast, Firestone et al. [11] developed and deployed FPGA-based SmartNIC on Azure for offloading network functions. They found that FPGAs are well suited for the dynamic handling of network traffic and can reduce the main CPU's compute load. Through partial reconfiguration [22], FPGAs are capable of replacing part of their hardware logic during runtime while the remaining design continues to run. With this technology, FPGAs are suitable for advanced network research topics including multi-tenant networking [17] and runtime programmable networking [10].

C. Related Work

Since the first release of the P4 language, there have been several efforts to enable P4 on FPGAs and achieve programmable networking. Several different approaches to solving this problem exist. P4FPGA [23] is an open-source P4 compiler extending the original P4 compiler with a custom Verilog generator. It can translate high-level P4 into hardware logic. NetFPGA [27] provides an open-source modular-based platform for rapid prototyping of network devices on FPGAs. VitisNetP4 (SDNet) [2] is a vendor-specific toolchain to convert P4 into bitstreams for AMD/Xilinx FPGAs. In more recent work [13], they provide P4 support for their NetFPGA platform. Yan [25] implemented an inter-data center communication solution that uses a vendor specific (Raymax), P4-enabled SmartNIC. In our design, we utilize VitisNetP4 as our P4 to hardware tool, as the FPGAs in OCT are from Xilinx. The VitisNetP4 toolchain provides a user-friendly environment for development and simulation. Our first example enabling P4 on an FPGA starts with AMD/Xilinx VitisNetP4 example. We gained insights from the Xilinx example that combines a P4 block with a network interface shell (OpenNIC shell [1]) and integrated the toolchain into our OCT network testbed. Our toolchain is able to map different P4 logic onto the FPGAs and provide several custom features. The details of such integration are described in Sect. IV.

The StaRR-NIC has applied partial reconfiguration on the openNIC shell [3], [4]. It demonstrated the potentiality of having interchangeable user functions in the openNIC shell. Our framework allows users to swap between different P4-spcific functions during runtime and provides research opportunities. ESNet [9], collaborating with AMD/Xilinx, recently published their toolchain of combining VitisNetP4 with OpenNIC shell, which provides a similar tool flow to our work. Their approach supports full automation at the cost of flexibility. In contrast, our work focuses on a P4-enabled testbed that allows fast prototyping from P4 to FPGA and focuses on flexibility. We provide a set of examples that teach experimenters how to use the toolchain and the testbed. While our approach is less automated than the one provided by ESNet, it offers more flexibility in terms of functionality that can be implemented via P4.

III. P4 Testbed in OCT

This section discusses a sub-testbed built in OCT using the P4-enabled SmartNIC for programmable network research. As shown in Fig. 2, there are currently four FPGA nodes allocated for the P4-enabled SmartNIC testbed. The number of nodes can be increased for larger experiments. These FPGAs connect through a Dell 100GbE switch with two 100GbE links each.

Similar to the nodes in Fig. 1, each node is made up of a host processor and FPGA connected through the PCIe bus. On the host processor, the P4 runtime driver controls the P4 IP block on the FPGA. Like normal network interfaces, our FPGA SmartNIC supports both DPDK and kernel-level network drivers. We designed the P4 overlay specifically for

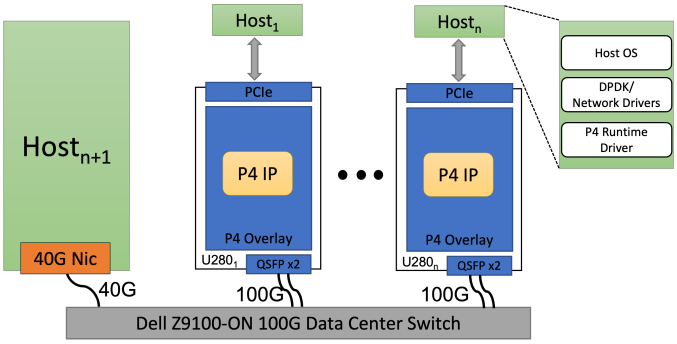

Fig. 2. OCT P4 Testbed

P4 functions, as our shell on the FPGAs. Details of this overlay will be described in Sec. IV.

In addition to the FPGA nodes, a single host node with 40GbE NIC is also part of the testbed for basic loopback tests and verification. In addition, this host-only node can be network programmable and allow user customization via the eBPF framework [7]. eBPF extends kernel abilities by adding C functions into the network stack. Enabling eBPF in the OCT testbed is one of our future research directions. The OCT P4 network testbed consisting of different classes of network endpoints satisfies various network research needs. Although the switch is not programmable, it is possible to re-arrange the topology and substitute it with another FPGA node as a two-port switch. Note that each FPGA supports two ports directly connected to the network.

IV. P4 Overlay on FPGA

To enable P4 on the OCT FPGAs, we have used an overlay design as a P4 function container. While serving as a standard 100G network interface for the host processor, this overlay design allows cloud users to insert additional functionality into their network by plugging in custom P4 functions. Additionally, this design is partial-reconfigurable so that the P4 logic can be re-programmed during runtime. In this section, we will describe the design of this overlay and its features. In Sec. IV-C, we will discuss the design verification and simulation process.

A. Framework Design

This overlay uses AMD/Xilinx's OpenNIC shell [1], an open-source FPGA-based 100G NIC platform, as the backbone design, Fig. 3(a). The OpenNIC shell includes a Queue-based Direct Memory Access (QDMA) for transferring packets between the NIC shell and host CPU through the PCIe bus, and a 100G MAC block (CMAC) for Ethernet support. The user plugin is a single pipe that connects QDMA and CMAC. As illustrated in Fig. 3(b), we fill the block *User Plugin@250Mhz* with the P4 hardware IP block generated by AMD/Xilinx's Vitis Networking P4 (VitisNetP4) [2]. The VitisNetP4 toolchain will generate hardware IP from P4 source code. The OpenNIC shell provides NIC functions for supporting 100Gb/s throughput. Our framework creates the necessary packets and control logic paths between the OpenNIC shell and VitisNetP4 outputs.

With this approach, a user only needs to provide P4 code as input, and can quickly generate a P4-enabled NIC and achieve fast deployment on the OCT using this tool flow. It provides an easy way for network researchers unfamiliar with FPGA or hardware design to conduct their own research on OCT.

To enable a runtime programmable network, we have also utilized Partial Reconfiguration (PR) as described in Sec. II to divide the design into two parts: the static region and the dynamic region, as shown in Fig. 3(c). Cloud users only need to feed P4 code as input and generate P4 hardware IP with particular interfaces using VitisNetP4. A partial bitstream rather than a complete design bitstream is produced by using the partial configuration script provided.

B. Features

1) Fast and Secure FPGA Programming: The common way to program Xilinx's Alveo U280 is to load the bitstream via JTAG or program the flash-based configuration memory device via JTAG. However, as a low-level hardware access tool, JTAG is also exposed to potential attacks. The FPGA and host PC are both vulnerable in this case [8]. To prevent this, we use the PCIe Base Address Register (BAR) instead of JTAG to program the FPGA. As the PCIe bus is faster than JTAG, it will reduce the time to load a bitstream.

The PR design is also helpful for fast and secure programming. A study on Amazon's commercial cloud (AWS) FPGA nodes, AWS F1 instances, that use an AWS shell to wrap users' accelerators using partial reconfiguration shows that static region provides an extra layer of protection to the device [21]. Instead of giving full control of the hardware to the cloud user, this separation provides more control over FPGA devices and helps protect FPGA resources from malicious users. In addition, the partial bitstream has a relatively smaller size compared to programming the entire FPGA, which will reduce programming time. A detailed programming time comparison is provided in Sec. V.

2) Runtime Reconfigurability: With FPGA partial reconfiguration, the FPGA NIC is runtime reconfigurable. This enriches the features of this network testbed and allows dynamic P4 application switching. For example, a cloud user can change the network application from telemetry to a firewall without shutting down nodes for reconfiguration. Also, having more dynamic regions can accommodate multiple tenants, making network virtualization possible in the testbed.

3) User Externs: P4 externs are objects manipulated by P4 programs, which provide extra functionality to P4 programs. These externs are defined and provided by P4 architectures. Since VitisNetP4 uses a narrowly-defined P4 architectural model Xilinx Switch Architecture (XSA) that only supports limited functionaility, extern objects are big components for XSA. It is important to support user externs on our testbed to provide more flexibility. In our P4 framework, user externs are available and examples are provided. Arbitrary extern logic can be generated from a hardware description language or

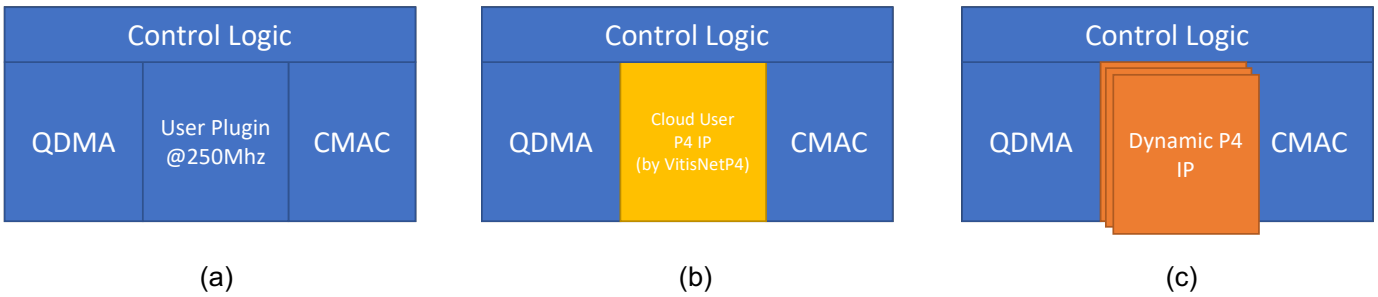

Fig. 3. OCT P4 Framework (a): OpenNIC Shell; (b): OpenNIC Shell with P4 Logic; (c): Partial Reconfigurable P4 Framework

created from C using high-level synthesis. These logic blocks connect with P4 IP through pre-defined interfaces.

C. Design Simulation and Verification

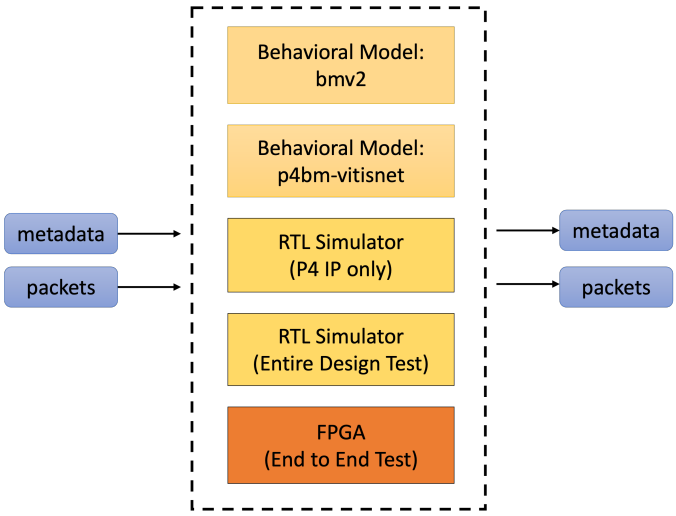

Fig. 4. Design Simulation and Verification

To simulate the design and verify its correctness, we performed several levels of simulation and verification as shown in Fig. 4. There are three block-level simulations for verifying the P4 logic itself. The two lower levels are for system-level verification. To start, cloud users may have their P4 program running in MiniNet on BMv2, such as the examples provided with the P4 tutorials [15]. VitisNetP4 will take P4 programs and then generate corresponding hardware P4 IPs. There are two levels of verification for the P4 IP. One is p4bm-vitisnet, which is similar to BMv2, serving as Xilinx's behavioral model for the P4 program. The lower level is Register Transfer Level (RTL) simulation. At this level, the hardware logic of the source code is verified by comparing the RTL simulation results with Xilinx's behavior results. This is used to verify the P4 hardware logic generated by VitisNetP4.

Another process is RTL simulation for the entire system. Such a simulation testbench can be created using cocotb [6], a Python-based verification framework. Once the system is programmed onto FPGAs, an end-to-end test can be run on the testbed nodes. Packets are sent and received between nodes. The design is verified by comparing the received packets with standard output packets from the behavioural model in the previous step.

V. Examples and Results

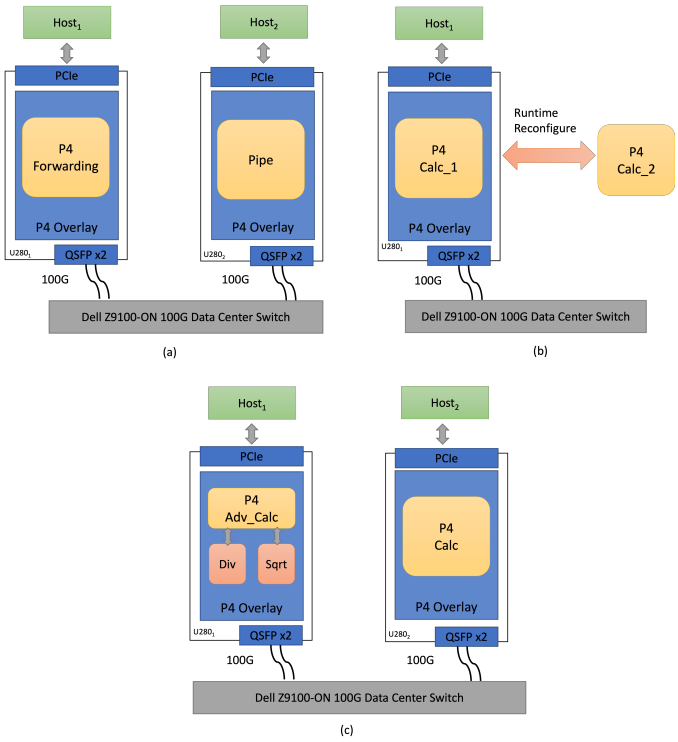

Fig. 5. Examples: (a) Forwarding; (b) Reconfigurable calculator; (c) Advanced calculator with externs

In this section, we present three examples that demonstrated the features of our tools and the OCT testbed: forwarding, a basic calculator that makes use of partial reconfiguration and an advanced calculator that makes use of externs. These examples are shown in Fig. 5. Forwarding and the basic calculator are examples from the P4 tutorials [15]; forwarding, the basic calculator and the advanced calculator are included in the Xilinx VitisNetP4 toolchain demos. We have combined them with NIC on FPGAs using our framework and tested them in OCT, and added partial reconfiguration as a new feature.

A. Forwarding

TABLE I
FPGA PROGRAMMING TIME COMPARISON BETWEEN BITSTREAMS

	Size (Byte)	Loading time	Speed
Complete bitstream	40068175	23 seconds	1.66MB/s
Partial bitstream	5631808	3 seconds	1.79MB/s

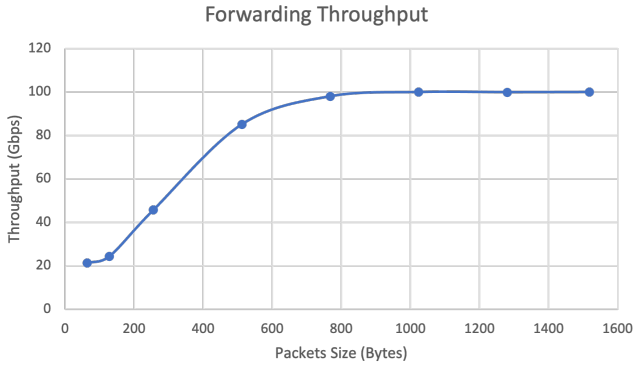

Fig. 6. Forwarding Performance

This example achieves packet forwarding for packets with specific source or destination IP addresses. All other packets will be dropped. It is implemented as shown in Fig. 5(a). For this example, we ran the forwarding application on one FPGA node and set the other FPGA node as a pipe that allows any packet to pass. We send packets from the pipe node to the FPGA node with DPDK running on both hosts. The performance is measured using pktgen [16]. The packet size is varied from 64 to 1572 bytes, yielding the performance results shown in Fig. 6. Note that the connection utilizes less than 50% of the available bandwidth with a lower packet size. The bandwidth saturates when the packet size is 600 bytes per packet or higher.

B. Reconfigurable Calculator

In this example, Fig.5(b), we demonstrate how a design is partially reconfigured. This example is the basic calculator example from the P4 tutorial, and performs in-networking computing in the data plane. This P4 application will execute one 32-bit operation chosen from among addition, subtraction, and multiplication; and three bitwise operations for packets with specific headers. Based on the original calculator example from the P4 tutorials, we create two different calculator instances, calc_1 and calc_2. The two instances consist of two different sets of the above six operations. For the same packets and match action tables, the two calculators will perform different operations and produce different result packets. We thus have two different P4 source codes that provide different functionality. These two P4 codes are sent to the toolchain as input. After processing, two complete bitstreams with the entire design are generated. Each complete bitstream contains a different P4 calculator. These two bitstreams are similar to the general bitstreams that can configure the entire FPGA. In addition, two partial bitstreams only containing the Dynamic P4 region are also generated by the framework. While loading a general bitstream requires the entire system to be shut down, loading the partial bitstream does not. In addition, the latency for downloading the partial bitstream is significantly faster than that for the general bitstream, as shown in Table I.

C. Advanced Calculator with Extern Functions

This example (Fig.5(c)) illustrates how extern functions can be combined with the P4 logic in the framework. Two nodes are connected via the switch. Both nodes are capable of processing the packet with specific headers for all six basic calculator operations. However, the left node provides extra operations: division and square root. In addition to the functions supported by the basic calculator, the advanced calculator increases its operations with 32-bit division and 64-bit square root functions. The division will return a 32-bit quotient and a 32-bit remainder. Both functions are not defined in the XSA. Therefore, two extern functions are introduced into the design to provide extra power in the calculations. These two hardware blocks are generated from Xilinx's arithmetic library and then connect to the P4 harware block as shown in the figure. As a result, with using extra 1% of the resources on FPGA for extern functions compared to the original calculator design, the FPGA provides two extra operations for the network processing.

VI. Discussion and Future Work

In this section, we discuss some open issues with our testbed and the potential for future improvements.

A. Floorplanning

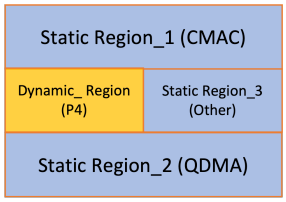

Fig. 7. Floorplan: Illustration of Blocks' Actual Positions on FPGA

Fig. 7 shows the floorplan of our overlay design on FPGA. As shown in Table II, only a small portion of the entire FPGA on-board resources are used. In addition, as shown in the figure, only a small area has been allocated to the P4 dynamic region. The static region of the design is larger than it needs to be. This is a result of the OpenNIC shell from AMD/Xilinx as well as partitioning decisions. While our initial design is a proof-of-concept, better partitioning between regions is needed to fully harness the power of the FPGA board.

The P4 dynamic region is under-utilized under the current floorplanning scenario. We can easily increase the size of the P4 region to allow more complicated designs with compute-intensive or data-intensive user externs. It is also possible to allow multiple P4 regions instead of one. With multiple P4

regions, it can achieve the virtualization of FPGA resources for multiple tenants.

B. Partial Reconfiguration

We have created a framework for P4 that supports partial reconfiguration, as demonstrated in Sec. V. With our current implementation, the P4 region is empty during reconfiguration and this leads to packet losses. We are investigating ways to continue to receive packets during reconfiguration. Any such solution would need to be incorporated as part of the static region in the design.

TABLE II
UTILIZATION

	LUTs(%)	RAMs(%)	FFs(%)	DSPs(%)
Full design	7.46	4.08	5.04	0.03
static_region_1	2.57	1.76	4.10	0
static_region_2	18.01	0.89	9.53	0
dynamic_region	3.79	0	3.25	0.39

VII. Conclusion

This paper introduces a P4-enabled network testbed in the Open Cloud Testbed (OCT) for advanced network topics. The main component provided in this testbed is runtime programmable P4-enabled FPGAs nodes. Based on the user demand, a certain number of such nodes can be allocated for their experiments in the OCT. Beyond the basic P4 programmability, our framework is also capable of transferring user-defined P4 externs into hardware logic and provides an option for runtime reconfiguration. These features broadly extend the coverage of different network-related experiments. We use three examples to demonstrate these usages in our framework. Analysis of these example evaluations shows that this framework offers a new approach for conducting network research in our testbed. The examples discussed in this paper are publicly available from github.com/OCT-FPGA/P4OpenNIC_Public.

References

[1] AMD OpenNIC Project. https://github.com/Xilinx/open-nic, 2022. [Online; accessed 01-01-2023].

[2] Vitis Networking P4. https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html, 2022. [Online; accessed 01-01-2023].

[3] Agarwal Anup, Daehyeok Kim, and Srinivasan Seshan. StaRRNIC: Enabling Runtime Reconfigurable FPGA NICs. Technical Report CMU-CS-23-100, Carnegie Mellon University, 2023.

[4] StaRR-NIC. https://github.com/StaRR-NIC/starrnic-public, 2023. [Online; accessed 02-28-2023].

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, July 2014.

[6] cocotb. https://www.cocotb.org, 2022. [Online; accessed 01-01-2023].

[7] eBPF - Introduction, Tutorials & Community Resources. https://ebpf.io, 2022. [Online; accessed 01-01-2023].

[8] Maik Ender, Amir Moradi, and Christof Paar. The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1803–1819, 2020.

[9] ESNet SmartNIC. https://github.com/esnet/esnet-smartnic-hw, 2022. [Online; accessed 01-01-2023].

[10] Yong Feng, Zhikang Chen, Haoyu Song, Wenquan Xu, Jiahao Li, Zijian Zhang, Tong Yun, Ying Wan, and Bin Liu. Enabling In-situ Programmability in Network Data Plane: From Architecture to Language. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 635–649, 2022.

[11] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure Accelerated Networking:SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[12] Suranga Handagala, Miriam Leeser, Kalyani Patle, and Michael Zink. Network Attached FPGAs in the Open Cloud Testbed (OCT). In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2022.

[13] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. The P4->NetFPGA workflow for line-rate packet processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 1–9, 2019.

[14] Miriam Leeser, Suranga Handagala, and Michael Zink. FPGAs in the Cloud. *Computing in Science & Engineering*, 23(6):72–76, 2021.

[15] P4 Tutorial. https://github.com/p4lang/tutorials, 2022. [Online; accessed 01-01-2023].

[16] pktgen. https://pktgen-dpdk.readthedocs.io/en/latest/, 2022. [Online; accessed 01-01-2023].

[17] Mateus Saquetti, Guilherme Bueno, Weverton Cordeiro, and Jose Rodrigo Azambuja. P4VBox: Enabling P4-based switch virtualization. *IEEE Communications Letters*, 24(1):146–149, 2019.

[18] S.Handagala. OCT FPGA Tutorial. https://github.com/OCT-FPGA, 2021. [Online; accessed 01-01-2023].

[19] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. Flightplan: Dataplane disaggregation and placement for p4 programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 571–592, 2021.

[20] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. Taurus: a data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1099–1114, 2022.

[21] Steve Trimberger and Steve McNeil. Security of FPGAs in data centers. In *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, pages 117–122, 2017.

[22] Kizheppatt Vipin and Suhaib A Fahmy. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. *ACM Computing Surveys (CSUR)*, 51(4):1–39, 2018.

[23] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. P4FPGA: A Rapid Prototyping Framework for P4. In *Proceedings of the Symposium on SDN Research*, pages 122–135, Santa Clara CA USA, April 2017. ACM.

[24] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Hongyi Liu, Matty Kadosh, Alan Lo, Aditya Akella, Thomas Anderson, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. A Vision for Runtime Programmable Networks. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 91–98, Virtual Event United Kingdom, November 2021. ACM.

[25] Yan Yan, Arash Farhadi Beldachi, Reza Nejabati, and Dimitra Simeonidou. P4-enabled Smart NIC: Enabling sliceable and service-driven optical data centres. *Journal of Lightwave Technology*, 38(9):2688–2694, 2020.

[26] Abbas Yazdinejad, Reza M Parizi, Ali Dehghantanha, and Kim-Kwang Raymond Choo. P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking. *Computers & Security*, 88:101629, 2020.

[27] Noa Zilberman, Yury Audzevich, Georgina Kalogeridou, Neelakandan Manihatty-Bojan, Jingyun Zhang, and Andrew Moore. NetFPGA: Rapid prototyping of networking devices in open source. *ACM SIGCOMM Computer Communication Review*, 45(4):363–364, 2015.

[28] Michael Zink, David Irwin, Emmanuel Cecchet, Hakan Saplakoglu, Orran Krieger, Martin Herbordt, Michael Daitzman, Peter Desnoyers, Miriam Leeser, and Suranga Handagala. The Open Cloud Testbed (OCT): A platform for research into new cloud technologies. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 140–147. IEEE, 2021.