SVFF+: Kubernetes FPGA virtualization and reconfiguration for network virtualization

Samuele Paone, Giorgos Armeniakos, Stefanos Gerangelos, Michele Paolino, Daniel Raho
Virtual Open Systems SAS, Grenoble, France
{s.paone, g.armeniakos, s.gerangelos, m.paolino, s.raho}@virtualopensystems.com

***Abstract*—Network virtualization concepts are extending their influence to other fields, such as Satellite Communication. The WAVE consortium initiative is an example, as they aim to create a new SATCOM ecosystem with interoperable and hardware accelerated virtual functions. This paper presents SVFF+, a software/hardware framework that enables easy FPGA usage in virtualized environments, introducing partial reconfiguration and Kubernetes (K8S) support. SVFF+ was developed with WAVE consortium applications in mind, but it can be extended to any cloud computing or network virtualization use cases, such as NFV. We demonstrate the effectiveness of SVFF+ via benchmarks, showcasing its potential to accelerate network applications and cloud environments. SVFF+ enhances a previuos work (SVFF) with support for Kubernetes and partial reconfiguration.**

***Index Terms*—FPGA virtualization, network virtualization, cloud computing**

I. Introduction

Network virtualization was born with the intent of decoupling software functions from their customized hardware platforms implementation. These concepts were born with telecom applications in mind, but today they are extending their influence also to other fields, such as Satellite Communication (SATCOM). As a matter of facts, the WAVE [1] consortium is an IEEE initiative that aims to create a new SATCOM ecosystem composed by standardized and interoperable components. WAVE is significantly relying on Network Function Virtualization (NFV) concepts to transform SATCOM custom hardware boxes into virtual functions, and is relying on FPGA hardware accelerators to provide the right level of performance and efficiency required by waveform processing.

In fact, FPGA accelerators notably provide a very good performance per watt ratio. However, they are harder to program if compared with other accelerators i.e., GPUs. As a result, the FPGA programmability and the possibility to change hardware kernels at runtime, remains unadressed [2] in many virtualized infrastructure managers such as Kubernetes (K8S), OpenStack, etc.

In this paper we present SVFF+ a solution to enable an easy FPGA usage in virtualized environments targeting network virtualization applications such as SATCOM, NFV, etc. SVFF+ extends an earlier work named SR-IOV Virtual Function Framework (SVFF) [3] with partial reconfiguration and K8S support. A set of benchmarks are included to prove the efficiency and feasibility of the approach.

This work was funded by the European Union under Horizon Europe grant 101092850 (project AERO).

The two key contribution of this paper extends SVFF to create SVFF+: first, we develop a K8S plugin to use FPGAs in orchestrated containers/VMs, and second we add support for partial reconfiguration. SVFF+ was developed with WAVE consortium applications in mind, but it can be extended to any cloud computing or network virtualization use cases, such as NFV.

The paper is organized as follows: Section II provides background about SVFF, K8s and partial reconfiguration, while Section III describes the contributions on this paper on both K8S and hardware partial reconfiguration sides. Benchmarks details are provided in Section IV and finally Sections V and VI provide related works and conclusions.

II. Background

In this section we describe SVFF (the FPGA virtualization framework on which this work is based), as well as we introduce Kubernetes and the Xilinx Hardware Partial Reconfiguration basic concepts.

A. SR-IOV Virtual Function Framework(SVFF)

SR-IOV Virtual Function Framework(SVFF) [3] (Figure 1) is a software/hardware framework that leverages SR-IOV to automate the creation, attachment, detachment, and reconfiguration of accelerators to different guests. SR-IOV, or Single Root I/O Virtualization, is a technology that allows a single physical PCIe device to present itself as multiple Virtual Functions (VFs) that can be directly attached to guests. This enables efficient resource sharing and improved performance by allowing direct access to the hardware, reducing latency and CPU overhead in virtualized environments.

More in detail, the SVFF hardware part is composed by a custom design that makes use of the Xilinx QDMA IP. On the other hand, the software part is composed by custom OS drivers and programs/scripts that improve FPGA flexibility and usability in virtualized environments automating several operations. The interaction with the hypervisor is based on libvirt, which makes SVFF portable to different virtualization solutions.

As a result, the framework is able to automate the following operations in a virtualized environments:

- **Virtual Function management:** VFs initialization, creation and configuration.

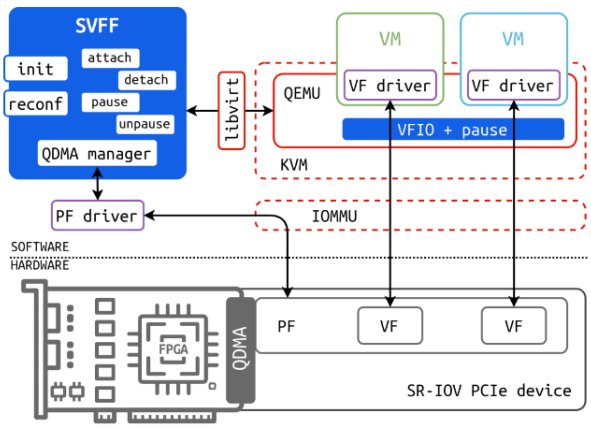

Fig. 1. SR-IOV Virtual Function Framework overview [3]

- **FPGA bitstream flashing:** ability to flash the FPGA bitstream using the xilinx *xsct* tool. SVFF also automatically detaches all VFs from VMs prior to flashing operations. This is required to make sure that SR-IOV is able to safely detach accelerators from VMs. It is valid only on systems that do not implement Partial Reconfiguration as described in Section III.
- **Resources monitoring:** ability to see at runtime which VF is attached to each guest and what accelerator is idle/busy. This is done deploying a custom QDMA kernel driver and saving resources information on an XML file. Such information is then used to describe the VF to be attached/detached to one of the guests, as well as to enable a safe detachment of the accelerator from the guest.
- **Accelerator API library:** enable user space application access to the accelerator functionalities. This API hides implementation details (memory addresses, registers, etc) to provide functions like accelerator init, run and destroy. This is accelerator specific and needs to be customized when changing accelerator.

B. Kubernetes (K8S)

Kubernetes (K8S) [4] is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. The Kubernetes architecture consists of a control plane and multiple worker nodes. The former includes one or multiple controller nodes that manage the cluster orchestrating and scheduling resources. Worker nodes, on the other hand, run kubelets (applications).

At its core, Kubernetes provides a framework to run microservices based application resiliently. It manages the lifecycle of containers, which are lightweight, portable units that package an application and its dependencies. This allows developers to focus on writing code without worrying about the underlying infrastructure.

One of the key features of Kubernetes is its ability to orchestrate containers across a cluster of machines. This means that if one container fails, Kubernetes can automatically restart it or move it to another machine, ensuring high availability. Additionally, it can scale applications up or down based on demand, making it an efficient choice for businesses that experience fluctuating workloads. Kubernetes also offers a rich ecosystem of tools and services. It supports various networking options, storage solutions, and monitoring tools, allowing teams to customize their environments to meet specific needs. With its declarative configuration model, users can define the desired state of their applications, and Kubernetes will work to maintain that state. It supports also the execution of applications in virtual machines via a plugin called KubeVirt, that similarly to SVFF uses libvirt and KVM. This improves security because virtual machines notably provide better isolation than containers, and in addition provide advantages in multi-tenant environment by isolating tenants containers. Summing up, Kubernetes is widely adopted in network virtualization applications and is a key target for the WAVE consortium. For these reasons, we selected K8S and Kubevirt for the orchestration of our FPGA virtualization solution.

C. Partial Reconfiguration

Partial reconfiguration (PR) has recently gained significant attention in FPGA virtualization, enabling a single FPGA to host multiple accelerators concurrently in distinct PR regions [5]. However, hardware expertise requirements, constraints due to device architectures, and limited tool support have limited more widespread adoption. Many FPGA virtualization systems still lack PR integration [6], [7], while Amazon's AWS EC2 F1, though employing PR, restricts usage to a single tenant, contradicting multitenancy goals. In this work, alongside state-of-the-art approaches [8], [9], we leverage Xilinx Dynamic Function eXchange (DFX) to enable the reconfiguration of designated FPGA regions while the rest of the device continues operation. This allows dynamic switching of functionalities within the FPGA. In addition, we exploit recent improvements and the new media configuration access port (MCAP) of Xilinx UltraScale devices to enable a fast, direct PCIe interfacing and supports arbitrary applications from multiple users sharing the same FPGA. Our FPGA design implementation, along with static and partial bitstreams, are generated using Xilinx Vivado 2022.1, and the Alveo U55C platform, operating at 250 MHz, is used as the target device. Table I presents the FPGA resource utilization of the static (connecting 4 VFs) and dynamic (reconfigurable) regions of our design, solely, alongside the reconfiguration time required for swapping between implemented QAM-4 and QAM-16 kernel. As shown, with 16 parallel VFs, less than 24% of the entire FPGA is utilized (the static part equals to 3.1%, and 16 VF equals to 16*1.3%), leaving much space for extra user logic design. Moreover, the overhead in resources due to our dynamic reconfiguration support is less than 1%, which is negligible compared to the baseline design without DFX, and is due to the small additional logic for extra interfaces (such as MCAP) and managing the partial reconfiguration process.

TABLE I
FPGA UTILIZATION OF STATIC AND DYNAMIC REGIONS

Regions	LUTs	FFs	DSPs	BRAM	Reconf. Time
Static	5.4%	3.1%	0.0%	4.9%	N/A
Dynamic (per VF)	0.7%	1.3%	0.3%	0.0%	405ms

III. FOREGROUND

In this section we detail the key contriutions of this paper to enable Kubernetes and partial reconfiguration in SVFF, thus creating what we call SVFF+.

A. Cloud integration and K8S plugin description

Figure 2 is an high-level view of the SVFF integration with Kubernetes and Kubevirt.

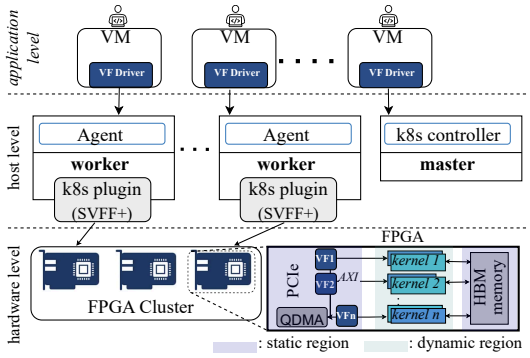

Fig. 2. SVFF+ foundations: integration with Kubernetes/Kubevirt and partial reconfiguration

In practice, in order to integrate SVFF with K8S, a new plugin has been developed and installed on every worker machine that is equipped with an FPGA.

Every worker node who wants to use our virtualization solution must program the FPGA with the SVFF+ custom design. At this point, the K8S plugin is responsible of detecting the FPGA and make it visible to the Kubernetes cluster so that it can deploy FPGA accelerated guests. Thanks to SR-IOV and the work done on this custom design, every FPGA kernel corresponds to a Virtual Function (VF) that can be passed-through to the guest. Guests can use the assigned kernels independently from each other. Eventually, it is also possible to swap kernels so that the guests can even use different kind of accelerators based on their necessities.

The key features of the SVFF+ K8S plugin are:

- **Recognize the VFs specific to the design on FPGA**, eventually letting the administrator decide which VF has to be visible to the Kubernetes cluster.
- **Attach and detach VFs** from the guests at runtime, thus giving the opportunity to swap VFs between guests.
- **Prepare the FPGA for the pass-through** (create VFs, unbind from host driver, bind to vfio-pci). This is necessery because KubeVirt does not support this natively.

The plugin handles the low level details related to virtualization and SR-IOV (i.e., the interactions with the SVFF+ linux kernel driver) so that it is possible to have a fine grade control over these details eventually from a GUI.

More in details, the SVFF+ driver implements QDMA related functions as well as monitoring functionalities that allows the framework manager to know if a kernel is actually in use, through a lock mechanism; this helps accelerators lifecycle management preventing the detachment of a VF while it is in use.

Summing up, our plugin takes care of detecting the SVFF FPGA design while VMs deletion/creation is managed by KubeVirt. VFs allocation to the VMs can be done with the standard K8S methodologies, e.g., via YAML description.

B. Partial reconfiguration implementation

As shown in Fig. 2 (FPGA details at the bottom right), we integrate partial reconfiguration support via the Xilinx Dynamic Function eXchange technology for each FPGA device in our Kubernetes cluster. Each device comprises static regions, hosting QDMA, HBM memory and AXI interconnections. Dynamic regions on the other hand house the implemented kernels. Each dynamic region is associated with a single VF, enabling kernel swapping as needed. As a proof-of-concept scenario, we implemented two signal demodulation kernels—QAM-4 and QAM-16—representing different telecommunication standards. These Quadrature Amplitude Modulation (QAM) kernels are common in SATCOM communication and employ hard-decision demodulation [10] which calculates the Hamming distance for each received sample and selects the symbol with the minimum distance. Note, however, that users can implement any logic suited to their needs using the provided interfaces. By maintaining the same interfaces and pipeline flow while modifying only the functional logic, SVFF+ delivers consistent performance across applications.

Once generated, the partial bitstreams are stored and managed at the host level. Workers access these bitstreams through the MCAP configuration access port, which is connected to one of the PCIe hardmacros, thus enabling reconfiguration of designated FPGA regions during runtime using Xilinx XVSEC software. This setup ensures seamless bitstream deployment and runtime flexibility while maintaining high scalability across the system.

IV. EXPERIMENTAL ANALYSIS

In this section we assess our SVFF+ FPGA virtualization framework in terms of performance and flexibility. To demonstrate the capabilities of our framework, we evaluated the two QAM demodulation kernels, focusing on their throughput and performance in terms of communication between the kernels, HBM memory, and the PCIe connection with the host server. Additionally, we analyzed the utilization of our devices and the

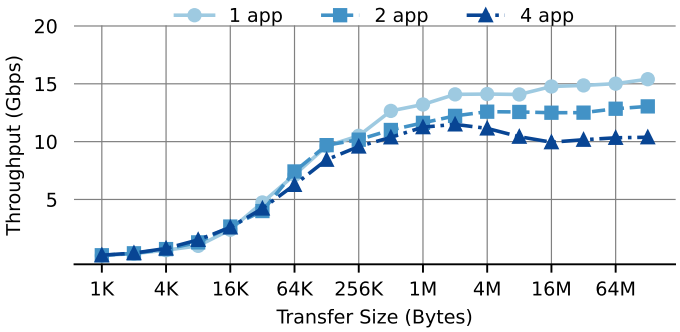

Fig. 3. Throughput of QDMA read operations with respect to varying transfer sizes and concurrent QDMA queue requests.

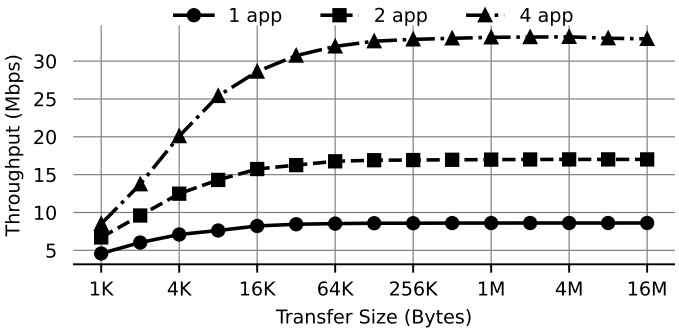

Fig. 4. Round-trip throughput for different numbers of concurrent applications across different transfer sizes, considering both QDMA read, write, and FPGA kernel processing.

resource efficiency achieved through partial reconfiguration, highlighting the dynamic adaptability of our approach.

Our experiments are conducted using a host PC with an Intel(R) Xeon(R) CPU E5-2623 V4 running at 2.6 GHz and a PCIe Gen 3 x16 expansion bus. The host system is a 32-core, 128GB RAM x86 64 server running Red Hat Enterprise Linux (RHEL) 8.7 with Linux kernel 4.18. The implementations of FPGA kernels, along with static and partial bitstreams, were generated using Xilinx Vivado 2022.1, and the Alveo U55C platform, operating at 250 MHz, was used as the target device. Finally, the Kubernetes orchestration layer manages the VFs corresponding to the FPGA kernels, facilitating the deployment of isolated guest VMs that can dynamically utilize and reconfigure the FPGA kernels as needed.

A. FPGA Design and reconfigurable modules

Partial reconfiguration significantly enhances hardware utilization by eliminating the need to implement multiple kernels in parallel. As shown in Table I, partial reconfiguration for only 4 VFs can save over 5% of total resources, with savings increasing almost linearly as the number of VFs grows. On the other hand, the primary overhead introduced is the reconfiguration latency via XVSEC utilities. However, reconfiguring a single VF each time incurs a latency of only $450ms$ (approximately 1 order of magnitude less than of the initial bitstream [11]), enabling our framework to dynamically switch between applications on demand with minimal overhead and without disrupting the execution of other users.

B. Round-throughput analysis

To evaluate our framework's performance in terms of throughput between QDMA, user logic, and FPGA HBM memory, we run an application with one or more simultaneous VF (*app*) instances at a time. First, we analyze the transfer throughput of QDMA read operations with respect to data transfer size, shown in Fig. 3. This figure illustrates throughput during data transfer from the host PC to FPGA HBM memory via QDMA and its performance scaling with multiple concurrent applications. To generate this figure, the average execution time of 1,000 runs was measured for each transfer size. As observed, for data sizes up to 256 KB, QDMA handles traffic independently of the number of running applications, but beyond this point, throughput saturates. Almost identical results (omitted for brevity) are also observed for QDMA write operations, which involve traffic from HBM memory to the host PC through PCIe.

In Fig. 4, we present the round-trip throughput of our system while running various numbers of applications simultaneously, as a function of the transfer size. The process involves transferring plain data from each VM to the FPGA user logic via QDMA/PCIe, performing pipeline kernel computations, and transferring the computed results back to memory. Hardware counters in the FPGA fabric are used for measurements, ensuring no overhead is introduced. As depicted in Fig. 4, throughput increases as transfer sizes grow, but it eventually becomes bounded by the computational capacity of the hardware kernels. Higher parallelization can effectively enhance the system's overall throughput, but only up to the QDMA interface's data rate (see Fig. 3). Note, also, that while adopting an AXI-Stream protocol instead of AXI-lite could significantly enhance these numbers, such optimizations were beyond the scope of this work.

V. Related Work

FPGAs virtualization gathered significant attention from both academic researchers and industry practitioners, resulting in a variety of proposed solutions.

At the resource level, FPGAVirt [12] utilizes virtio and FPGA overlays to support multi-tenant virtualization, streamlining I/O access. Coyote [11], on the other hand, provides PR-based abstractions for FPGA resources, enabling integration with operating systems in cloud environments. While it integrates simple round-robin scheduling, it does not pursue advanced performance optimizations or leverage SR-IOV for enhanced multi-tenant support. Nimblock [2] employs partial reconfiguration alongside advanced scheduling algorithms to enable fine-grained FPGA sharing but incurs unavoidable overhead due to its software-driven approach and limited runtime reconfiguration support through PCIe. FFIVE [6] implements FPGA virtualization with lightweight Docker managed by K8S. However, it lacks support for Partial Reconfiguration (PR) and SR-IOV technology, which limits its adaptability to dynamic network environments. Additional works have

also explored adding operating system-like capabilities to FPGAs [7], [13] adding support for shared memory access, networking, or peripheral access.

Overall, a recurring deficiency of most solutions lack dynamic reconfiguration capabilities and fine-grained management features which are critical for modern multi-tenant environments. Furthermore, almost all of the solutions fail to provide user-friendly integration frameworks, requiring significant effort to implement custom applications or optimize for specific workloads. Our work distinguishes itself from most relevant approaches by offering a flexible infrastructure that leverages management features introduced in SVFF [3] and enables dynamic FPGA resource management, making it well-suited for multi-tenant environments.

Moreover, for what concerns Kubernetes integration, existing similar Kubernetes plugins [14], [15] are able to recognize the FPGAs but target mainly non virtualized environments. For what concerns commercial solutions that enable users to program an FPGA in an orchestrated environment, there is Azure [16] and AWS F2 [17]. These however do not provide full control of the hardware infrastructure. Finally Rosebudvirt [8] provides SR-IOV based virtualization as well as K8S support. However, they strictly define the hardware accelerators as processing units with a RISC-V controller, which might be a limitation for certain applications.

Conversely, the proposed solution gives more freedom to the hardware accelerators programmer while featuring a K8S plugin that enables multiple VMs/containers to use the FPGA in an orchestrated way and with the flexibility of partial reconfiguration.

VI. Conclusion and future work

In this paper we present the results of the extension of an existing FPGA virtualization framework (SVFF) with Kubernetes and partial reconfiguration support, creating what we call SVFF+. A K8S plugin and partial reconfiguration integration development are the key contribution of this paper. We measured hardware accelerators performance to demonstrate the feasibility of the approach.

As part of our future work, on the host/system side we plan to improve the software part in the directions of FPGA resources monitoring and defining a generic API/interface for the FPGA kernels to enable interoperability between kernels.

For what concerns the FPGA side, we aim to improve latency associated with data transfer between the kernels and HBM memory via QDMA, by adopting high-speed network interfaces (such as QSFP cables) for direct data transfer, using AXI-Stream. This approach would allow data to be processed by our kernels and then transferred back efficiently via high-speed links.

References

[1] W. consortium, "Wave consortium website," 2025. [Online]. Available: https://waveconsortium.org

[2] M. Mandava, P. Reckamp, and D. Chen, "Nimblock: Scheduling for fine-grained fpga sharing through virtualization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[3] S. Cirici, M. Paolino, and D. Raho, "Svff: An automated framework for sr-iov virtual function management in fpga accelerated virtualized environments," in *2023 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2023, pp. 1–6.

[4] "Kubernetes website," 2025. [Online]. Available: https://kubernetes.io

[5] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018.

[6] M. Ewais, J. C. Vega, A. Leon-Garcia, and P. Chow, "A framework integrating fpgas in vnf networks," in *2021 12th International Conference on Network of the Future (NoF)*. IEEE, 2021, pp. 1–9.

[7] J. Ma, G. Zuo, K. Loughlin, X. Cheng, Y. Liu, A. M. Eneyew, Z. Qi, and B. Kasikci, "A hypervisor for shared-memory fpga platforms," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 827–844.

[8] Y. Chang and Z. Guo, "Rosebudvirt: A high-performance and partially reconfigurable fpga virtualization framework for multitenant networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 1, pp. 298–302, 2025.

[9] Z. Han, S. Handagala, K. Patle, M. Zink, and M. Leeser, "A framework to enable runtime programmable p4-enabled fpgas in the open cloud testbed," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–6.

[10] I. Stratakos, V. Leon, G. Armeniakos, G. Lentaris, and D. Soudris, "Design space exploration on high-order qam demodulation circuits: Algorithms, arithmetic and approximation techniques," *Electronics*, vol. 11, no. 1, p. 39, 2021.

[11] D. Korolija, T. Roscoe, and G. Alonso, "Do {OS} abstractions make sense on {FPGAs}?" in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 991–1010.

[12] J. Mbongue, F. Hategekimana, D. T. Kwadjo, D. Andrews, and C. Bobda, "Fpgavirt: A novel virtualization framework for fpgas in the cloud," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 862–865.

[13] J. Zhang, Y. Xiong, N. Xu, R. Shu, B. Li, P. Cheng, G. Chen, and T. Moscibroda, "The feniks fpga operating system for cloud computing," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, 2017, pp. 1–7.

[14] Intel, "intel-device-plugins-for-kubernetes," 2018. [Online]. Available: https://github.com/intel/intel-device-plugins-for-kubernetes

[15] Xilinx, "Xilinx fpga_as_a_service," 2023. [Online]. Available: https://github.com/Xilinx/FPGA_as_a_Service/tree/master/k8s-device-plugin

[16] Microsoft, "Fpga attestation for azure np-series vms," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/field-programmable-gate-arrays-attestation

[17] Amazon, "Aws f2." [Online]. Available: https://github.com/aws/aws-fpga/blob/f2/User_Guide_AWS_EC2_FPGA_Development_Kit.md