Harmonia: A Unified Framework for Heterogeneous FPGA Acceleration in the Cloud

Luyang Li
liluyang@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Heng Pan
panheng@cnic.cn
Computer Network Information Center, Chinese Academy of Sciences
Beijing, China

Xinchen Wan
xinchen.wan@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Kai Lv
lvkai20z@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China

Zilong Wang
zwangfb@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Qian Zhao
zhaoqian.cn@bytedance.com
Douyin Co., Ltd.
Beijing, China

Feng Ning
snoopy068@126.com
Douyin Co., Ltd.
Beijing, China

Qingsong Ning
ningqs@foxmail.com
Douyin Co., Ltd.
Beijing, China

Shideng Zhang
zhangshideng@bytedance.com
Douyin Co., Ltd.
Beijing, China

Zhenyu Li
zyli@ict.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Layong Luo
luolayong@gmail.com
Researcher
Beijing, China

Gaogang Xie
xie@cnic.cn
Computer Network Information Center, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Abstract

FPGAs are gaining popularity in the cloud as accelerators for various applications. To make FPGAs more accessible for users and streamline system management, cloud providers have widely adopted the *shell-role* architecture on their homogeneous FPGA servers. However, the increasing heterogeneity of cloud FPGAs poses new challenges for this architecture. Previous studies either focus on homogeneous FPGAs or only partially address the portability issues for roles, while still requiring laborious shell development for providers and ad-hoc software modifications for users.

This paper presents Harmonia, a unified framework for heterogeneous FPGA acceleration in the cloud. Harmonia operates on two layers: a platform-specific layer that abstracts hardware differences and a platform-independent layer that provides a unified shell for diverse roles and host software. In detail, Harmonia provides automated platform adapters and lightweight interface wrappers to manage hardware differences. Next, it builds a modularized shell composed of *Reusable Building Blocks* and employs hierarchical tailoring to provide a resource-efficient and easy-to-use shell for different roles. Finally, it presents a command-based interface to minimize software modifications across distinct platforms. Harmonia has been deployed in a large service provider, Douyin, for several years. It reduces shell development workloads by 69%-93% and simplifies role and software configurations with negligible overhead (<0.63%). Compared with other frameworks, Harmonia supports cross-vendor FPGAs, reduces resource consumption by 3.5%-14.9% and simplifies software configurations by 15-23× while maintaining comparable performance.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

Keywords: Heterogeneous FPGA; Shell-Role Architecture; Cloud Framework; Reusability;

ASPLOS '25, March 30-April 3, 2025, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1079-7/2025/03.
https://doi.org/10.1145/3676641.3716259

ACM Reference Format:
Luyang Li, Heng Pan, Xinchen Wan, Kai Lv, Zilong Wang, Qian Zhao, Feng Ning, Qingsong Ning, Shideng Zhang, Zhenyu Li, Layong Luo, and Gaogang Xie. 2025. Harmonia: A Unified Framework for Heterogeneous FPGA Acceleration in the Cloud. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3676641.3716259

1 Introduction

Field-Programmable Gate Arrays (FPGAs) have garnered attraction in the cloud due to their capacity to enhance performance and efficiency for various cloud applications [15, 19, 55, 77, 98, 102, 104, 105]. Developing applications on FPGAs requires users to not only implement application logic itself but also manage complex I/O connectivity and intricate system integration. Thus, major commercial cloud providers (*e.g.*, AWS [3], Azure [62], etc.) have introduced a *shell-role* architecture on homogeneous FPGA clusters [16, 18, 70]. In this architecture, the FPGA logic is divided into two partitions: the provider-owned region (*a.k.a.*, *shell*) and the user-owned region (*a.k.a.*, *role*) (see Figure 1). The role contains different application logic, while the shell is responsible for managing the FPGAs, providing a range of common services for roles (*e.g.*, Ethernet, DMA, etc.), and handling data and control exchanges with the host software. This architecture significantly alleviates the development burden for users.

However, the growing heterogeneity of cloud FPGAs poses new challenges for the shell-role architecture. First, providers have to build individual shells tailored to new FPGA devices due to significant differences in FPGA architectures [3, 16, 70] and hardware capabilities [32, 50]. For example, the SmartNIC architecture [16], used in networking applications, requires high-speed network interfaces (*e.g.*, QSFP112 [82]) and transport stacks (*e.g.*, RDMA [91]). In contrast, the SmartSSD architecture [50], deployed in storage applications, demands extensive storage capacity (*e.g.*, HBM [41]). Addressing these hardware disparities requires substantial development efforts in shells (§2.3). Second, providers often select FPGAs from different vendors for supply chain security and cost considerations [11]. These vendors define their own hardware interfaces, configurations, and compilation methods. When users migrate applications to new FPGAs, they have to manually adjust both roles and host software, which involves considerable ad-hoc and error-prone modifications (§2.3).

To address the above issues, recent studies present FPGA virtualization [14, 18, 49, 99–101] and portable operating system (OSs) abstractions [44, 47, 59, 79, 103]. These approaches aim to provide *portable roles* that allow users to deploy applications seamlessly on distinct FPGAs. Specifically, FPGA virtualization creates an intermediate layer to deploy roles on platform-agnostic virtual FPGAs, while hardware mappings rely on vendor-specific frameworks [99, 101]. FPGA OSs like

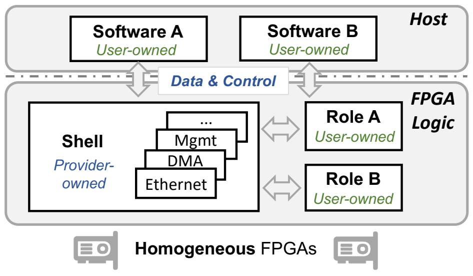


Figure 1. The shell-role architecture. The user-owned *role* contains application logic; the provider-owned *shell* manages FPGAs, provides common services to roles and host software.

AmorphOS [44] and Coyote [47] directly provide a unified interface for roles across different FPGA platforms through dynamic wrappers. However, these methods still require laborious reconstruction of shells and ad-hoc modifications to the host software when migrating to new FPGAs (see §2.3). Commercial frameworks (*e.g.*, OFS [66] in oneAPI [65] and Vitis [86]) provide off-the-shelf shells that support a set of FPGAs. Nevertheless, the design and integration of these shells are closely tailored to specific FPGA series (*e.g.*, Agilex, Alveo, etc.). When deploying applications onto FPGAs with different architectures and peripherals, providers still have to invest substantial efforts in modifying shells to ensure cross-vendor compatibility.

As a result, we move a step further and take the shell and host software development into account, providing a comprehensive framework to address heterogeneity issues. Specifically, this framework should cover the following objectives: (i) providing a unified shell compatible with different FPGAs without laborious development workloads for platform providers; (ii) supporting portable roles with simple interfaces and configurations for applications; and (iii) offering a consistent host interface that without requiring ad-hoc modifications to simplify software integration.

To this end, we present Harmonia, a unified framework for heterogeneous FPGA acceleration in the cloud. Harmonia consists of two layers: a platform-specific layer that abstracts the hardware differences and a platform-independent layer that provides a unified shell for diverse roles and host software. Specifically, Harmonia develops automated platform adapters to manage platform-specific configurations and employs lightweight interface wrappers to cover interface variations (§3.2). To create a unified shell without extensive development workloads, Harmonia proposes a modularized shell composed of a series of *Reusable Building Block* (RBB) abstractions (§3.3.1). To conserve on-chip resources for roles and simplify user configurations, Harmonia provides a hierarchical shell tailoring mechanism to generate application-specific shell instances (§3.3.2). To ease host software development and integration, Harmonia presents a

Table 1. Existing frameworks either target homogeneous FPGA platforms or partially address heterogeneous issues. In this paper, Harmonia aims to address heterogeneity issues comprehensively.

Framework	Heterogeneity	Unified Shell	Portable Role*	Consistent Host IF
Cloud Platform [2, 3, 62]	✘	✔	✘	✔
Virtualization [14, 45, 49, 99–101]	✔	✘	✔	✘
FPGA OS [44, 47, 59, 79, 103]	✔	△	✔	✘
Commercial Framework [65, 66, 86]	✔	△	✔	△
Harmonia	✔	✔	✔	✔

∗: migrating roles to FPGA platforms that have appropriate hardware capabilities requires only minimal modifications.
△: requires laborious development workloads or ad-hoc modifications when deployed on FPGAs from different vendors.

command-based interface that abstracts control operations to reduce ad-hoc software modifications (§3.3.3).

We have implemented the Harmonia framework and deployed in the cloud datacenters of Douyin over several years (§4). It supports a variety of internal applications across tens of thousands of FPGA accelerators, including networking, security, computing, and infrastructure applications (§5). The production results demonstrate that Harmonia can reduce shell development workloads by 69%-93%, save hardware resources by 3%-25.1% with shell tailoring, and reduce software modifications by 88-107× using command-based interfaces. Importantly, Harmonia maintains the throughput and latency of applications with negligible resource overhead (<0.63%). Compared to commercial frameworks (Vitis and oneAPI) and open-source frameworks (Coyote), Harmonia lowers shell resource consumption by 3.5%-14.9%, supports cross-vendor FPGAs, simplifies 15-23× software configurations, while achieving comparable throughput and latency across multiple benchmarks.

As a summary, Table 1 compares Harmonia with previous frameworks, highlighting their primary differences. Harmonia aims to provide a unified framework that addresses heterogeneity issues for shells, roles, and host software to facilitate heterogeneous FPGA acceleration in the cloud.

This paper makes the following contributions:

- We conduct practical experiments to demonstrate the challenges posed by FPGA heterogeneity in cloud applications, which are only partially addressed in previous work (§2).
- We design a modularized shell composed of novel RBB abstractions to simplify development efforts (§3.3.1), a hierarchical shell tailoring method to provide resource-efficient and easy-to-use shells for roles (§3.3.2), and a command-based interface to ease host software developments and integrations (§3.3.3).
- We have implemented Harmonia and deployed it at cloud-scale to a large service provider with extensive validation (§4). Practical results demonstrate its benefits for heterogeneous FPGA applications (§5).

2 Background & Motivation

2.1 Cloud FPGA Acceleration Overview

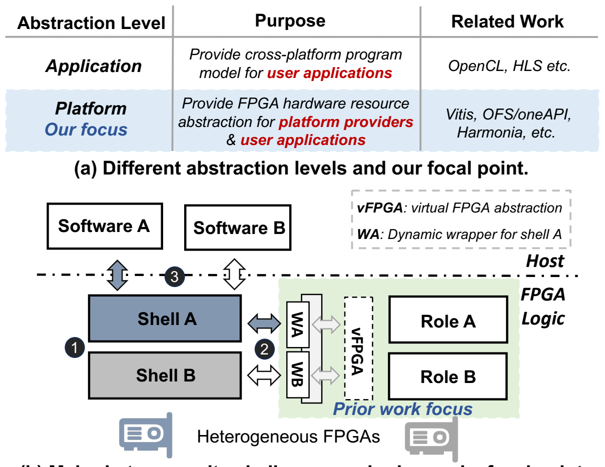
In recent years, cloud providers have incorporated FPGAs into their infrastructures [2, 3, 62] to accelerate applications, such as networking [32, 98], storage [15, 55], and machine learning [80, 102], etc. They typically employ the shell-role architecture [16, 18, 70] for FPGA-accelerated applications (Figure 1), which consists of three main components:

- *Shell.* The shell acts as the FPGA OS to manage FPGA resources and provide common services to improve the usability of cloud FPGAs [70, 103]. A production-grade shell entails multiple functionalities, such as I/O connectivity [71, 81], memory management [48], dynamic configuration [47], health monitoring [70], and more.
- *Role.* The role refers to the accelerated application logic that is partially or fully offloaded onto FPGA devices. Roles typically utilize the common services (*e.g.*, Ethernet [81], DMA [71], etc.) offered by the shell to establish communication with cloud networks and hosts.
- *Host Software.* The host software communicates with the FPGAs for data exchange and control operations. During application deployment, it performs initialization tasks such as table configuration [98] and task enablement [19]. At runtime, it handles data exchange for tasks involving software-hardware co-design [25, 94].

2.2 Increasing Heterogeneity of Cloud FPGAs

Some cloud providers deploy homogeneous FPGA clusters to maintain infrastructure manageability and reduce operation costs [2, 3, 62]. However, heterogeneous FPGAs are increasing in some scenarios due to several practical reasons:

(i) Distinct acceleration architectures. Different applications exhibit unique functions and workloads that demand specific acceleration architectures to achieve desired performance. For example, networking applications [21, 57, 98] require FPGA integration of high-speed network interfaces and network protocol stacks. They deploy FPGAs between network switches and hosts as SmartNICs [32]. In contrast,



Abstraction Level	Purpose	Related Work
Application	*Provide cross-platform program model for* ***user applications***	*OpenCL, HLS etc.*
Platform ***Our focus***	*Provide FPGA hardware resource abstraction for* ***platform providers*** *&* ***user applications***	*Vitis, OFS/oneAPI, Harmonia, etc.*

(a) Different abstraction levels and our focal point.

(b) Major heterogeneity challenges and prior works focal point.

Figure 2. (a). Harmonia focuses on a platform-level hardware resource abstraction. (b). Heterogeneous FPGAs require shell reconstructions (❶) and interface modifications for roles and host software (❷ & ❸), while prior works partially addressing portability challenges for roles (❷).

storage applications [15] desire FPGAs equipped with high-capacity memory and incorporate I/O operators like compression [10], which involve attaching FPGAs directly to SSDs as SmartSSD [50]. Hence, the internal architectures and hardware capabilities can vary significantly across FPGAs.

(ii) Customized FPGA devices. Some cloud providers order customized FPGA devices from different vendors (*e.g.*, Intel[34], Xilinx[5], etc.). On the one hand, these FPGAs are equipped with suitable on-chip resources and off-chip peripherals tailored to the target applications, which helps to reduce costs when deployed at scale. On the other hand, using multi-vendor FPGAs can also provide backup options, enhancing supply chain security. These devices exhibit distinct capabilities and vendor-specific toolkits for users.

(iii) Multiple FPGA generations. In our cloud, the lifecycle of FPGA servers (commodity servers equipped with FPGAs) typically extends for at least four years, while new FPGA devices are typically introduced every one to two years [5]. Consequently, multiple generations of FPGAs can coexist during the evolution of applications. They exhibit differences in IP properties such as interfaces and configurations.

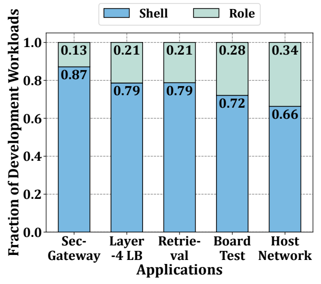
2.3 Prior Works & Limitations

Hardware abstraction is a common method to address heterogeneity. Prior works abstract FPGAs at different levels based on the needs of different developers. As shown in Figure 2a, application-level abstractions aim to streamline user application development by providing a unified cross-platform programming model (*e.g.*, HLS [22], OpenCL [63], etc.). Platform-level abstractions further allows fine-grained management and optimization of hardware resources for providers (*e.g.*, shell-role platforms [66, 70, 86]). We focus on designs at the platform level in this paper.

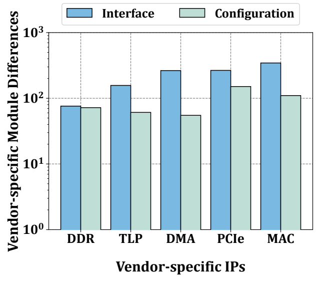
FPGA virtualization [14, 45, 49, 99–101] and portable OSs [44, 47, 59, 79, 103] are proposed to address the heterogeneity challenges, as shown in Figure 2b. FPGA virtualization abstracts the platform-agnostic virtual FPGA to enable portable role development, while the hardware deployment relies on different vendor-specific shells (*e.g.*, Vitis [86], etc.). FPGA OSs employ dynamic wrappers tailored to each shell, enabling roles to connect directly to the shell via a unified interface. The above works address the portability issue for roles; however, they still involve laborious development workloads for the shell and ad-hoc modifications to the host software:

(i) Laborious development workloads for the shell. In practical FPGA acceleration development, we observe three main factors that cause laborious development workloads:

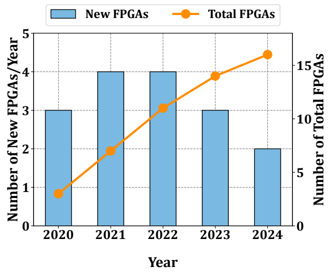
- *First,* the shell demands extensive development and validation efforts to ensure its integrity and stability. Apart from using off-the-shelf IPs directly, the shell also requires massive in-house design and development to cover the functionalities mentioned in §2.1. We compare the development workloads between shells and roles (measured by the ratio of hardware logic codes) in five typical cloud applications (detailed in §5.1). After excluding the script-generated portions that can be automated by vendor tools [35, 88], the handcraft development workloads are represented in Figure 3a. Shells occupy the majority of workloads (66%- 87%). Considering a moderate FPGA project that includes tens of thousands of lines of hardware code and requires several months for verification [19, 97], customized shell development can be time-consuming.
- *Second,* the platform-specific module differences (*e.g.*, interfaces and configurations) make it difficult to directly reuse shell modules. In detail, we analyze module differences (measured by the number of interfaces and configurations) across common I/O modules in shells for different FPGAs (*i.e.*, xilinx [5] and intel [34]). Figure 3b illustrates the massive disparities of properties among common modules, ranging from tens to hundreds. Therefore, providers have to perform intricate modifications instead of simply reusing shell modules across different FPGAs.
- *Finally,* platform providers have to reconstruct shells to support new FPGA devices instead of making a one-time effort. As the number and types of heterogeneous FPGAs increase, the demand for developing new shells keeps growing. Figure 3c illustrates the annual deployment of new FPGA devices and the total number of FPGAs in our cloud. The increasing variety of heterogeneous FPGAs drives us to seek an efficient solution for constructing shells to reduce development workloads.



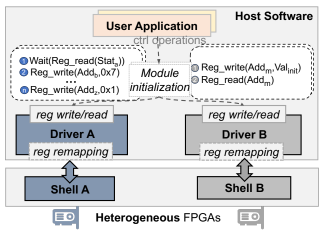
(a) Production-grade shells occupy the majority of FPGA logic development workloads.



(b) Vendor-specific IPs exhibit massive interface and configuration differences across FPGAs.



(c) The total number of heterogeneous FPGAs in our cloud is increasing every year.



(d) The user application requires ad-hoc modifications when migrating into new FPGA platforms.

Figure 3. Heterogeneous FPGAs introduce more laborious development workloads and ad-hoc modifications.

(ii) Ad-hoc modifications to the host software. Host software has to perceive hardware variations and make ad-hoc modifications to ensure correct control. Specifically, commercial framework (*e.g.*, Vitis [86], OFS [66], etc.) provides a register read/write an interface for user applications to perform control operations. However, heterogeneous FPGAs introduce differences in register widths, addresses, and functionalities, requiring software modifications. Some frameworks help mitigate register width and address changes in their drivers by mapping registers into a unified address space. Nevertheless, users still need to modify the sequence of register operations and care about the differences in control logic across platforms. Figure 3d shows an example of module initialization across different FPGA platforms. For shell A, users need to wait for the status register *a* read completion before proceeding with a series of initialization logic. In contrast, Shell B includes automation logic that allows users to directly write the initial values. Thus, user applications must consider both the register values and their operational dependencies, which introduces complex changes.

Thus, we transition towards devising a comprehensive solution to fully address heterogeneity challenges.

3 Harmonia Design

3.1 Framework Overview

We follow the bottom-up design principle [28] that abstracts platform-specific differences from the underlying diverse FPGAs, providing a unified interface for the upper-layer shell, role, and host software. Guided by this principle, we present Harmonia, a unified framework to streamline the development and deployment of heterogeneous FPGA-accelerated applications. Figure 4 illustrates the Harmonia's architecture:

Platform-specific layer. This layer aims to handle disparities among diverse FPGA platforms (§3.2). It comprises two distinct components: the automated *platform adapter* responsible for managing platform-specific configurations, and the lightweight *interface wrapper* to convert vendor-specific interfaces into a uniform format. In summary, this layer acts as a unifying bridge, ensuring seamless migration of upper layers across heterogeneous FPGA platforms.

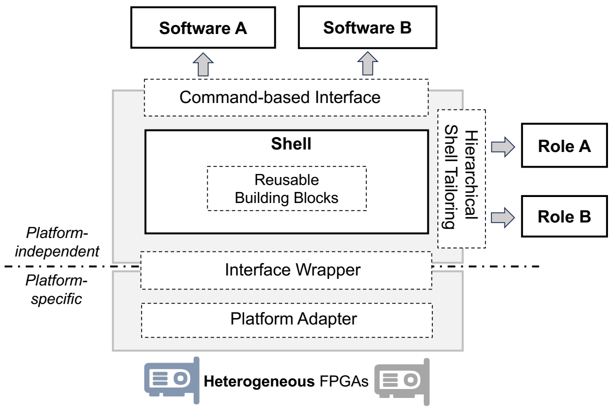


Figure 4. Harmonia architecture. The platform-specific layer manages FPGA heterogeneity and the platform-independent layer provides a unified shell for roles and software.

Platform-independent layer. This layer is independent of specific FPGA platforms, serving the shell, role, and host software (§3.3). Harmonia creates a unified shell abstraction to manage FPGA chips and provide connectivity with peripherals using a series of *Reusable Building Blocks* (RBBs) abstractions (§3.3.1). To simplify interfaces and configurations for roles and reduce unnecessary resource consumption, Harmonia introduces a hierarchical shell tailoring mechanism based on application demands (§3.3.2). To conceal the details of underlying controls for the host software, Harmonia abstracts the register-level register interface into a unified behavior-level command-based interface for hardware-software communications (§3.3.3).

3.2 Platform-specific Layer

Automated platform adapter. Different FPGAs vary in many aspects, including chip families [1, 6], peripherals [71, 81], compilation tools [35, 88], etc. To avoid error-prone manual operations, Harmonia integrates the automated platform adapters to manage platform-specific configurations. These differences are divided into two parts based on their dependencies: resource differences related to FPGA devices and deployment differences related to vendors. As shown in Figure 5, these differences are managed by two sub-adapters:

- **Device adapters** are responsible for hardware resource-related configurations. Harmonia separate resource configurations into two groups: the *static* and *dynamic* group. The former maintains all the inherent resource properties of FPGA chips and peripherals (*e.g.*, channel numbers, virtual functions, etc.), which only need to be configured once and reused anywhere. The latter pertains to dynamic mapping constraints between the logic and the device, such as I/O pins and clock mappings configured on-demand.
- **Vendor adapters** manages the major deployment differences among FPGA vendors like specific IP packaging format [39], compilation CAD tools [35, 88], etc. In practice, compatibility issues between modules and deployment environments commonly arise in heterogeneous FPGA scenarios. For example, the common DMA engine from different vendors [5, 34] may rely on dedicated PCIe hard IPs and corresponding compilers [35, 88]. To avoid manual troubleshooting, Harmonia incorporates the *built-in handler* to structure the vendor dependencies of each module as a series of *key-value* pairs and performs rigid inspections to ensure compatibility during deployment. The *key* defines vendor-specific attributes such as CAD tools, IP catalogs, etc. The values are specified with independent version numbers to simplify dependency checks.

Both adapters are generated using vendor-provided tcl [87] and ruby [74] scripts, enabling easy development.

Lightweight interface wrapper. Developers often utilize third-party IPs to reduce their development efforts. These vendor-specific IPs follow distinct interface protocols (*e.g.*, AXI [4] and Avalon [9]). Consequently, replacing one vendor's IP with another requires modifications to the upper-layer logic. To address this issue, Harmonia develops lightweight interface wrappers that encapsulate different interfaces into a uniform format for the upper-layer logic.

The interface wrapper aims to fulfill the common interface demands of applications while ensuring minimal performance overhead. To achieve this, Harmonia leverages two observations: (i) cloud applications exhibit similar characteristics in data transfer and control; (ii) primary interface protocols have similar interface types. Specifically, cloud applications either transfer continuous data in streaming format or block data with specific addresses and sizes. The

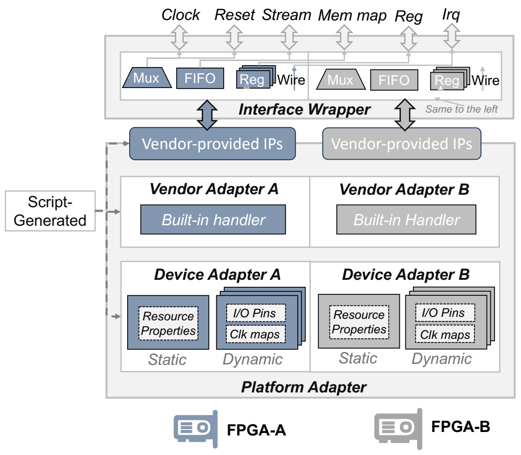


Figure 5. *Platform adapters* manage platform-specific configurations and *interface wrappers* convert vendor-specific interfaces into a uniform format.

control operations are usually carried out by general registers of reading and writing. These data transfer and control are also described in the major interface protocols [4, 9].

Therefore, along with the basic clock and reset signals, Harmonia provides five basic types: *clock*, *reset*, *streaming*, *mem map*, and *reg*. In detail, Harmonia integrates multiple clock and reset signals (*e.g.*, differential clocks, soft reset, etc.) into the clock and reset arrays. Other modules use indices to select specific signals according to their performance needs. For data interfaces, Harmonia provides *streaming* and *mem map* interfaces. The *streaming* interface specifies the start and end of the data stream, and the *mem map* interface defines the address and size of the data chunk. The output data from vendor-specific IPs are stored in FIFO buffers along with the sideband signals (*e.g.*, masks, empty flags, etc.). Harmonia designs fully pipelined sequential translation logic to convert data with varying widths into the unified format (see §3.3.1). It operates without generating bubbles in the processing and consumes a few fixed clock cycle. For control interfaces, Harmonia registers diverse control signals and assigns unique addresses to access them through the register read/write approach. To address latency-intensive signal requirements, Harmonia introduces a special type, *irq*, which exposes raw signals to the upper-level logic.

3.3 Platform-independent Layer

3.3.1 Unified Shell Abstraction. Similar to the host OS [58], the shell should connect applications with FPGA resources. Based on this inspiration, Harmonia creates a unified shell abstraction to manage FPGA chips and provide connectivity with peripherals. As shown in Figure 6, Harmonia abstracts a series of *Reusable Building Blocks* (RBBs) based on FPGA peripherals and chips. Each RBB consists of two parts: the *specific instance* and the *reusable logic*. The *specific*

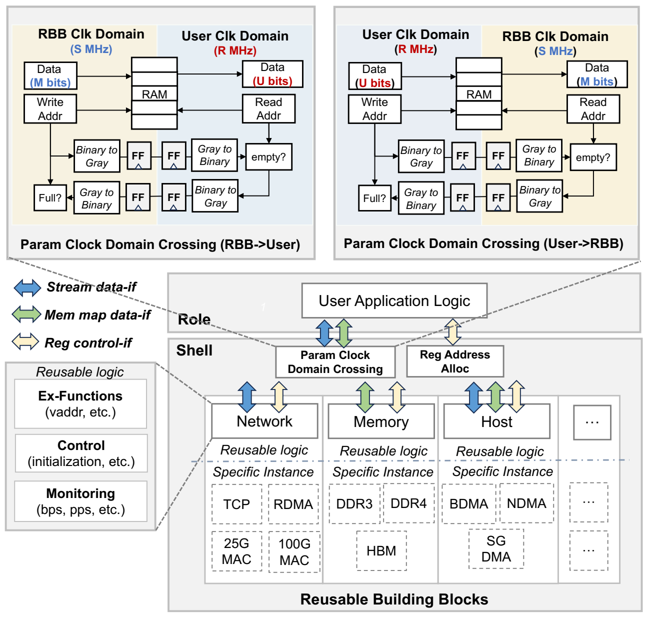


Figure 6. The unified shell abstraction with a set of *Reusable Building Blocks* to operate peripherals and chips.

instance comprises various vendor-specific IPs that provide the basic functionality for connecting with FPGA resources (*e.g.*, MAC[81], PCIe [71], etc.). The *reusable logic* provides the common logic that extends beyond these instances, including the Ex-function logic for performance optimization and feature enhancement, as well as control and monitoring logic for necessary hardware management.

Harmonia provides multiple RBBs for various cloud applications, including *Network* RBB for networking applications [32, 98], *Memory* RBB for storage applications [55, 56], and *Host* RBB for computation applications [97], as shown below:

Network RBB. Network RBB deals with network traffic, including packet-level processing (*e.g.*, MAC [81]) and flow-level processing (*e.g.*, RDMA [91]). To support diverse network scenarios, Harmonia provides both the *packet filter* and *flow director* in its Ex-function part. The *packet filter* intercepts packets with destination addresses that do not belong to the local machine, thereby supporting multicast scenarios [27]. The *flow director* effectively directs incoming flows to their corresponding host queues, ensuring network isolation for multi-tenant environments [90]. Additionally, *Network* RBB monitors the real-time throughput, packet loss, queue usage, and processing rate. It adopts the *stream* interface for data transfer and a 32-bit *reg* interface for control. The main parameter change lies in data-width, which scales (128/512/2048 bits) with network advancements (25/100/400Gbps). Harmonia uses a parameterized clock domain crossing to perform the data-width conversion (see discussions below). Roles can select specific network instances (e.g., 25/100/400G MAC[30]) that fit their demands.

Memory RBB. Memory RBB manages FPGA memory resources such as DDR and HBM. There are common functional demands in these memories. For example, the memory access pattern significantly impacts I/O efficiency [76]. Thus, Harmonia incorporates *address interleaving* and *hot cache* mechanisms in the Ex-function. The *address interleaving* maps data into different bank groups to improve the efficiency of read/write operations. Meanwhile, the *hot cache* stores consecutively accessed data on-chip for fast access, avoiding situations where interleaved access is impossible. These memories transfer data using the memory-mapped approach [61]. Like other frameworks [89, 95], we provide a 512-bit *mem map* interface for data transfer and a 32-bit reg interface for control. The parameter change lies in the channel number, which depends on the underlying devices (2 channels for DDR and 32 channels for HBM). Given the distinct performance of these memories (460GB/s for HBM, 19.2GB/s for DDR), roles should select the appropriate storage instance (HBM/DDR) based on their application demands.

Host RBB. The RBB establishes connections with the host. Cloud FPGAs typically communicate with the host servers via PCIe DMA [19, 32, 98]. To ensure secure communication in host multi-tenancy environments, Harmonia provides *multi-queue isolation* in the Ex-function. The *multi-queue isolation* provides 1K DMA queues to isolate the transmitted data from different tenants. Harmonia maintains an active/inactive state for each queue, and only schedules active queues to improve the scheduling rate. *Host* RBB monitors per-queue information including the queue depth, transmitted packet, and speed. It provides *mem map* and *stream* interfaces for data transfer, along with a 32-bit *reg* interface for control. The key parameter changes are the data width and clock frequency, which double with each PCIe generation upgrade (Gen3/4/5). Harmonia employs parameterized clock domain crossing to handle these conversion (see the below discussions). Roles should select specific PCIe instances [37] that align with their host communication demands.

Discussion of RBB generalizability. RBB categorizes hardware modules that provide similar functionalities. These modules share similarities in data transfer, monitoring, and control functions, while differing in interfaces (*e.g.*, data width, frequency, control signals) and performance (*e.g.*, 25/100G MAC, PCIe Gen3/4/5). Thus, the main purpose of RBB is to maintain the common *function* reusability (*e.g.*, real-time statistics, packet filter, etc.) across modules, allowing roles to select *specific instances* (choose 25 or 100G MAC) that match their application processing *performance* without the necessity to redevelop common functions.

On the data interface, Harmonia integrates the param clock domain crossing to perform frequency and data width conversion. As shown in Figure 6, Roles and RBBs can operate at different clock frequencies and data widths. To synchronize an RBB at S MHz clock and M bits data width with a user application at R MHz clock and U bits data width, Harmonia employs the widely used asynchronous FIFO to perform cross-domain data read and write (design details see [23]). The clock and data width are configurable and do not affect the data processing workflow. Users can select instances that match $S \times M = R \times U$ to achieve lossless bandwidth. On the control interface, Harmonia allocates unique addresses to different module instances and provides the register interface for their controls.

Harmonia employs RBBs to enable common function reuse across multiple FPGA generations. An FPGA generation is characterized by vendors, chip families (process nodes), and device peripherals with the technological advancement. Specifically, Harmonia supports both commercially available FPGAs from Xilinx and Intel, as well as in-house FPGAs. The supported chip families span various process nodes, including Virtex UltraScale+ (XCVU3P, XCVU9P, XCVU23P, XCVU35P at 14/16nm)[84], Virtex UltraScale (XCVU125 at 20nm)[85], Zynq 7000 (28nm)[7], Agilex (5, 7 at 10nm)[1], Stratix 10 (14nm)[38], and Arria 10 (20nm)[36]. For device peripherals, Harmonia provides support for PCIe Gen3x8/x16, Gen4x8/x16, Gen5x8/x16, DDR3/4, HBM, QSFP112/56/28, DSFP and etc. The FPGA generation evolves in parallel with advancements in data center infrastructure. For instance, as network link speeds increase from 25Gbps to 400Gbps and host bandwidths expand from 8GB/s to 32GB/s, FPGAs should be enhanced to keep pace with these advancements.

3.3.2 Hierarchical Shell Tailoring.

The unified abstraction streamlines shell developments for platform providers and provides common services for roles. However, cloud applications have unique acceleration requirements and typically require only a subset of the features [19, 55, 98]. Providing a *one-size-fits-all* shell not only leads to unnecessary hardware resource consumption but also adds configuration complexity for roles. Therefore, Harmonia implements a hierarchical shell tailoring approach that provides a resource-efficient and easy-to-use shell for different roles.

As shown in Figure 7, Harmonia first perform the *module-level* tailoring. It removes non-essential RBBs from the unified shell based on the resource and functional requirements of the role. For the remaining RBBs, Harmonia selects instances that fulfill the performance demands of data transfer. For example, a BDMA instance may be chosen for bulk data transfer, while an SGDMA instance may be chosen for discrete data transfer. Next, Harmonia conduct the *property-level* tailoring to remove properties that are not relevant to the role. Vendor-specific instances usually provide various interfaces and configurations to cover all scenarios, while applications only need to focus on a subset of standardized cloud deployment scenarios [70]. Therefore, Harmonia further tailors the properties of instances into two parts: the shell-oriented part and the role-oriented part, exposing only

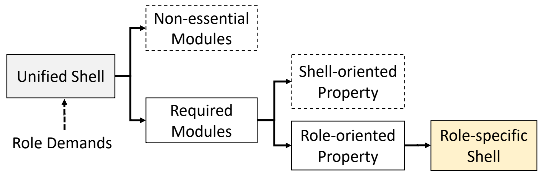


Figure 7. Hierarchical shell tailoring, including both module level and property level.

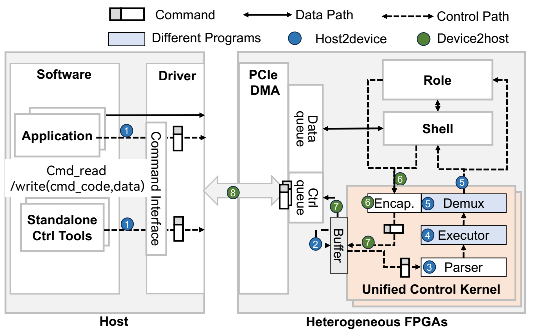


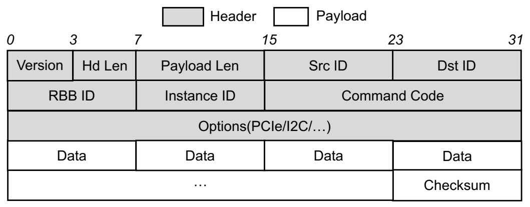
Figure 8. The hardware-software interface re-abstractions and an example walkthrough.

the necessary properties required by each role (*e.g.*, occupied channels, desired queues, etc.)

3.3.3 Command-based Interface.

In FPGA acceleration applications, software and hardware commonly utilize the *register_read* and *register_write* interfaces for flexible controls. While the role-specific register control logic is determined by the application and remains consistent across platforms, the shell-specific register control logic is decided by third-party IPs and varies across different FPGAs (Figure 3b). When using the register interface, applications consider not only the register values but also their operational dependencies (§2.3). However, we observe that while register details change frequently, control operations typically remain consistent (e.g., table read/write, module initialization, etc.). Thus, we provide a series of commands that allow users to issue control operations directly. Figure 8 gives a diagram of interfaces, Harmonia provides *cmd_read* and *cmd_write* as control interfaces for software-hardware communication.

Interface Details. The command should consider both generality and extensibility. First, the commands should be applicable to different FPGA architectures rather than being specific to certain platforms using domain-specific instructions (*e.g.*, using specific ISA [8]). Second, the commands should support the extension to new hardware modules (*e.g.*, i2c) and software (*e.g.*, standalone control tools). To achieve generality, Harmonia adopts the widely used packet format in communication to define the command [13]. As illustrated in Figure 9, the $Version$ records the update of the commands.



Command Code	Description	Command Code	Description
0x0000	Module Status Read	0x0003	Module Reset
0x0001	Module Status Write	0x0004	Table Write
0x0002	Module Initiation	...	...

Figure 9. The command format and common examples.

The following *HdLen* and *PayloadLen* indicate the length of the packet header and payload, respectively, measured in units of four bytes to ensure alignment. The *SrcID* represents the type of host software controllers and the *DstID* represents the underlying hardware modules. Harmonia assigns unique *SrcID* and *DstID* values to differentiate between various software and hardware modules, thereby supporting the extension to new controllers. The second set of four bytes consists of the module operation code, which is divided into three sub-fields. The *ModuleID* indicates the target modules, whereas the *InstanceID* represents the specific module instances. The *CommandCode* specifies the dedicated control operations defined by each RBB for its operational needs. The *Options* describes the information associated with the physical interface used, such as PCIe. For the command payload, users can insert control information into the *Data* field, and the *CheckSum* is provided as an error handling.

Example walkthrough. Harmonia develops software that runs on lightweight software cores within FPGAs (*e.g.*, Nios [64]) as a *unified control kernel* to centralize the command execution. The decision to deploy soft cores in hardware rather than on the host is due to the presence of various controllers on production servers (such as applications, BMC, and standalone tools), which makes unified management more convenient in hardware. Figure 8 provides an example walkthrough using the command-based interface:

1. **Command generation**: host applications or control tools call *cmd_read/write* interface to issue control operations with *cmd_code* and *data*. The *cmd_code* indicates the command for the control operations, while the *data* carries control data. Figure 9 presents some common commands in Harmonia. Users can select commands to initialize or reset hardware and retrieve their status. The driver then generates command packets.
2. **Command transfer**: the driver transfers commands to the hardware via PCIe. Harmonia integrates a separate control queue in the DMA engine to ensure performance isolation from the data path. These commands are sent to the buffer of the unified control kernel (with configurable depth) to await processing.
3. **Command parsing**: the unified control kernel uses *HdLen* and *PayloadLen* to determine command boundaries, extracting the values of each field.
4. **Command execution**: the unified control kernel sequentially executes commands, each of which defines its own processing logic (such as register read/write, flash erase, time count, etc.), identified by a unique *command code*.
5. **Command distribution**: some commands require reading and writing registers in the shell and role modules. The control kernel initiates register read/write requests to the relevant modules based on the target *ModuleID* and *InstanceID* through the *reg* interface.
6. **Command encapsulation**: the read response data (*e.g.*, temperatures, voltages, performance statistics, etc.) are encapsulated in the unified control kernel as command response packets and fed back to the host software.
7. **Command upload**: The response command packet is returned to the software through the same DMA engine and then delivered to the corresponding host software based on the *srcID* specified in the command.

4 Implementation

We have implemented the Harmonia and deployed it to a large service provider for several years. Here, we introduce the lifecycle of cloud FPGA applications with Harmonia:

Stage 1, Requirement Analysis. A feasibility validation is conducted to evaluate the acceleration benefits. Users provide application performance bottlenecks. Next, the hardware designers estimate the benefits of using FPGA acceleration through proof-of-concept (PoC) validation.

Stage 2, Design & Development. FPGA-accelerated applications are divided into two parts: the shell part designed by the platform provider, and the role part along with host software designed by the user:

- **Shell.** Platform providers build platform adapters and interface wrappers for new FPGA devices while creating a unified shell and tailoring it to a role-specific instance. First, providers build device and vendor adapters for new FPGA devices as described in §3.2. Next, the lightweight interface wrappers are added for vendor-specific IPs used in RBBs. These components can effectively manage platform-specific configurations. Therefore, providers can effortlessly create a unified shell by utilizing RBBs from the common library with just on-demand modifications to some logic details. Finally, providers perform hierarchical shell tailoring to generate a role-specific instance (§3.3.2).
- **Role & Software.** The role and software can be developed independently based on the unified abstraction, without concerns about the adaptation to different underlying platforms. Once the development is finished, the role is

connected to the shell to form a complete FPGA logic, and the software is integrated into the project.

- **Project implementation.** Harmonia provides the automated integration toolchains. Firstly, Harmonia loads the vendor adapter and checks the dependencies between modules and environments (§3.2). After ensuring that there are no dependency conflicts, Harmonia completes platform configurations and invokes corresponding CAD tools for compilation (*e.g.*, Vivado [88], Quartus [35]). Finally, the FPGA executable bitstream and software are packaged together into a consolidated project file.

Stage3, Integration Test. This stage aims to evaluate the overall functionality, performance, and stability of the generated project. Testers perform rigorous integration testing to cover every component in the system, ensuring that each part is thoroughly validated before online deployment.

Stage 4, Deployment. The release projects are ultimately deployed in the corresponding application clusters, involving software installation and hardware configuration. During this process, scripts in the platform adapter automate hardware configuration, environmental dependency checks, and hardware initialization based on the deployed FPGAs.

5 Evaluation

In this section, we evaluate Harmonia using a series of benchmarks and diverse real-world cloud applications on heterogeneous FPGAs by addressing the following questions:

- **What are the benefits of Harmonia's component design?** We use micro-benchmarks to show that the lightweight interface wrapper maintains module throughput and latency; the hierarchical shell tailoring lowers 3-25.1% resource usage for shells and simplifies 8.8-19.8× configuration items for roles; the command-based interface reduces 88-107× register modifications for host software.
- **What are the benefits and overheads of Harmonia for cloud FPGA application?** We evaluate a variety of real-world cloud applications built on Harmonia. The results show that Harmonia reduces development workloads by 69%-93% with negligible resource overhead (< 0.63%) and minimal performance impact (< 1%).
- **How does Harmonia compare against other frameworks?** We compare Harmonia with representative frameworks (*i.e.*, Vitis, oneAPI, and Coyote). The results show that Harmonia reduces shell resource usage by 3.5%-14.9%, supports cross-vendor FPGAs, and simplifies software configurations by 15-23×, while maintaining comparable throughput and latency across a range of benchmarks.

5.1 Experimental Setup

Applications. We select five real-world FPGA-accelerated applications with different acceleration architectures and functions. As shown in Table 2, the Sec-Gateway deploys the FPGAs at the cloud network boundary to prevent cross-network malicious traffic [19, 105]. FPGAs filter out specific traffic based on the deployed policies. The Layer-4 LB provides layer-4 stateful load-balancing services for public applications [29, 68]. FPGAs work as SmartNICs to distribute incoming flows to many real servers. The Host Networking offload network functions (*e.g.*, Checksum [20], OVS [67], etc.) into FPGAs. The Retrieval chooses relevant candidates from a large corpus for recommendation systems [54] and FPGAs accelerate the similarity calculation and top-K selection. The Board Test serves infrastructure services to test the performance of custom FPGA boards.

Table 2. Selected FPGA-accelerated applications and heterogeneous FPGA cards.

Application	Architecture	Type	Function
Sec-Gateway	BITW	Security	DCI access control
Layer-4 LB	BITW	Network	Layer-4 load balancer
Host Network	BITW	Network	Network offloading
Retrieval	Look-aside	Computation	Embedding retrieval
Board Test	△	Infrastructure	Custom board testing

FPGA	Vendor	Chip	Peripheral
Device A	Xilinx	XCVU35P	HBM,DDR,QSFP×2, PCIe Gen4×8
Device B	Internal	XCVU9P	DDR×2, QSFP×2, PCIe Gen3×16
Device C	Internal	AGILEX®7	DSFP×2, PCIe Gen4×16
Device D	Intel	AGILEX®7	QSFP×2, PCIe Gen4×16, DDR

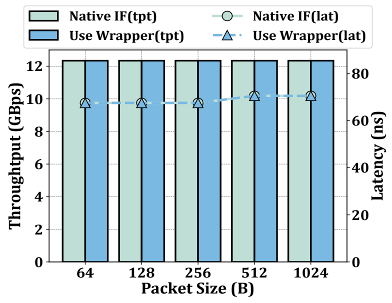
BITW: Bump-in-the-wire architecture.
△ indicates that it supports diverse architectures.

FPGAs. To comprehensively evaluate FPGA heterogeneity, we select four typical FPGA devices that have widespread deployment in our cloud with distinct vendors, chip families, and peripherals, as illustrated in Table 2.

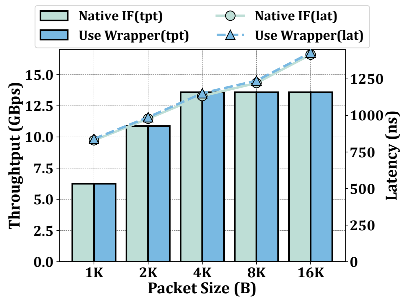
RBBs. We select three common RBBs in cloud applications including *Network*, *Memory* and *Host* as described in §3.3.1 to evaluate Harmonia at the module level.

Frameworks. We compare Harmonia with frameworks that provide platform-level abstractions (see §2.3), including commercial frameworks (Vitis [86] and oneAPI [65]) and open-source frameworks (Coyote [47]).

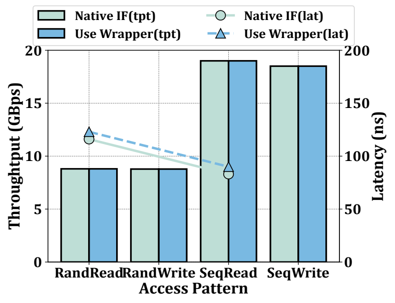
Benchmarks. The framework benchmarks cover three important categories for cloud FPGA applications, including computation, storage and networking. *Matrix multiplication* [60] represents a typical compute-intensive task. we perform single-precision floating-point matrix calculations for matrices sized 64 × 64 across 1024 iterations, measuring the number of matrix calculations per second. *Database access* [24] is a storage-intensive task. we deploy a vector database on external memory and sequentially, fixedly, and randomly read and write 32-bit vectors to measure the number of vectors processed per second. *TCP transmission* [17] is a communication-intensive task. We deploy FPGAs on two



(a) The performance of MAC Module.



(b) The performance of PCIe Module.



(c) The performance of DDR Module.

Figure 10. Throughput and latency comparisons of native modules vs. using the interface wrapper.

servers and connect them via the device network interfaces. The FPGAs directly forward the host's TCP traffic, measuring end-to-end throughput and latency with varying packet sizes.

5.2 Micro-benchmarks

Harmonia's interface wrapper can maintain throughput and latency. Harmonia provides a lightweight interface wrapper to encapsulate the interface differences of vendor-specific IPs. To evaluate its performance impact, we choose three common vendor-specific IPs (MAC [81], PCIe DMA [71], DDR4 [26]). We compare their native performance (using Avalon/AXI) with the performance after interface wrapping. To measure MAC performance, we use loopback tests by directly connecting the RX and TX QSFP ports. For PCIe DMA, we post PCIe read requests of different sizes on the host software. Moreover, we perform both random and sequential read and write operations with fixed-size data to the DDR. Figure 10a-10c shows the experimental results. Firstly, Harmonia maintains native throughput for these modules. This is because Harmonia employs a pipelined processing logic on datapaths (see §3.3.3), ensuring that no bubbles are introduced in the processing, thus preventing throughput degradation. Secondly, Harmonia introduces minimal latency in PCIe DMA and DDR modules (only a few cycles). This is due to the necessary timing optimization for multi-channel data width conversion that can support higher clock frequencies for users [78]. Generally, the nanosecond-level delays are negligible relative to the application end-to-end microsecond-level delay.

Harmonia's shell tailoring can reduce shell resource consumption and simplify role configurations. The shell tailoring provides resource-efficient and easy-to-use shells for different roles. First, we compare the resource consumption of the unified shell design with the shells tailored to applications. We deploy applications on device A and their respective resource consumption is depicted in Figure 11. The unified shell consumes more resources as it incorporates all peripheral connectivity and FPGA management functions. In contrast, the tailored application shells achieve a resource consumption reduction of between 3% and 25.1%. Furthermore, we analyze the configuration items posed by the native modules compared to those actually required by roles, as shown in Figure 12. Harmonia reduces the number of role configurations by 8.8-19.8×. This is mainly because vendors usually provide various configurations to cover all scenarios, while applications only need to focus on a subset of configurations for cloud scenarios.

Command-based interface reduces ad-hoc software modifications. The command-based interface reduces the register-level modifications for host software. We evaluate the changes made to the Host Network software for initializing all hardware modules while transitioning from device C to device D. We compare the register interface used in commercial frameworks with the command-based interface in Harmonia. Figure 13 shows the experimental results. Harmonia reduces software modifications by 88-107× across applications. This reduction is mainly because Harmonia provides a command to abstract control operations and leverages the unified control kernel to execute them, thereby avoiding complex register modifications.

5.3 Benefits & Overheads

Harmonia aims to minimize development workloads for platform providers while ensuring native application performance with negligible resource overhead.

Harmonia reduces development workloads for shell. We measure the development workloads of the shell across different platform configurations, including across chip families (devices A & B) and vendors (devices A & C). To ensure a fair comparison, we adopt the same programming language and developer, measuring the relative proportions of manually developed versus reusable hardware logic code, following the metrics in previous studies [53, 69, 96]. Figure 14 illustrates the test results. Harmonia achieves RBB reuse rates from 69% to 76% for cross-vendor configurations and from

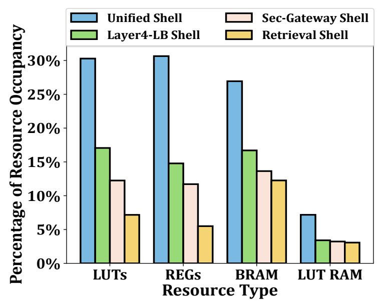


Figure 11. Shell tailoring reduces resource consumption.

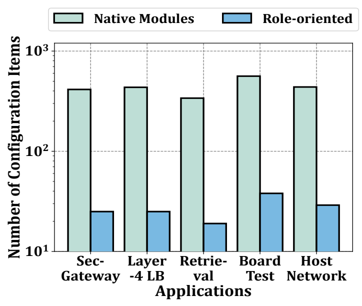


Figure 12. Shell tailoring reduces module configurations for roles.

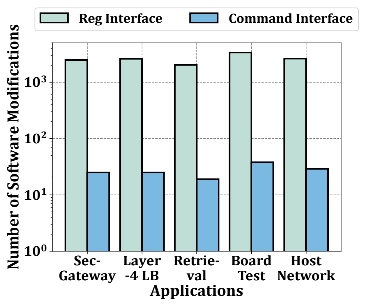


Figure 13. Command-based interfaces reduce software changes.

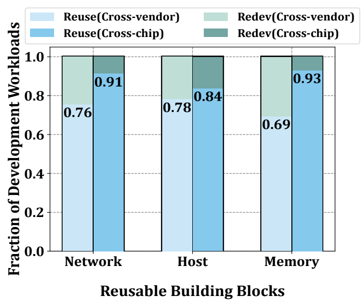


Figure 14. Harmonia reduces development workloads of RBBs across different vendors and chips.

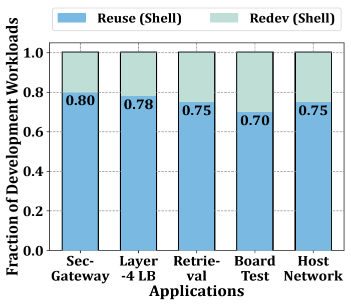


Figure 15. Harmonia reduces development workloads of applications across different FPGAs.

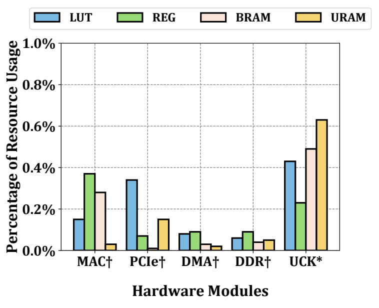


Figure 16. The resource overhead of the interface wrappers† and the unified control kernel*.

84% to 93% for cross-family configurations. This variation occurs because modules from the same vendor typically share design similarities, whereas cross-vendor modules have notable differences. To address this heterogeneity, Harmonia offers automated platform adapters and lightweight interface wrappers to manage hardware configurations. Moreover, Harmonia proposes RBBs that can abstract similar functionalities from vendor-specific IPs to reduce development efforts. The redevelopment portions are located at the control and monitor logic, as their implementation often depends on hardware details. Applications deployed on different platforms also exhibit similar results, showing 70% to 80% shell reuse across applications (Figure 15). Compared to building individual shells from scratch, Harmonia can significantly reduce development workloads for platform providers.

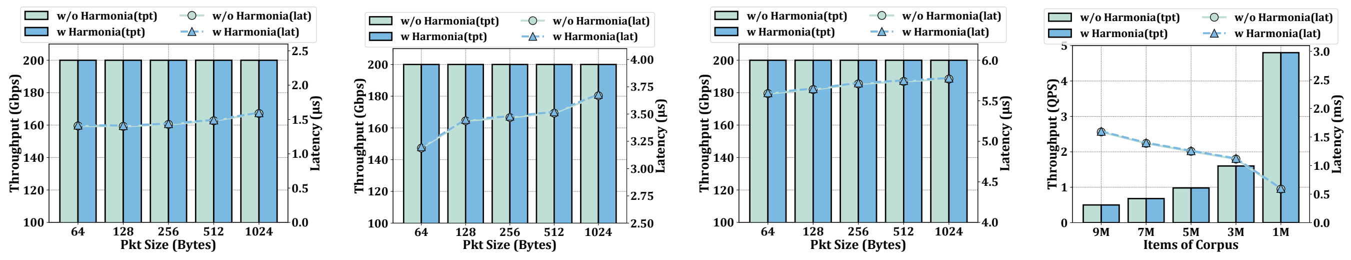
Harmonia introduces minimal resource overheads. To hide hardware disparities, Harmonia incorporates interface wrappers and unified control kernels in hardware. We collect resource usage percentages for those additional hardware components on different FPGA platforms. Figure 16 displays the highest resource consumption percentages. Specifically, Harmonia consumes less than 0.37% of resources in the interface wrapper and less than 0.67% of resources in the unified control kernel, which indicates negligible consumption compared to the functional modules.

Harmonia maintains the throughput and latency of FPGA applications. We compare the performance of applications using Harmonia with those that do not, based on real workloads in their deployment scenarios. The experimental results are presented in Figure 17. Firstly, Harmonia achieves full bandwidth for applications based on the BITW architecture and the desired QPS for applications using the Lookaside architecture. This is because Harmonia employs full pipeline designs to ensure lossless data bandwidth. Second, Harmonia achieves near-native latency with an increase of less than 1%. The slight increase in latency mainly comes from the data buffering in the interface wrapper. However, the added delay at the nanosecond level is negligible compared to the microsecond-level latency in cloud applications.

5.4 Harmonia vs. Other Frameworks

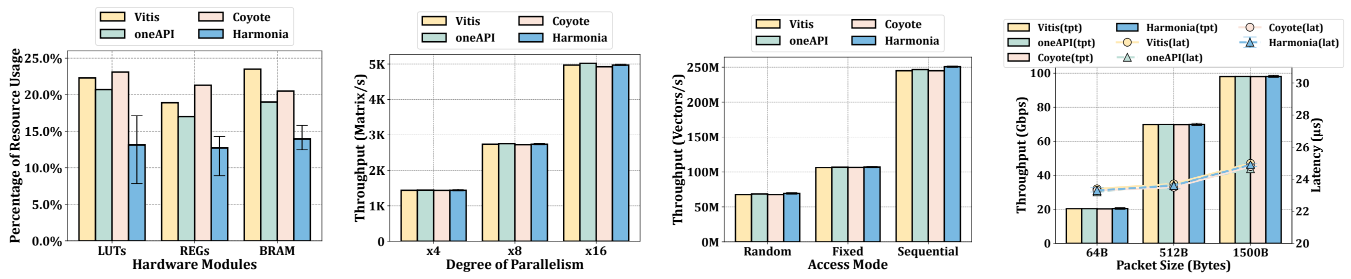
We compare Harmonia with commercial frameworks (Vitis [86] and oneAPI [65]) and open-source frameworks (Coyote [47]) that target platform-level abstractions:

Harmonia reduces shell resource consumption. We deploy frameworks on their respective supported FPGAs. Specifically, Vitis and Coyote support Device A, oneAPI supports Device D, and Harmonia provides support across Devices A to D. We measure their percentage of shell resource



(a) Sec-Gateway Performance. **(b)** Layer-4 Lb Performance. **(c)** Host Network Performance. **(d)** Retrieval Performance.

Figure 17. The throughput and latency comparison with and without using Harmonia across a range of applications.



(a) Framework Resource Usage. **(b)** *Matrix Multiplication* Perf. **(c)** *Database Access* Perf. **(d)** *TCP Transmission* Perf.

Figure 18. The comparison of shell resource usage and benchmark performance of Harmonia with other frameworks.

usage in different benchmarks. As shown in Figure 18a, Harmonia reduces the shell resource consumption from 3.5% to 14.9%. This is due to the fine-grained shell tailoring that removes non-essential modules and functions based on application demands. This optimization not only provides more resources for roles to implement more complex functions but also helps reduce dynamic power consumption.

Harmonia supports a wider variety of FPGAs. We compare the FPGA devices supported by each framework, as shown in Table 3. Coyote is compatible with Xilinx Alveo FPGAs, while Vitis supports a wider range of Xilinx FPGAs, including Alveo/Zynq/Versal, etc. OneAPI currently supports Intel FPGAs such as Agilex and Stratix. However, cloud providers may select FPGAs from multiple vendors or deploy custom devices (§2.2). We find that these frameworks employ a monolithic design for new devices integration, requiring shell redesign. To address this, Harmonia introduces fine-grained RBBs and integrates platform adapters and interface wrappers to support cross-vendor compatibility.

Harmonia simplifies software configurations across different platforms. When deploying benchmarks on different platforms, users need to modify the software control logic to properly configure different hardware modules. We analyze three typical configurations: monitoring statistics, network initialization, and host interaction configuration. We compare these configurations using traditional register interfaces and command-based interfaces. The results are shown in Table 4. Harmonia abstracts control operations into a series of commands executed by the unified control kernel in the hardware. This approach simplifies software configuration by 15-23×, helping to avoid frequent software modifications when switching platforms.

Table 3. FPGA devices supported by each framework.

Device	Vitis	oneAPI	Coyote	Harmonia
Intel FPGAs	✗	✓	✗	✓
Xilinx FPGAs	✓	✗	✓	✓
In-house (Custom) FPGAs	✗	✗	✗	✓

Table 4. The command-based interface simplifies hardware module configuration for host software.

Interface	Host Interaction Config	Monitoring Statistics	Network Initialization
Registers	84	115	60
Commands	4	5	4

Harmonia achieves comparable throughput and latency to other frameworks. We compare the benchmark performance across different frameworks (see §5.1 for the settings), as shown in Figure 18b-18d. For *matrix multiplication*, the speed of matrix calculations improves with increased parallelism through loop unrolling and using more DSPs. Harmonia achieves consistent throughput relative to other frameworks, as it does not introduce additional overhead in computational units (*e.g.*, LUT, DSP). On *database access*, sequential access exhibits higher throughput than other access patterns, and Harmonia achieves comparable vector read-/write speed. This is due to Harmonia ensuring there are no bubbles in the memory RBB interface wrappers that guarantee lossless bandwidth. Regarding *TCP communications*, both throughput and latency increase as packet size grows. Harmonia performs comparably to other frameworks. As analyzed in §5.3, Harmonia provides native throughput while the introduced nanosecond-level latency remains negligible when compared to microsecond-level network latency.

6 Discussion

Multi-tenancy with Harmonia. Harmonia supports multi-tenancy to enhance the resource utilization of cloud FPGAs. Specifically, Harmonia utilizes the Ex-function in RBBs to achieve resource isolation in the shell (§3.3.1), while employing typical partial reconfiguration techniques [44, 47] to enable multi-tenancy deployment in the role. Moreover, Harmonia provides multiple independent queues to isolate host software belonging to different users.

Benefits for External Users. Harmonia streamlines application deployment on heterogeneous FPGAs for various internal applications. While providing individual shells for external users is impractical, they can still benefit from Harmonia in the following ways: (i) they can deploy different applications on diverse FPGAs supported by Harmonia. (ii) it is possible to provide module-level tailoring using a few config interfaces for users. (iii) the command-based interface simplifies their software development.

7 Related work

FPGA virtualization. FPGA virtualization abstracts virtual FPGAs to enable resource sharing [72, 83]. The overlay architecture [12, 40, 43, 46, 73] compiles FPGA logic into an intermediate representation (IR) and then maps it on a specific FPGA platform. It supports runtime compilation but decreases resource utilization and performance. To address these problems, the slot-based method [14, 18, 45, 93, 99, 100] divides the FPGA into finer-grained slots . ViTAL [99] proposes virtual blocks to enable fine-grained FPGA sharing, and Hetero-ViTAL [100] further supports heterogeneous FPGAs. Some work [49, 101] can enhance the productivity. These works address role's portability issues but require laborious shell developments and ad-hoc modifications to software.

FPGA OS abstractions. FPGA operating system (OS) manages FPGA resources and provides basic services for the upper-layer applications [44, 47, 59, 79, 103]. Some works provide OS semantics for FPGA platforms, such as virtual memory [47, 48], time-sharing [31, 52], relocation [42], task scheduling [33, 47, 92] and context switching [42, 51, 75], Another studies provide complete OSs[44, 47]. However, the construction of the OS (shell) still requires laborious work.

Commercial Frameworks. There are many commercial frameworks that abstract FPGAs at different levels tailored to different developers. OpenCL [63] and HLS [22] provide cross-platform programming abstraction to simplify user logic development. Catapult [70] and AWS F1 [3] provide homogeneous shell-role platforms while oneAPI[65]/OFS[66] and Vitis [86] support more FPGAs. Compared with them, Harmonia further propose a more efficient shell construction approach and introduce new command-based interfaces to streamline software development.

8 Conclusion

FPGAs are attractive for cloud applications to boost their performance. The increasing heterogeneity of FPGAs presents new challenges for applications. We present Harmonia, a unified framework that abstracts heterogeneous FPGAs and streamlines the development and configuration of shell, role, and software. Harmonia's successful deployment in a large service provider reduces development workloads with minimal performance impact and negligible resource overhead. This approach holds promise in enhancing FPGA accessibility and efficiency in diverse cloud environments.

Acknowledgments

We thank our shepherd Andrew Putnam and all the anonymous reviewers for their insightful feedback. This work is supported in part by National Key R&D Program of China (Grant No. 2022YFB2901300) and in part by the National Natural Science Foundation of China (Grant No. 62202447). Heng Pan and Gaogang Xie are the corresponding authors.

References

[1] Agilex™ FPGA Portfolio. https://www.intel.com/content/www/us/en/products/details/fpga/agilex.html.
[2] Aliyun. Aliyun ECS F3 Instances. https://www.alibabacloud.com/help/doc-detail/108504.html.
[3] Amazon EC2 F1 Instances. https://aws.amazon.com/es/ec2/instance-types/f1/.
[4] AMBA AXI4 Interface Protocol. https://www.xilinx.com/products/intellectual-property/axi.html.
[5] AMD Alveo™ Adaptable Accelerator Cards. https://www.xilinx.com/products/boards-and-kits/alveo.html.
[6] AMD Virtex™ UltraScale+™ VU19P FPGAs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus-vu19p.html#portfolio.
[7] AMD Zynq™ 7000 SoCs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html.
[8] Krste Asanović and David A Patterson. Instruction sets should be free: The case for risc-v. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.
[9] Avalon® Interface Specifications. https://www.intel.com/content/www/us/en/docs/programmable/683091/20-1/introduction-to-the-interface-specifications.html.

[10] Matěj Bartík, Sven Ubik, and Pavel Kubalik. Lz4 compression algorithm on fpga. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 179–182. IEEE, 2015.

[11] Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, et al. The future of fpga acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(3):1–42, 2022.

[12] Alexander Brant and Guy GF Lemieux. Zuma: An open fpga overlay architecture. In *2012 IEEE 20th international symposium on field-programmable custom computing machines*, pages 93–96. IEEE, 2012.

[13] Ravi Budruk, Don Anderson, and Tom Shanley. *PCI express system architecture*. Addison-Wesley Professional, 2004.

[14] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 109–116. IEEE, 2014.

[15] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, et al. POLARDB meets computational storage: Efficiently support analytical workloads in Cloud-Native relational database. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 29–41, 2020.

[16] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[17] Vinton Cerf and Robert Kahn. A protocol for packet network intercommunication. *IEEE Transactions on communications*, 22(5):637–648, 1974.

[18] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, pages 1–10, 2014.

[19] Jian Chen, Xiaoyu Zhang, Tao Wang, Ying Zhang, Tao Chen, Jiajun Chen, Mingxu Xie, and Qiang Liu. Fidas: Fortifying the cloud via comprehensive fpga-based offloading for intrusion detection: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 1029–1041, 2022.

[20] Hsin-Chieh Chiang, Yuan-Pang Dai, and Chuei-Yu Wang. Full hardware based tcp/ip traffic offload engine (toe) device and the method thereof, January 12 2010. US Patent 7,647,416.

[21] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. λ-nic: Interactive serverless compute on programmable smartnics. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 67–77. IEEE, 2020.

[22] Philippe Coussy and Adam Morawiec. *High-level synthesis*, volume 1. Springer, 2010.

[23] Clifford E Cummings. Simulation and synthesis techniques for asynchronous fifo design. In *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, volume 281. Citeseer, 2002.

[24] Database Access. https://github.com/Xilinx/Vitis_Libraries/tree/main/database.

[25] Shail Dave, Tony Nowatzki, and Aviral Shrivastava. Explainable-dse: An agile and explainable exploration of efficient hw/sw codesigns of deep learning accelerators using bottleneck analysis. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, pages 87–107, 2023.

[26] DDR4 Controller. https://www.xilinx.com/products/intellectual-property/ddr4.html.

[27] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the ip multicast service and architecture. *IEEE network*, 14(1):78–88, 2000.

[28] Giuliano Donzellini and Domenico Ponta. A bottom-up approach to digital design with fpga. In *2011 IEEE International Conference on Microelectronic Systems Education*, pages 31–34. IEEE, 2011.

[29] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *Nsdi*, volume 16, pages 523–535, 2016.

[30] Ethernet SubSystems 10G/25G/40G/50G/100G/200G/400G. https://adaptivesupport.amd.com/s/article/71820?language=en_US.

[31] Suhaib A Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized fpga accelerators for efficient cloud computing. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 430–435. IEEE, 2015.

[32] 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.

[33] Wenyin Fu and Katherine Compton. Scheduling intervals for reconfigurable computing. In *2008 16th International Symposium on Field-Programmable Custom Computing Machines*, pages 87–96. IEEE, 2008.

[34] Intel. Intel® FPGAs and Programmable Solutions. https://www.intel.com/content/www/us/en/products/programmable.html.

[35] Intel Quartus Prime. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html.

[36] Intel® Arria® 10 FPGA and SoC FPGA. https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html.

[37] Intel® FPGA IP for PCIe. https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/interface-protocols/pci-express-protocol.html.

[38] Intel® Stratix® Series FPGAs and SoCs. https://www.intel.com/content/www/us/en/products/details/fpga/stratix.html, 2013.

[39] IP-XACT. https://www.accellera.org/downloads/standards/ip-xact.

[40] Abhishek Kumar Jain, Suhaib A Fahmy, and Douglas L Maskell. Efficient overlay architecture based on dsp blocks. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 25–28. IEEE, 2015.

[41] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2017.

[42] Heiko Kalte and Mario Porrmann. Context saving and restoring for multitasking in reconfigurable systems. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 223–228. IEEE, 2005.

[43] Nachiket Kapre and Jan Gray. Hoplite: Building austere overlay nocs for fpgas. In *2015 25th international conference on field programmable logic and applications (FPL)*, pages 1–8. IEEE, 2015.

[44] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 107–127, 2018.

[45] Oliver Knodel and Rainer G Spallek. Rc3e: provision and management of reconfigurable hardware accelerators in a cloud environment. *arXiv preprint arXiv:1508.06843*, 2015.

[46] Dirk Koch, Christian Beckhoff, and Guy GF Lemieux. An efficient fpga overlay for portable custom instruction set extensions. In *2013 23rd international conference on field programmable logic and applications*, pages 1–8. IEEE, 2013.

[47] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. Do OS abstractions make sense on FPGAs? In *14th USENIX Symposium on Operating*

Systems Design and Implementation (OSDI 20), pages 991–1010, 2020.

[48] Joshua Landgraf, Matthew Giordano, Esther Yoon, and Christopher J Rossbach. Reconfigurable virtual memory for fpga-driven i/o. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 556–571, 2023.

[49] Joshua Landgraf, Tiffany Yang, Will Lin, Christopher J Rossbach, and Eric Schkufza. Compiler-driven fpga virtualization with synergy. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 818–831, 2021.

[50] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. Smartssd: Fpga accelerated near-storage data analytics on ssd. *IEEE Computer architecture letters*, 19(2):110–113, 2020.

[51] Trong-Yen Lee, Che-Cheng Hu, Li-Wen Lai, and Chia-Chun Tsai. Hardware context-switch methodology for dynamically partially reconfigurable systems. *Journal of Information Science and Engineering*, 26(4):1289–1305, 2010.

[52] Lorne Levinson, R Manner, Matthias Sessler, and Harald Simmler. Preemptive multitasking on fpgas. In *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No. PR00871)*, pages 301–302. IEEE, 2000.

[53] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14, 2016.

[54] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. Embedding-based product retrieval in taobao search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3181–3189, 2021.

[55] Sheng Li, Hyeontaek Lim, Victor W Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David G Andersen, O Seongil, Sukhan Lee, and Pradeep Dubey. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 476–488, 2015.

[56] Shuangchen Li, Dimin Niu, Yuhao Wang, Wei Han, Zhe Zhang, Tianchan Guan, Yijin Guan, Heng Liu, Linyong Huang, Zhaoyang Du, et al. Hyperscale fpga-as-a-service architecture for large-scale distributed graph neural network. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 946–961, 2022.

[57] Jiaxin Lin, Kiran Patel, Brent E Stephens, Anirudh Sivaraman, and Aditya Akella. Panic: A high-performance programmable nic for multi-tenant networks. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 243–259, 2020.

[58] Linux Kernel. https://www.kernel.org/.

[59] Enno Lübbers and Marco Platzner. Reconos: Multithreaded programming for reconfigurable computers. *ACM Transactions on Embedded Computing Systems (TECS)*, 9(1):1–33, 2009.

[60] Matrix Multiply. https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/C++SYCL_FPGA/ReferenceDesigns/matmul.

[61] Memory-Mapped Interfaces. https://www.intel.com/content/www/us/en/docs/programmable/683364/18-1/memory-mapped-interfaces.html.

[62] Microsoft Azure. https://azure.microsoft.com/.

[63] Aaftab Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

[64] Nios® Soft Processor Ser. https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor.html.

[65] oneAPI: A New Era of Heterogeneous Computing. https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html, 2024.

[66] Open FPGA Stack Overview. https://ofs.github.io/ofs-2024.2-1/, 2024.

[67] Heng Pan, Peng He, Zhenyu Li, Pan Zhang, Junjie Wan, Yuhao Zhou, XiongChun Duan, Yu Zhang, and Gaogang Xie. Hoda: a high-performance open vswitch dataplane with multiple specialized data paths. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 82–98, 2024.

[68] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. Ananta: Cloud scale load balancing. *ACM SIGCOMM Computer Communication Review*, 43(4):207–218, 2013.

[69] Oscar E Perez-Cham, Carlos Soubervielle-Montalvo, Alberto S Nunez-Varela, Cesar Puente, and Luis J Ontanon-Garcia. Source code metrics to predict the properties of fpga/vhdl-based synthesized products. In *2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 93–98. IEEE, 2018.

[70] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 13–24. IEEE, 2014.

[71] QDMA Subsystem for PCI Express. https://www.xilinx.com/products/intellectual-property/pcie-qdma.html.

[72] Masudul Hassan Quraishi, Erfan Bank Tavakoli, and Fengbo Ren. A survey of system architectures and techniques for fpga virtualization. *IEEE Transactions on Parallel and Distributed Systems*, 32(9):2216–2230, 2021.

[73] Rafat Rashid, J Gregory Steffan, and Vaughn Betz. Comparing performance, productivity and scalability of the tilt overlay processor to opencl hls. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 20–27. IEEE, 2014.

[74] Ruby, A PROGRAMMER'S BEST FRIEND. https://www.ruby-lang.org/en/documentation/, 2024.

[75] Kyle Rupnow, Wenyin Fu, and Katherine Compton. Block, drop or roll (back): Alternative preemption methods for rh multi-tasking. In *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pages 63–70. IEEE, 2009.

[76] Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, and Lee-Sup Kim. Bank-group level parallelism. *IEEE Transactions on Computers*, 66(8):1428–1434, 2017.

[77] David Sidler, Zsolt István, Muhsen Owaida, Kaan Kara, and Gustavo Alonso. doppiodb: A hardware accelerated database. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1659–1662, 2017.

[78] Kanwar Jit Singh, Albert R Wang, Robert K Brayton, and Alberto Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *1988 IEEE International Conference on Computer-Aided Design*, pages 282–283. IEEE Computer Society, 1988.

[79] Hayden Kwok-Hay So. *Borph: An operating system for fpga-based reconfigurable computers*. University of California, Berkeley, 2007.

[80] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[81] UltraScale+ Integrated 100G Ethernet. https://www.xilinx.com/products/intellectual-property/cmac_usplus.html.

[82] Kohei Umeta, Taketsugu Sawamura, Yuki Shiroishi, and Hideyuki Nasu. Characterization of qsfp and osfp cpo els modules employing an 8-channel cwdm tosa in practical air-cooling conditions. In *2024 IEEE 74th Electronic Components and Technology Conference (ECTC)*, pages 112–117. IEEE, 2024.

[83] Anuj Vaishnav, Khoa Dang Pham, and Dirk Koch. A survey on fpga virtualization. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 131–1317. IEEE, 2018.

[84] Virtex™ UltraScale+™ FPGAs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus.html.

[85] Virtex™ UltraScale™ FPGAs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale.html.

[86] Vitis Unified Software Platform. https://www.xilinx.com/products/design-tools/vitis.html.

[87] Vivado Design Suite Tcl Command Reference Guide. https://docs.amd.com/r/en-US/ug835-vivado-tcl-commands, 2024.

[88] Vivado Design Suite Tutorial. https://www.xilinx.com/developer/products/vivado.html.

[89] Vivado Design Suite Tutorial. https://www.intel.com/content/www/us/en/docs/programmable/683091/22-3/introduction-to-memory-mapped-interfaces.html.

[90] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. Isolation mechanisms for High-Speed Packet-Processing pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1289–1305, 2022.

[91] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. SRNIC: A scalable architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, 2023.

[92] Guy Wassi, Mohamed El Amine Benkhelifa, Geoff Lawday, Francois Verdier, and Samuel Garcia. Multi-shape tasks scheduling for online multitasking on fpgas. In *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–7. IEEE, 2014.

[93] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. Enabling fpgas in hyperscale data centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1078–1086. IEEE, 2015.

[94] Yuhong Wen, Xiaogang Zhao, You Zhou, Tong Zhang, Shangjun Yang, Changsheng Xie, and Fei Wu. Eliminating storage management overhead of deduplication over ssd arrays through a hardware/software co-design. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 320–335, 2024.

[95] Memory Mapped Interfaces. https://docs.amd.com/r/en-US/ug1399-vitis-hls/Memory-Mapped-Interfaces.

[96] Wenquan Xu, Zijian Zhang, Yong Feng, Haoyu Song, Zhikang Chen, Wenfei Wu, Guyue Liu, Yinchao Zhang, Shuxin Liu, Zerui Tian, et al. Clickinc: In-network computing as a service in heterogeneous programmable data-center networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 798–815, 2023.

[97] Chaoliang Zeng, Layong Luo, Qingsong Ning, Yaodong Han, Yuhang Jiang, Ding Tang, Zilong Wang, Kai Chen, and Chuanxiong Guo. FAERY: An FPGA-accelerated embedding-based retrieval system. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 841–856, 2022.

[98] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1345–1358, 2022.

[99] Yue Zha and Jing Li. Virtualizing fpgas in the cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 845–858, 2020.

[100] Yue Zha and Jing Li. Hetero-vital: A virtualization stack for heterogeneous fpga clusters. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 470–483. IEEE, 2021.

[101] Yue Zha and Jing Li. When application-specific isa meets fpgas: a multi-layer virtualization framework for heterogeneous cloud fpgas. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 123–134, 2021.

[102] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.

[103] Jiansong Zhang, Yongqiang Xiong, Ningyi Xu, Ran Shu, Bojie Li, Peng Cheng, Guo Chen, and Thomas Moscibroda. The feniks fpga operating system for cloud computing. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*, pages 1–7, 2017.

[104] Teng Zhang, Jianying Wang, Xuntao Cheng, Hao Xu, Nanlong Yu, Gui Huang, Tieying Zhang, Dengcheng He, Feifei Li, Wei Cao, et al. FPGA-Accelerated compactions for LSM-basedKey-Value store. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 225–237, 2020.

[105] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1083–1100, 2020.