RosebudFlex: Enhancing Performance, Utilization, and Customizability for FPGA-Accelerated Network Function Offloading in Multi-Tenant Environments

Xin Dong[a,b], Lizhuang Tan[a,b], Huiling Shi[a,b], Wei Zhang[a,b], Peiying Zhang[a,c], Liang Wei[d,*]

[a]*Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, 250014, China*
[b]*Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing, Shandong Fundamental Research Center for Computer Science, Jinan, 250014, China*
[c]*Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao, 266580, China*
[d]*Jiangsu Future Network Innovation Institute, Nanjing, 211111, China*

Abstract

FPGA-based SmartNICs are increasingly adopted in data centers to enable high-speed and low-latency network acceleration. However, existing FPGA virtualization techniques suffer from limited elasticity, redundant function instantiation, and resource contention. These issues make them unsuitable for multi-tenant scenarios. To overcome these challenges, we propose RosebudFlex, a partially reconfigurable FPGA virtualization framework based on spatial division multiplexing. This framework introduces two key abstractions: Private Function Regions for hardware-level tenant isolation, and Flexible Function Regions for shared and dynamically partially reconfigurable network functions. This design enables fine-grained customization and efficient resource reuse. It supports customizable acceleration by enabling Private and Flexible Function Regions via partial reconfiguration. RosebudFlex supports parallel multi-tenant packet processing by isolation in Private Function Regions with service dispatcher controlled sharing in Flexible Function Regions. Analytical results indicate that flexible function region can reduce function blocking probability under the assumptions of uniform demand and service dispatcher controlled access. Experimental results on Xilinx UltraScale and FPGA demonstrate that RosebudFlex reduces logic resource usage by up to 16.8% compared to static partitioning while sustaining 100 Gbps throughput across tenants. Furthermore, evaluation under burst traffic attacks confirms that the proposed isolation mechanism effectively shields victim tenants, guaranteeing performance stability with negligible reconfiguration overhead.

Keywords: Field-Programmable Gate Array, Partial Reconfiguration, SmartNIC, Network Function Offloading, Multi-Tenant Network.

1. Introduction

Field-Programmable Gate Arrays (FPGAs) have evolved from application-specific accelerators into critical components of modern data center networks [1]. FPGA-based smart network interface cards (SmartNICs) [2] achieve high-speed and low-latency data exchange by offloading the network protocol stack from the CPU. To provide high-speed network acceleration capabilities to multiple tenants on demand [3], cloud service providers are exploring FPGA-based network virtualization technologies [4]. However, the mainstream slicing-based approach statically partitions the FPGA into dedicated logic regions and shared resource areas. This approach struggles to meet the elasticity [5], fairness [6], and high-performance demands of multi-tenant environments.

In a multi-tenant environment, FPGA virtualization [7] faces several fundamental challenges. First, rigidly binding each tenant to a fixed resource quota causes excess traffic to be throttled. This occurs even when idle hardware resources are available elsewhere and leads to degraded performance and inefficient hardware utilization [8]. Second, when multiple tenants independently load identical network functions, the repeated instantiation significantly increases the usage and configuration latency of LUTs and BRAMs. Third, malicious or misconfigured tenants may monopolize PCIe bandwidth [9]. This increases latency for well-behaved tenants [10]. Therefore, despite the performance benefits offered by FPGAs, their limitations remain pronounced in multi-tenant scenarios.

To address these challenges, we propose RosebudFlex. It is a high-performance partially-reconfigurable FPGA virtualization framework based on spatial division multiplexing and specifically designed for multi-tenant network function acceleration [11]. Compared to Rosebud [12], RosebudFlex treats the hardware accelerator as a collection of multiple microservers. It introduces two key abstractions with a clear division of labor. Private Function Regions are dedicated to individual tenants. They integrate isolated send and receive queues, DMA streams, and tenant-specific custom accelerators via SR-IOV [13]. This ensures hardware-level isolation and prevents cross-tenant interference. Flexible Function Regions are a shared and dynamically reconfigurable pool. The term Flexible Function

*Corresponding author.
Email addresses: 10431240209@stu.qlu.edu.cn (Xin Dong), tanlzh@sdas.org (Lizhuang Tan), shihl@sdas.org (Huiling Shi), wzhang@sdas.org (Wei Zhang), zhangpeiying@upc.edu.cn (Peiying Zhang), weiliang@fnii.cn (Liang Wei)

Region refers to a shared and dynamically assignable function pool for commonly used network functions. The term flexible does not imply that only this region is reconfigurable, since Private Function Regions can also be updated through partial reconfiguration. The key distinction is that Private Function Regions are tenant-dedicated, whereas Flexible Function Regions provide dispatcher-mediated sharing of reusable network functions. They host common network functions frequently used by multiple tenants. These functions include AES encryption, firewall, and compression. RosebudFlex deploys a single copy of each shared function instead of instantiating them for each tenant. This eliminates redundant resource consumption and reduces function blocking probability. An embedded RISC-V [14] core in the FPGA manages the Flexible Function Regions in software. It enables runtime deployment, updates, and scheduling of shared functions. It works without disrupting tenant-specific workloads in Private Function Regions.

RosebudFlex is designed to support parallel packet processing for customized acceleration across multiple tenants. This work has made the following contributions:

- We design and implement RosebudFlex to support parallel processing of network acceleration requests in multi-tenant environments. It satisfies customized network processing needs and significantly improves the utilization of virtualized FPGA resources.
- Building on RosebudVirt [15], RosebudFlex decouples each Reconfigurable Packet Processing Unit (RPU) into two distinct abstractions which are Private Function Regions and Flexible Function Regions. This allows fine-grained customization of functional requirements across regions.
- We theoretically demonstrate that RosebudFlex achieves higher performance gains in multi-tenant scenarios by significantly reducing function blocking probability through the introduction of shared function regions.
- Through FPGA virtualization into multiple Private Function Regions mapped to individual virtual functions (VFs), RosebudFlex incurs near-native performance. It reduces FPGA logic resource usage up to 16.8%. Even under high load, the forwarding latency due to dynamic function hot-swapping remains negligible.

The remainder of this paper is organized as follows. Section 2 presents the relevant background and motivation. Section 3 analyzes the performance gains and blocking probabilities of RosebudFlex in comparison with RosebudVirt. Section 4 describes the overall architecture of RosebudFlex. Section 5 details its implementation. Section 6 evaluates the throughput and latency of RosebudFlex. Finally, Section 7 summarizes its advantages and future prospects. The symbols and their definitions used throughout the paper are listed in Table 1.

Table 1: Description of Symbols

Parameter	Description
$\mathcal{N}$	The number of tenants
n_{i}	The number of each tenant's exclusive functions
s	The number of shared functions
S	Normalized reconfigurable FPGA resource capacity
P_{virt}	Blocking probability of Rosebudvirt
P_{Flex}	Blocking probability of RosebudFlex
ΔP	The performance gain of RosebudFlex
ρ	Shared area coverage
inP	Incoming packet
$inP.\mathrm{L}$	Packet length
$\mathcal{T}$	Tenant state object
$\mathcal{T}.K$	Current token count
$\mathcal{T}.K_{\max}$	Token bucket capacity
$\mathcal{T}.T_{\mathrm{last}}$	Last update timestamp
$\mathcal{T}.\phi$	Token fill rate
T_{curr}	Current timestamp
ΔT	Elapsed time since last update
L_{used}	Consumed tokens for current packet
$\mathcal{T}.A$	Anomaly (throttle) count
$\mathcal{T}.W$	Sliding window of recent usage
$\mathcal{T}.W_{\mathrm{end}}$	End timestamp of current window
$\mathcal{T}.\omega$	Window interval
$\mathcal{T}.\theta$	Global usage threshold
$\mathcal{T}.\delta$	Consecutive anomaly limit
A_i	Activity level of tenant i
L_i	Network load of tenant i
U_i	Resource saturation
$T_{\mathrm{low}}, T_{\mathrm{high}}$	Load thresholds for scoring adjustment
L_s	Load adjustment score: $\{-1, 0, +1\}$
S_i	Scheduling score of tenant i
R_i	Allocated resource area to tenant i
R_{total}	Total available reconfigurable resource area
α, β, γ	Weight factors ($\alpha + \beta + \gamma = 1$)

2. Related Work and Motivation

In this section, we briefly review the state of the art in FPGA virtualization, partial reconfiguration, and tenant isolation, and present the motivation for RosebudFlex.

2.1. FPGA Virtualization for Network Function Acceleration

In general, virtualization refers to the creation of virtual abstractions of computing resources to conceal low-level hardware details from users. This abstraction is implemented in software. It allows users to interact with virtual resources without direct knowledge of the underlying complexity. Virtualized resources are transparently presented to users. This gives the illusion of unlimited and exclusive access [23]. A classical example of virtualization is the use of virtual machines (VMs) to run multiple operating systems on a single processor. VMs create the appearance of independent computing environments. This enables users to switch between different system setups or operating systems without modifying physical hardware [24].

Compared to traditional software virtualization, FPGA virtualization aims to abstract reconfigurable hardware resources. This shields users or application developers from intricate

Table 2: Comparison of FPGA Virtualization Frameworks

Framework	I/O Virtual	Multi-tenant	Reconfigurable	Hardware Acc.	Network-Oriented	Fairness
FPGAVirt [16] (2018)	Y	Y	N	Y	N	N
FFIVE [17] (2021)	N	Y	N	Y	N	Y
HiPR [18] (2022)	Y	Y	N	Y	Y	Y
Nimblock [19] (2023)	N	Y	N	N	N	Y
Lemon-NFV [20] (2023)	N	Y	Y	Y	N	Y
PR-ESP [21] (2023)	Y	Y	N	N	Y	Y
Rosebud [12] (2023)	N	N	Y	Y	N	N
S-NIC [22] (2024)	N	N	Y	N	N	N
SuperNIC [2] (2024)	N	Y	Y	Y	Y	Y
Rosebudvirt [15] (2024)	Y	Y	Y	Y	Y	N

hardware-level details. The definition of FPGA virtualization has evolved over time. Early work in the 1990s applied operating system concepts such as partitioning, segmentation, and stacking to FPGAs. These techniques were collectively referred to as FPGA virtualization. Later, in the 2000s, the concept matured into a layered abstraction over FPGA hardware which is commonly referred to today as an overlay architecture [25].

Three major approaches to FPGA virtualization have been identified. These are temporal partitioning, virtualized execution, and mapping to abstract virtual machines [4]. With the rapid evolution of FPGA technology in recent decades, the goals of FPGA virtualization have also shifted. In modern cloud and edge computing contexts, the focus has expanded to include application-level accessibility, efficient multi-tenant sharing, and robust resource isolation.

The primary goals of FPGA virtualization can be summarized as multi-tenancy, resource management, flexibility, isolation, scalability, performance, security, resilience, and programmer productivity [26]. While we acknowledge these objectives, we argue that scalability, performance, and resilience are broader goals of cloud and edge infrastructure rather than being specific to FPGA virtualization. Conversely, flexibility and developer productivity are more closely aligned with high-level synthesis (HLS) technologies than with virtualization itself. Various solutions have been proposed for FPGA-based network function acceleration as shown in Table 2. For example, Single Root I/O Virtualization (SR-IOV) is widely used to partition a single FPGA into multiple virtual functions (VFs). This enables resource sharing across tenants.

SuperNIC [2] enables multi-tenant SmartNICs by mapping network functions to physical chains through partial reconfiguration. It achieves fair spatial and temporal hardware sharing even though it lacks efficient I/O virtualization. FPGAVirt [16] leverages Virtio for communication between VMs and FPGAs. However, its static allocation scheme fails to meet diverse tenant demands. High-demand tenants face resource bottlenecks while low-demand tenants leave resources underutilized. Nimblock [19] manages FPGA resources by creating isolated and secure environments per tenant. It dynamically allocates resources via a preemption-based scheduler to meet differentiated tenant requirements and improve responsiveness. FFIVE [17] incorporates Kubernetes and Docker to manage FPGA-based virtual connections. It enables virtual function migration and aligning FPGA resource deployment with container-based virtual network functions (VNFs). However, it does not address resource allocation challenges.

2.2. Partial Reconfiguration

A partial reconfiguration (PR) design consists of two primary elements. These are the static design and reconfigurable modules. The static design refers to the portion of the FPGA fabric that remains unchanged during reconfiguration while reconfigurable modules represent the portions that can be dynamically reloaded at runtime [27].

Hierarchical partial reconfiguration (Hierarchical PR) [28] is an advanced technique that enables the nesting of one or more reconfigurable modules within another reconfigurable module. This facilitates greater modularity. In Xilinx Vivado's PR flow, designers define PR regions using pblocks and assign a reconfigurable module to each region. After placement and routing, the design flow isolates the reconfigurable modules and locks the rest of the logic and routing to produce the static portion. Interfaces between static and reconfigurable logic are known as partition pins. These pins are also specified. Notably, Vivado permits static logic to temporarily utilize resources within PR pblocks to enhance routability during implementation [29].

PR enables functional modules to be dynamically loaded based on demand. This significantly improves hardware flexibility. As summarized in Table 2, while half of the surveyed frameworks support partial reconfiguration, there remains a significant gap in combining dynamic hardware reconfiguration with efficient I/O virtualization and multi-tenant fairness. For example, HiPR [18] introduces an open-source framework that allows developers to define partially reconfigurable C/C++ functions. It supports incremental FPGA compilation and automates the transformation from high-level language to bitstream. Nevertheless, HiPR combines high-level synthesis and PR but does not address low-level concerns such as resource or I/O allocation. Similarly, PR-ESP [21] is an open-source platform for system-level design targeting SoC architectures built on partially reconfigurable FPGAs. PR-ESP also aims to reduce total compilation time by automatically generating bitstreams.

2.3. Isolation in FPGA Virtualization

In a multi-tenant FPGA virtualization system, each tenant may share access to available hardware resources such as one or more partially reconfigurable regions, a subset of I/O interfaces, and on-chip or off-chip memory [30]. To maintain security and stability, the system must strictly enforce access controls so that tenants can only access their own allocated resources at authorized times. Tenants must be prevented from accessing or tampering with other tenants' data or interfering with the execution of other accelerators. From a hardware stack perspective, FPGA virtualization requires three fundamental types of isolation. These are functional isolation, performance isolation, and fault isolation. Traditional software-based scheduling strategies struggle to provide precise low-level hardware isolation and often lack the responsiveness to mitigate malicious resource contention in a timely manner. For example, S-NIC [22] implements extensive virtualization of hardware accelerators by enforcing single-owner semantics for each line of NIC cache and RAM. It also provides dedicated bus bandwidth per network function. This ensures strong hardware isolation and gives each function the illusion of an exclusive intelligent NIC. Additionally, LemonNFV [20] introduces a novel NFV framework that supports the integration of heterogeneous network functions without code modification. It achieves this by transforming traditional NFs into Least Modified Network functions. LemonNFV enables function switching through schedulable I/O and leverages Intel's Protection Keys for User space to enforce in-process memory isolation among NFs.

2.4. Motivation

Rosebud [12] is a framework for network function acceleration that leverages partial reconfiguration to partition the FPGA fabric. Its reconfigurable packet processing units (RPUs) integrate 64-bit RISC-V cores with hardware accelerators in dynamic regions to enable data plane processing at rates up to 200 Gbps. However, Rosebud focuses solely on architectural-level virtualization while overlooking I/O resource virtualization. To address this limitation, RosebudVirt [15] extends the architecture by virtualizing I/O resources and introducing load-balancing mechanisms to maintain high-throughput performance. It sustains bandwidths close to 200 Gbps. Nevertheless, both systems lack support for tenant-specific customization in multi-tenant environments. They provide a uniform network function to all tenants without considering functional diversity or tenant isolation at the virtualization layer. While FPGA-based frameworks such as Rosebud and RosebudVirt demonstrate promising performance in network function acceleration, their virtualization strategies remain coarse-grained. In particular, they lack support for function-level differentiation and tenant-specific customization in multi-tenant deployments. This gap motivates the need for a more fine-grained and flexible FPGA virtualization approach that addresses both architectural and I/O isolation while supporting diverse network functions across tenants.

3. Theoretical Analysis

This section analyzes the performance gain and function blocking probability of RosebudFlex compared to RosebudVirt [15]. The Flexible Function Region serves as the core design premise of our theoretical model. The Flexible Function Region is a shared pool that hosts several common network functions. All tenants can access these functions via time-division multiplexing. This eliminates redundant instantiation of shared functions. RosebudVirt has a critical limitation in this aspect. Each tenant must have its own copy of all required functions in RosebudVirt. This design basis supports the performance gain and function blocking probability reduction of RosebudFlex.

3.1. Performance Gain

Assuming the normalized reconfigurable FPGA resource capacity is S, the number of tenants is N, the average tenant active ratio is a, the public area is S_p, and the sharing ratio is ρ where $\rho = S_p/S$, the performance gain P_G brought by the RosebudFlex is

$$P_G = \frac{\frac{S-S_p}{N} + \frac{S_p}{a*N}}{\frac{S}{N}} - 1, \tag{1}$$

where $\frac{S-S_p}{N} + \frac{S_p}{a*N}$ is the virtualization resources obtained by each tenant of RosebudFlex, and $\frac{S}{N}$ is of RosebudVirt. We simplify and get

$$P_G = \rho * (\frac{1}{a} - 1). \tag{2}$$

As shown in Figure 1, as the a decreases or ρ increases, the gain of RosebudFlex increases. When $a = 100\%$, or $\rho = 0$, RosebudFlex degenerates to RosebudVirt. Although ρ has a positive effect on the gain, due to the strong isolation constraints between different tenants, ρ cannot be 100%.

3.2. Function Blocking Probability

Function blocking probability is the probability that a tenant network function request is rejected due to insufficient resources. We analyze the function blocking probability of FPGA function access under two resource allocation schemes which are the traditional static partitioning scheme RosebudVirt and our proposed dynamic sharing scheme RosebudFlex. Let the total number of functions be denoted as m, the number of tenants as N, shared resource ratio as ρ, and each tenant be assigned n_i exclusive functions. Additionally, s functions are globally marked as shared resources. The overall resource constraint must be satisfied

$$\sum_{i=1}^{N} n_i + s \leq m. \tag{3}$$

We assume a uniform function access distribution and no contention in the RosebudFlex scheme due to function-level isolation and dynamic scheduling. In RosebudVirt, shared functions are statically and evenly divided among tenants. Each tenant accesses $n_i + \frac{s}{N}$ functions, and thus the function blocking probability for tenant i is

$$P_{\text{Virt}} = \frac{m - n_i - \frac{s}{N}}{m}. \tag{4}$$

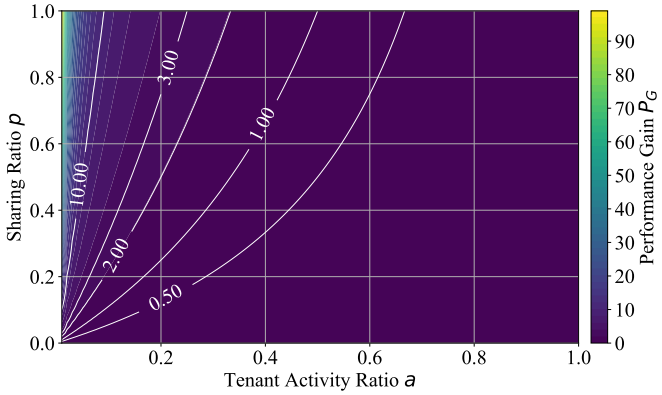

Figure 1: Performance gain of RosebudFlex.

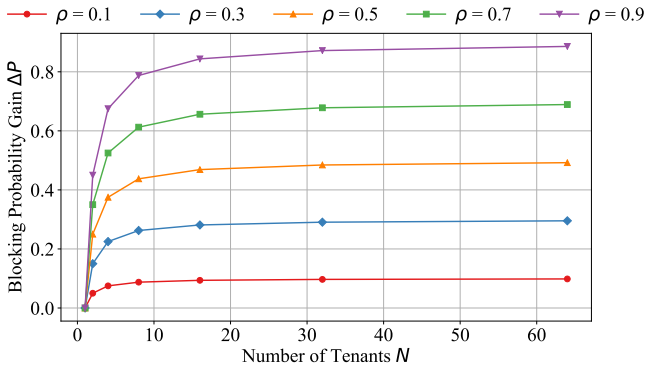

Figure 2: Blocking probability gain of RosebudFlex.

In RosebudFlex, all shared functions are globally accessible by all tenants. With contention eliminated, each tenant can access $n_i + s$ functions, and the function blocking probability becomes

$$P_{\text{Flex}} = \frac{m - n_i - s}{m}. \tag{5}$$

The performance gain of RosebudFlex over RosebudVirt is

$$\Delta P = P_{\text{Virt}} - P_{\text{Flex}} = \frac{s}{m} \cdot \left(1 - \frac{1}{N}\right). \tag{6}$$

As shown in Figure 2, when the number of tenants N is small, the difference between RosebudVirt and RosebudFlex is limited. However, as N increases, the shared resources per tenant in RosebudVirt shrink proportionally. RosebudFlex maintains access to the full shared pool. This leads to significantly lower function blocking probability under RosebudFlex in high-concurrency scenarios.

4. Architecture of RosebudFlex

To address the aforementioned challenges, we propose an FPGA-based hardware-accelerated packet processing architecture named RosebudFlex that enables packet-level sharing between a RISC-V [31] control core and specialized hardware accelerators. The system is composed of two primary components which are the Private Function Region and the Flexible Function Region.

4.1. Hardware Design

RosebudFlex targets network acceleration workloads by leveraging several key FPGA components. These include the SR-IOV PCIe interface, a customized Service Dispatcher, multiple vFPGA accelerators as Private Function Regions, and Flexible Function Regions. As illustrated in Figure 3, the hardware architecture comprises both fixed static regions and dynamically reconfigurable regions. By incorporating programmable Flexible Function Regions, Private Function Regions, and dedicated virtualization modules, RosebudFlex enhances the ability of resource sharing, reuse, and hardware isolation in multi-tenant environments.

Private Function Regions utilize SR-IOV [32] to divide FPGA into multiple VFs and one Physical Function (PF). Every VF includes isolated send and receive queues and bandwidth controls which ensures per-tenant isolation. Each VF is mapped on the corresponding virtual FPGA (vFPGA), a virtualized logical region divided from the physical FPGA to serve a single tenant. These vFPGAs host a dedicated RISC-V core and hardware accelerator per tenant and can be dynamically configured. During normal packet processing, the physical ports of accelerator-local memory are reserved for the accelerator datapath to avoid contention with DMA or control-plane access. During initialization, readback, or partial reconfiguration, packet forwarding to the target region is paused and the accelerator is quiesced. The DMA or control path can then safely initialize, save, or restore accelerator-local state before traffic is resumed.

Flexible Function Regions host frequently-used network functions such as AES encryption, compression, and firewall that are either preloaded or dynamically loaded at runtime. These functions are shared across vFPGAs and use pipelines [33] and time-division multiplexing [34] to improve utilization. Public resources are dynamically assigned to high-demand tenants as needed.

Control Plane is located within the PF of the FPGA. This control module maintains the hardware-level function mapping table, handles dynamic resource allocation, enforces scheduling decisions, and detects misbehavior. It supports traffic control via mechanisms such as token buckets and sliding windows. The runtime allows idle public functions to be reassigned to more active tenants.

SR-IOV PCIe Interface uses the SR-IOV protocol to enable multiple VFs to share PCIe resources while exposing isolated interfaces to each tenant. Our architecture demonstrates four VFs mapped to four distinct tenants. These tenants can submit packets to shared accelerators in the Flexible Function Regions. SR-IOV PCIe transports descriptors using Transaction Layer Packets (TLPs) for efficient DMA-based access [35].

Packet Switch, which can inspect destination MAC addresses and assign forwarding rules [36], routes packets to the appropriate vFPGA. This subsystem supports network ingress and egress, and then handles control-plane responsibilities such as resource orchestration across vFPGAs.

Accelerator Architecture enables the RISC-V core to communicate with the accelerator through a dedicated memory-mapped I/O (MMIO) [37] interface for configuration and result retrieval. This interface allows read and write access to the

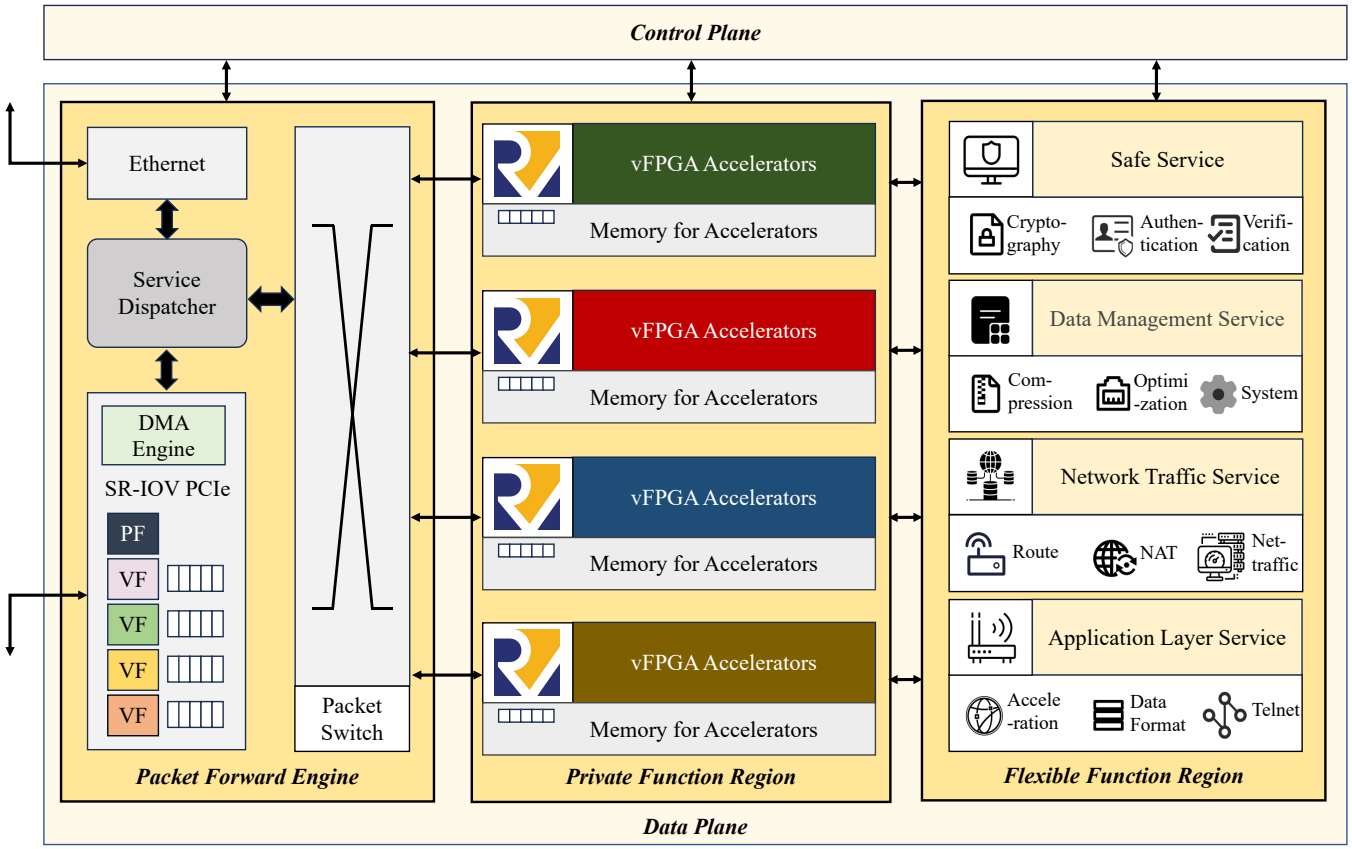

Figure 3: Overview of the RosebudFlex Framework.

accelerator's control registers. The RISC-V core can either be placed for full customization or implemented outside the PR region as part of the static support infrastructure. While the latter improves persistence, it introduces additional latency when accessing shared memory and accelerators due to the crossing of reconfiguration boundaries.

To perform a partial reconfiguration update, the following sequence is executed. The host instructs the DMA engine to pause packet transmission to the target RPU. The system waits for all in-flight packets to drain from the accelerator pipeline. The updated RISC-V core is initialized in the reconfigured Flexible Function Regions. Finally, the DMA engine resumes traffic forwarding to the Private Function Regions. This mechanism enables dynamic updates of logic blocks at runtime while preserving isolation and minimizing disruption to active tenants.

4.2. Microarchitecture of vFPGA

Figure 4 shows the microarchitecture of a vFPGA in the Private Function Region. Each vFPGA contains a software-programmable RISC-V control core, a dedicated hardware accelerator, local memories, a local DMA controller, and a system interconnect. The RISC-V core is responsible for rule management, accelerator configuration, DMA coordination, and exceptional packet handling, while line-rate packet classification and payload processing are performed by hardware datapaths.

The architecture separates the MMIO control path from the packet datapath. The MMIO path is used to configure accel-

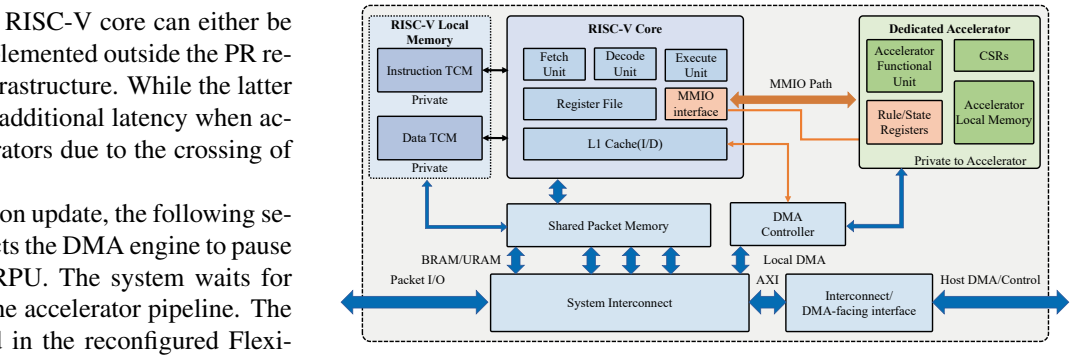

Figure 4: Microarchitecture of a vFPGA.

erator CSRs and control/status registers. The packet datapath moves descriptors and packet payloads through the system interconnect and shared packet memory. The left external interface connects to the Service Dispatcher or packet switch for packet input and output, while the right external interface connects to the PCIe static shell for host-side control, initialization, debugging, and readback.

4.3. Service Dispatcher

Each vFPGA contains a RISC-V control core and a tenant-specific accelerator datapath. The two components interact through two separated paths. The AXI-Lite MMIO path is used by the RISC-V core to configure accelerator CSRs, update

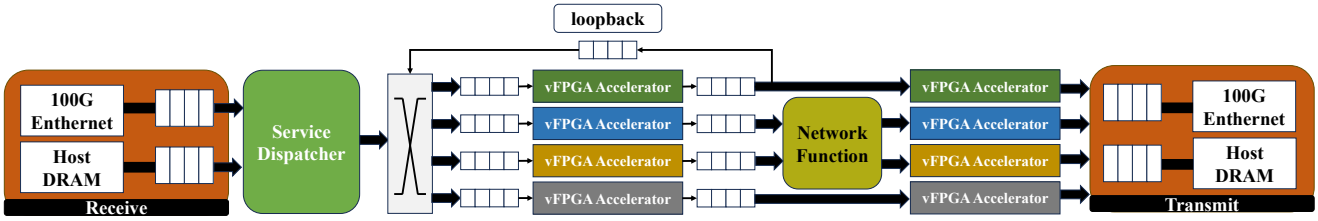

Figure 5: Packet distribution in RosebudFlex.

rules, and read status. The packet datapath is used by the Service Dispatcher and DMA engine to move packet descriptors and payloads through slot-based packet buffers. This separation prevents control-plane accesses from interfering with line-rate packet processing. Figure 5 illustrates the overall packet flow from network ingress to RosebudFlex.

RosebudFlex provides different isolation guarantees for private and shared regions. Private Function Regions are mapped one-to-one to tenant VFs and enforce isolation through dedicated TX/RX queues, DMA descriptors, MMIO address ranges, packet-buffer slots, and per-tenant rate policers. Shared Flexible Function Regions are not exposed as tenant-private execution domains. They are accessed only through the Service Dispatcher. Each shared-function request carries a tenant context ID, function ID, and packet-slot ID. The dispatcher validates these fields before forwarding the request to a shared function. Therefore, tenants cannot directly access another tenant's packet buffers, control registers, or shared-function state.

Incoming packets first arrive at either a physical or virtual Ethernet interface. These packets are then forwarded by the service dispatcher to the appropriate vFPGA accelerator. The vFPGA accelerator calls the corresponding Flexible Function Region to accelerate the request of the data packet. Additionally, packets may originate from internal sources primarily in loopback scenarios. Loopback is triggered when network functions required by a packet span multiple vFPGAs. In such cases, one vFPGA may request processing by another which results in a backhaul operation. Since loopback traffic is intra-board and non-network-facing, it typically incurs lower volume and shares the same infrastructure as external traffic without sacrificing overall throughput.

To ensure efficient and non-blocking service dispatcher, each ingress link is associated with a dedicated FIFO buffer. This per-link FIFO architecture allows packets to be transmitted at full interface width without interfering with other links. In situations where multiple packets target the same vFPGA, a dual detection strategy will be used to determine whether the traffic is malicious as described in the Section 5.

The packet switching fabric is designed to be unidirectional with separate pathways for inbound and outbound traffic. This ensures isolation between receive and transmit queues. Each vFPGA's interconnect module is responsible for address decoding and acts as a bridge between the Service Dispatcher and the RISC-V core. It also interfaces with the service dispatcher and the host-side DRAM [38] access manager.

Coordination among components is handled through dedicated control channels. One channel handles messages between the vFPGA and the service dispatcher while another manages memory access requests to the DRAM controller. These control paths are physically separated from the data paths to prevent contention.

The host system can access status counters associated with all physical and virtual Ethernet interfaces as well as each vFPGA. These counters record performance metrics such as the number of transferred bytes, frame counts, dropped packets, and stalled cycles. They enable developers to observe packet flow dynamics, such as how the DMA engine distributes packets, and to identify potential performance bottlenecks in the system [39].

4.4. Transmission Between vFPGA

In our architecture, data packets may require cross-vFPGA state sharing which occurs in two primary scenarios. First, sending short control messages to update the internal state of other processors. Second, forwarding packets that require multiple network functions distributed across different vFPGAs based on functional dependencies. RosebudFlex does not allow arbitrary data exchange between tenant vFPGAs. As shown in Figure 5, when a packet requires multiple network functions, the vFPGA submits a service chain request to the Service Dispatcher. The request includes the tenant context ID, function ID, and packet-buffer slot ID. The dispatcher then invokes the corresponding shared function or forwards the packet to an authorized processing stage according to the configured service chain table. This design avoids direct cross tenants memory access and keeps all inter-region communication under control plane policy enforcement. Due to limited on-chip memory and relatively low clock frequency in FPGAs, implementing a cache-coherent shared memory model is inefficient and impractical. Consequently, our system introduces a lightweight and message-passing communication model that is tailored to these use cases to optimize resource efficiency. Additionally, for exchanging short messages or control metadata between vFPGA Accelerators, the system provides dedicated FIFO [40] channels. These FIFOs help maintain message ordering and reliability ensuring no loss or reordering during intra-FPGA communication.

The Service Dispatcher is not a traditional load balancer across interchangeable workers. Since each vFPGA is mapped to a specific tenant VF, the dispatcher primarily performs traffic classification, tenant steering, shared-function scheduling, and

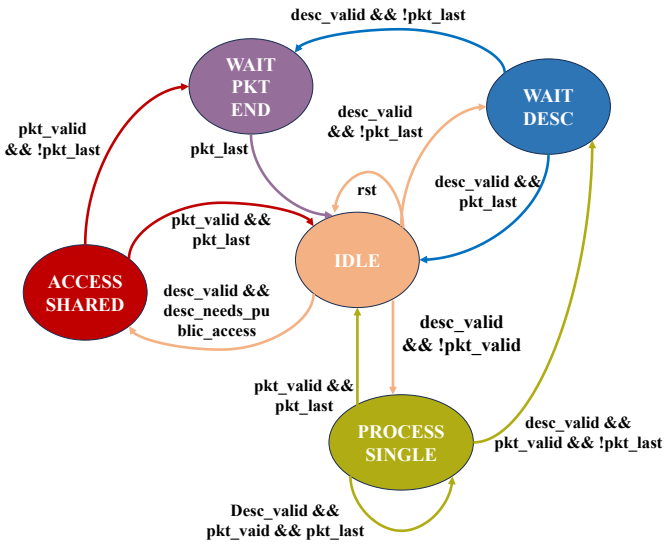

Figure 6: FSM of the Service Dispatcher.

traffic policing. For PCIe ingress traffic, the dispatcher identifies the tenant according to the VF or TLP function field. For network ingress traffic, it matches packet headers such as destination MAC address, VLAN tag, IP address, or flow-table entry. After classification, the dispatcher attaches a tenant context ID and forwards the packet descriptor to the corresponding vFPGA or to an authorized shared function.

As shown in Figure 6, the service dispatcher forward packet payloads to shared memory buffers while forwarding headers to RISC-V local memory for low-latency parsing. It also supports memory initialization and debug-level readback. Additionally, it can initialize memory regions from the host and read memory back for debugging purposes. Each packet is referenced by a descriptor, specifically a slot number, which points to a buffer in shared packet memory. The service dispatcher is thus primarily responsible for applying the load-balancing policy by tagging incoming packets with the destination vFPGA ID and assigned memory slot. These slot assignments are configured by RISC-V software at boot time which registers the number of available slots and their size with the service dispatcher.

4.5. Service Dispatcher Design

Similarly, communication between vFPGAs and host DRAM is also managed through slot-based descriptors referred to as DRAM tags. This enables symmetric access patterns and consistent memory mapping [41]. Once a packet is dispatched to a vFPGA, its local interconnect receives the descriptor and notifies the RISC-V core of its arrival. When sending a packet, the RISC-V core has two options. It can use direct transmission to instruct the interconnect to immediately forward the packet. Alternatively, it can use deferred transmission to inform the service dispatcher which slot is ready. The dispatcher then issues the corresponding transfer command. After packet transmission, the interconnect notifies the service dispatcher that the slot has been released and is available for reuse. This slot abstraction effectively decouples the system into a centralized control plane and distributed dataplanes improving system scalability and modularity.

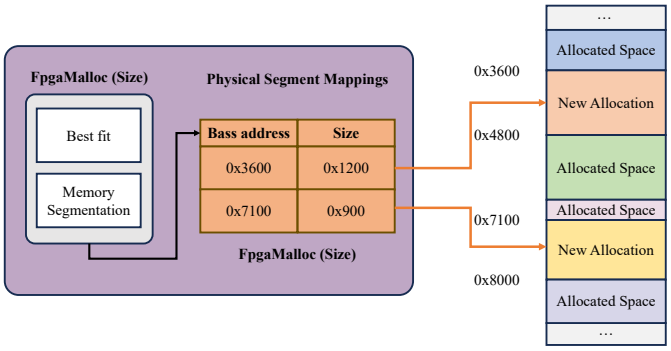

Figure 7: Allocation example that extends to multiple physical segments.

4.6. Resource Management

In order to achieve dynamic function sharing and resource scheduling in the above system, we present an example of our resource allocation mechanism in Figure 7. This mechanism spans multiple physical memory segments. The proposed memory segmentation scheme effectively manages the FPGA address space while ensuring isolation among tenants. It employs an optimization strategy to identify the most appropriate available segment for allocation. By aggregating distinct physical segments into a unified virtual memory region, the scheme creates a contiguous address space sufficient to support the operating system.

To achieve efficient and fair utilization of shared FPGA resources, we propose a tenant-aware resource allocation algorithm. In this algorithm, RosebudFlex collects three statistical indicators periodically. These are tenant activity level A_i measured by counting the number of network function invocations by tenant i over a recent period, network load L_i calculated as the ratio of current bandwidth or packet rate of tenant i to the total, and resource usage saturation U_i defined as the ratio of pending tasks in the shared resource area to the maximum queuing capacity of RosebudFlex. This algorithm dynamically adjusts resource assignments based on tenant activity, network load, and resource usage saturation by following the steps below:

(1) Load Adjustment Factor: Compute a load-based adjustment score based on L_i as

$$\text{LoadScore}(L_i) = \begin{cases} +1, & L_i < T_{\text{low}} \\ 0, & T_{\text{low}} \le L_i \le T_{\text{high}}, \\ -1, & L_i > T_{\text{high}} \end{cases} \tag{7}$$

where T_{low} and T_{high} are predefined thresholds for incentivizing or suppressing resource allocations.

(2) Scheduling Score: Combine all metrics into a unified score S_i for tenant i as

$$S_i = \alpha \cdot A_i + \beta \cdot U_i + \gamma \cdot \text{LoadScore}(L_i), \tag{8}$$

where $\alpha + \beta + \gamma = 1$ are configurable weight parameters.

Algorithm 1 Malicious Tenant Detection

Require: inP: incoming packet, $\mathcal{T}$: tenant state object
Ensure: Boolean: whether the tenant is malicious

$T_{\text{curr}} \leftarrow$ get_timestamp()
$\Delta T \leftarrow T_{\text{curr}} - \mathcal{T}.T_{\text{last}}$
$\mathcal{T}.K \leftarrow \min(\mathcal{T}.K_{\max},\ \mathcal{T}.K + \Delta T \times \mathcal{T}.\phi)$
$\mathcal{T}.T_{\text{last}} \leftarrow T_{\text{curr}}$
if $inP.L \le \mathcal{T}.K$ **then**
    $\mathcal{T}.K \leftarrow \mathcal{T}.K - inP.L$
    $L_{\text{used}} \leftarrow inP.L$
    $\mathcal{T}.A \leftarrow \max(0, \mathcal{T}.A - 1)$
else
    $L_{\text{used}} \leftarrow 0$
    $\mathcal{T}.A \leftarrow \mathcal{T}.A + 1$
end if
while $T_{\text{curr}} \ge \mathcal{T}.W_{\text{end}}$ **do**
    Remove first entry from $\mathcal{T}.W$
    Append 0 to $\mathcal{T}.W$
    $\mathcal{T}.W_{\text{end}} \leftarrow \mathcal{T}.W_{\text{end}} + \mathcal{T}.\omega$
end while
$\mathcal{T}.W[-1] \leftarrow \mathcal{T}.W[-1] + L_{\text{used}}$
if $\sum(\mathcal{T}.W) > \mathcal{T}.\theta$ **or** $\mathcal{T}.A \ge \mathcal{T}.\delta$ **then**
    return True
else
    return False
end if

(3) Global Demand Estimation: Compute the total system-wide scheduling demand $\sum_{j=1}^{N} \left|S_j\right|$, which represents the global contention level for shared resources.

(4) Resource Allocation: Allocate shared resources to each tenant proportionally as

$$R_i = R_{\text{total}} \cdot \frac{|S_i|}{\sum_{j=1}^{N} \left|S_j\right|}, \tag{9}$$

where R_i is the resource area allocated to tenant i, and R_{total} is the total size of the shared resource pool.

4.7. Malicious Tenant Detection

The parameters in Algorithm 1 are derived from the tenant SLA and packet-processing configuration. The token generation rate $T.\phi$ is set according to the bandwidth quota assigned to the tenant VF. The bucket capacity $T.K_{\max}$ is configured as the allowed burst size, calculated as the SLA bandwidth multiplied by the tolerated burst duration. The sliding-window length $T.\omega$ is set according to the monitoring interval of the dispatcher counters. The threshold $T.\theta$ is computed as the maximum allowed traffic volume within one window, optionally multiplied by a tolerance factor to avoid false positives caused by short bursts. The anomaly threshold $T.\delta$ specifies the number of consecutive token-exhaustion events required before throttling is triggered.

To ensure fairness and security in multi-tenant environments, RosebudFlex integrates a runtime anomaly detection mechanism within the Service Dispatcher. As detailed in Algorithm 1, we employ a hybrid approach combining a Sliding Window Counter [42] and a Token Bucket to monitor tenant request rates in real-time. The algorithm tracks the cumulative traffic usage within a sliding time window and monitors instantaneous burstiness using token availability. If a tenant's cumulative usage exceeds the preset threshold ($\mathcal{T}.\theta$) or if they trigger consecutive token exhaustion events ($\mathcal{T}.A \ge \mathcal{T}.\delta$), they are flagged as malicious. Upon detection, the system triggers isolation measures, such as dynamically reducing the token generation rate or redirecting traffic to a low-priority queue to protect well-behaved tenants.

5. Implementation

We implemented RosebudFlex on the Xilinx Virtex UltraScale+ VU37P FPGA using the FPGA X37 board. The VU37P chip is equipped with two 4GB HBM2 stacks providing a total bandwidth of 460 GB/s. The board features 8GB of on-chip HBM2 memory, supports dual QSFP28 100 GbE interfaces, and offers PCIe Gen3 x16 connectivity. The experiment uses eight KVM virtual machines. Each virtual machine maps to one tenant VF. Virtual machine overhead stays below three percent because CPU-affinity tuning already cuts this cost.

5.1. Key Component Implement

Each vFPGA is assigned an independent partially reconfigurable partition and can interconnect with the Flexible Function Region to invoke network functions. At the core of each vFPGA resides a lightweight RISC-V soft-core processor that handles control logic and task coordination. Each vFPGA is also equipped with one or more accelerator units and multiple memory blocks for storing data packets, lookup tables, and intermediary results. A dedicated PR block is also reserved for the dispatcher enabling runtime reconfiguration. However, this necessitates state backup before reconfiguration which significantly increases resource overhead. Moreover, the lengthy reconfiguration process can limit timely updates to scheduling parameters.

The physical Ethernet interface on the FPGA board connects through a Media Access Control (MAC) module while a PCIe interface facilitates communication with the host. The host is capable of customizing each vFPGA instance based on tenant-specific requests. For instance, vFPGA1 may deploy a customized TCP engine, vFPGA2 may implement a QUIC protocol handler, and vFPGA3 may perform routing and forwarding operations. DMA-based communication with the host enables dynamic partial reconfiguration of vFPGA functions, also known as hot-swapping. While this increases system flexibility, it may also introduce latency overhead and additional consumption of RAM and auxiliary resources such as buffers for data caching and state backup.

Table 3 shows the resource utilization of different Rosebud variants. RosebudFlex-16 VFs denotes a configuration with 16 tenant-facing VFs and 16 corresponding vFPGAs. Resource savings come from function reuse in Flexible Function region. Multiple tenants share one common function such as AES. This

Table 3: Resource Utilization of Different Rosebud Framework Variants

Framework	SR-IOV	Multiplexing	LUTs	Registers	BRAM	URAM
RosebudFlex-16 vFs	Y	Y	216.0k (18.82%)	272.6k (11.78%)	542 (25.10%)	613 (63.91%)
RosebudVirt-16 RPUs	Y	N	253.3k (22.00%)	324.0k (14.03%)	547 (25.32%)	613 (63.91%)
Rosebud-16 RPUs	N	N	252.6k (21.97%)	318.4k (13.82%)	541 (25.21%)	613 (63.91%)
RosebudFlex-8 vFs	Y	Y	138.0k (11.87%)	197.1k (8.55%)	342 (15.78%)	331 (34.50%)
RosebudVirt-8 RPUs	Y	N	160.3k (13.91%)	219.6k (9.53%)	342 (15.78%)	331 (34.50%)
Rosebud-8 RPUs	N	N	159.9k (13.88%)	215.0k (9.35%)	339 (15.64%)	331 (34.50%)
RosebudFlex-4 vFs	Y	Y	117.6k (9.91%)	207.2k (8.98%)	357 (16.50%)	239 (24.91%)
RosebudVirt-4 RPUs	Y	N	134.3k (11.63%)	223.7k (9.69%)	354 (16.36%)	239 (24.91%)
Rosebud-4 RPUs	N	N	133.4k (11.56%)	219.1k (9.51%)	351 (16.23%)	239 (24.91%)

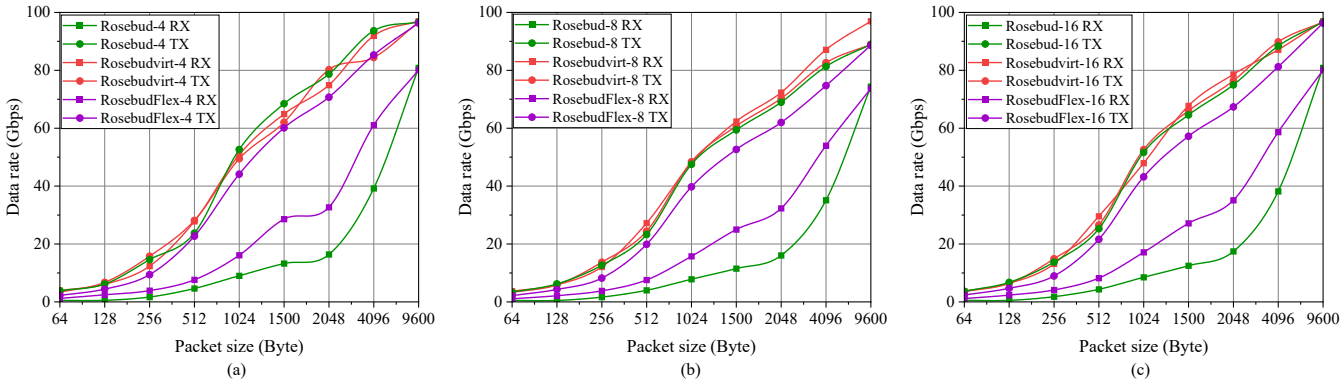

Figure 8: Comparison of the throughput between RosebudFlex, RosebudVirt and Rosebud. (a) 4 RPUs or vFPGAs. (b) 8 RPUs or vFPGAs. (c) 16 RPUs or vFPGAs.

sharing removes the redundant logic in RosebudVirt where each tenant had to instantiate its own copy. Rosebud and RosebudVirt consume approximately 22% of the LUTs, 14% of the registers, and 65% of the URAM in the entire design. In comparison, RosebudFlex improves resource efficiency by reducing LUT and register usage up to 16.8% while slightly increasing URAM utilization. This increase is negligible and highlights its enhanced data caching and resource reuse capabilities.

6. Evaluation

In this section, we introduce the limitations and advantages of RosebudFlex by analyzing several indicators. We use Xilinx VU37P FPGA equipped with 32GB DDR4 memory divided into 2 channels and two 100Gbps Ethernet ports on PCIe Gen3x16 expansion. The CPU used in the experiment is Intel Xeon Gold 5318Y@2.10GHz and 36M cache with 24 cores and a maximum thread count of 48.

6.1. Throughput

We evaluated packet sizes ranging from 64 bytes to 9600 bytes including standard data center MTUs such as 1500 and 9600 bytes. To assess throughput, we deployed a host-side script to simulate multiple flows, transmitting packets through the FPGA interface using a lightweight forwarding application, and verifying their successful delivery. When the packet rate was fixed at 125 MPPS, smaller packets were unable to fully saturate the 100 Gbps line rate.

As shown in Figure 8, experimental results show that throughput varied from 8.1 Gbps to 98.3 Gbps across different packet sizes and tenant counts. Figure 8 compares the throughput performance of Rosebud and RosebudVirt with our proposed RosebudFlex architecture under a 16-RPU configuration. The results demonstrate that RosebudFlex can support more tenants with customized features without significantly sacrificing throughput thus achieving better resource utilization. While RosebudFlex outperforms Rosebud in overall throughput, its performance is slightly below that of RosebudVirt. This marginal reduction is primarily due to the additional VF drivers and network interfaces introduced by the virtualization layer. Furthermore, the slight throughput degradation in RosebudFlex can be attributed to increased data caching and backup operations. However, this performance loss is negligible in practice.

Otherwise, as shown in Figure 9, in the baseline stress test at 100 Gbps, we evaluated the upper performance limit of RosebudFlex. As packet size increased, the system approached the 100 Gbps line rate. However, when the number of concurrent tenants became excessive, the effective throughput declined slightly due to VF contention and congestion. This indicates that the available VFs could not fully satisfy all tenant requests.

6.2. Latency

Next, we measured the latency of RosebudFlex's network function reconfiguration and forwarding latency. The former measures the time it takes from when the host receives a tenant's request for a new function to when it is sent to Rosebud-

Table 4: Dynamic Function Update Performance Metrics

Metric	Avg. time	Impact	Effect Details
Host to FPGA Command Delay	≤1 us	No	-
Bitstream Transfer Time	2–5 ms	Yes	Freezes packet flow
vFPGA Drain Time	10–100 us	Yes	Requires buffer
Partial Reconfiguration Time	3–15 ms	Yes	vFPGA offline
RISC-V Core Reboot Time	200–500 us	No	-
Packet Flow Recovery Time	≥100 us	Yes	Late trigger causes drops
Total Update Overhead	6–20 ms	Yes	Transient packet loss

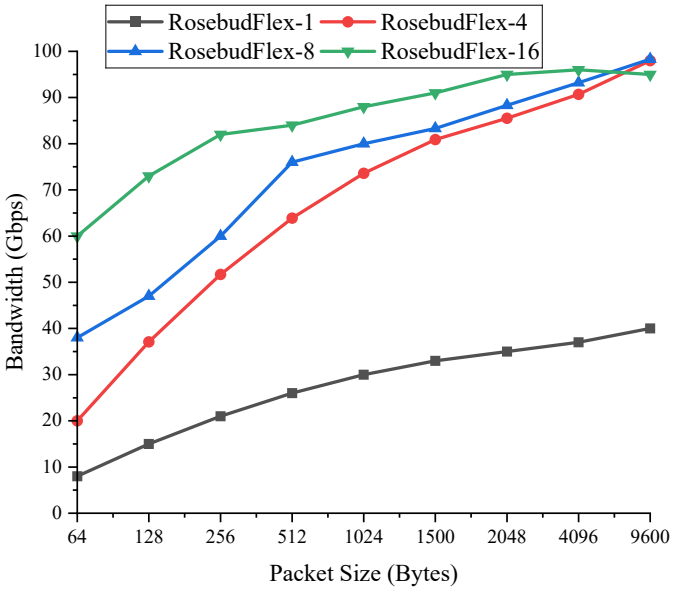

Figure 9: Throughput comparison under different numbers of tenants.

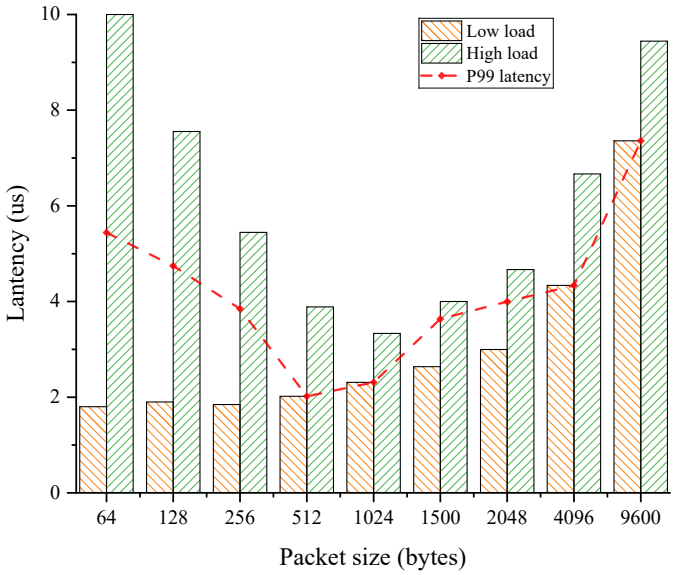

Figure 10: Forwarding Latency in RosebudFlex.

Flex for network function update, processing, and forwarding. We timestamp the packets and record the forwarding time after forwarding. The latter measures the time it takes from when the host sends a packet to RosebudFlex and then forwards it. This process also uses timestamps to record the arrival and forwarding times of the packet. Table 4 presents the packet latency across various payload sizes under both low and high network load conditions. A statistical analysis was conducted to evaluate the time required for loading different network functions. The key observations are as follows. The latency between the host receiving the update request and issuing the reconfiguration command has a negligible impact on overall service performance typically remaining below 1 us.

Bitstream transmission incurs significant overhead which is directly correlated with the size of the function binary being transferred. Larger binaries lead to increased transmission time and represent the primary bottleneck in dynamic updates. Prior to function replacement, it is necessary to flush the existing processing pipeline or back up in-flight data to prevent packet loss or logical inconsistencies. The partial reconfiguration process, which is intrinsically linked to the underlying FPGA architecture, contributes a fixed latency. Reinitialization of the RISC-V core is required to execute the updated logic. Data restoration is performed as the final step. Overall, the total reconfiguration latency ranges from approximately 6-20 ms primarily influenced by the bitstream size and the duration of the PR process.

Figure 10 shows the latency of packets with different sizes under low and high load conditions, along with the P99 latency measured under high load. We observe that the P99 latency under high load exhibits a U-shaped pattern, which aligns with the behavior of real-world high-load network scenarios. Under low load, latency increases linearly from 1.8 us to 7.6 us as packet size grows, a trend directly linked to increased buffer occupancy and DMA transfer time for larger packets. In contrast, high load introduces more pronounced latency overhead for smaller packets: frequent scheduling of small packets leads to higher transaction counts and context-switching costs, resulting in significantly elevated latency. For 64-byte packets, high-load latency is approximately five times higher than under low load, whereas for $MTU = 9600$, the latency increase is only about 25%.

6.3. Isolation Performance

To validate the effectiveness of the malicious tenant detection mechanism proposed in Section 4.7, we conducted a noisy neighbor experiment where Tenant A acted as a normal user while Tenant B behaved as an attacker.

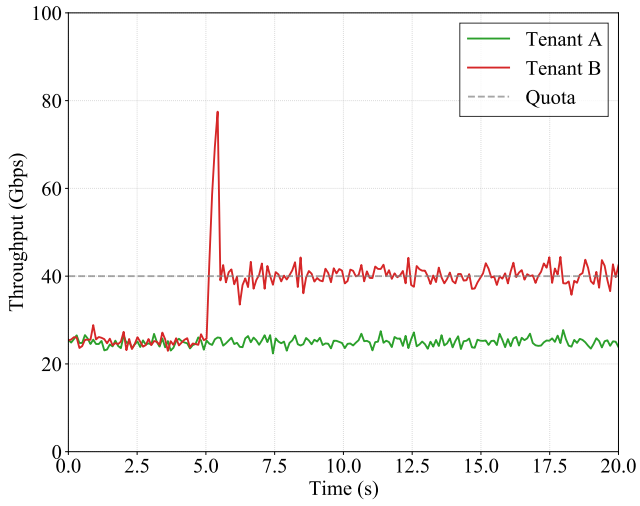

Figure 11: Performance isolation between tenants during a burst traffic attack.

As illustrated in Figure 11, both tenants initially operate with a stable throughput of approximately 25 Gbps. At $t = 5s$, Tenant B initiates a burst attack in an attempt to saturate the bandwidth. The Service Dispatcher detects this anomaly using the sliding window token bucket logic described in Algorithm 1 and triggers the traffic shaping mechanism at $t = 5.5s$. Consequently, the throughput of Tenant B is clamped to the preset quota of 40 Gbps represented by the dashed line.

Experimental results confirm that the throughput of Tenant A remains stable throughout the attack window, which demonstrates that RosebudFlex effectively enforces hardware level isolation and prevents resource contention in multi-tenant environments.

7. Conclusion

In this paper, we present RosebudFlex, a high-performance FPGA virtualization framework tailored for multi-tenant network acceleration. RosebudFlex introduces a hybrid abstraction of Private and Flexible Function Regions enabling dynamic sharing and isolation of network functions across tenants. By integrating partially reconfigurable regions, SR-IOV based vFPGA domains, and a runtime-managed service pool, our design achieves both resource elasticity and strong functional isolation. We analytically and empirically demonstrate that RosebudFlex significantly reduces function blocking probability and improves logic resource utilization. Analytical results indicate that flexible function region can reduce function blocking probability under the assumptions of uniform demand and service dispatcher controlled access. Experimental results show the potential of RosebudFlex as a foundation for FPGA as a Service (FaaS) infrastructures in multi-tenant cloud computing environments.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant No.2024YFB2907000, the Natural Science Foundation of Shandong Provincial under Grant No.ZR2022LZH015, the National Natural Science Foundation of China under Grant No.62402257, the Pilot Project for Integrated Innovation of Science, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) under Grant No.2025ZDZX01.

References

[1] T. Musale, A. Ganti, A. Gogoi, K. Manna, Low Power Network-on-Chip Architecture Design Technique, in: VLSI-SoC'24, IFIP/IEEE, Tanger, Morocco, 2024, pp. 1–6.

[2] W. Lin, Y. Shan, R. Kosta, A. Krishnamurthy, Y. Zhang, SuperNIC: An FPGA-based, cloud-oriented SmartNIC, in: FPGA'24, ACM, Monterey, USA, 2024, pp. 130–141.

[3] G. Dessouky, A.-R. Sadeghi, S. Zeitouni, SoK: Secure FPGA Multi-tenancy in the Cloud: Challenges and Opportunities, in: EuroS&P'21, IEEE, Vienna, Austria, 2021, pp. 487–506.

[4] M. H. Quraishi, E. B. Tavakoli, F. Ren, A Survey of System Architectures and Techniques for FPGA Virtualization, IEEE Transactions on Parallel and Distributed Systems 32 (9) (2021) 2216–2230.

[5] J. Ruan, Y. Chang, K. Zhang, K. Shi, M. Chen, Y. Bao, Increasing flexibility of cloud FPGA virtualization, in: FPL'22, IEEE, Belfast, UK, 2022, pp. 350–357.

[6] P. Miliadis, D. Theodoropoulos, D. Pnevmatikatos, N. Koziris, Architectural support for sharing, isolating and virtualizing FPGA resources, ACM Transactions on Architecture and Code Optimization 21 (2) (2024) 1–26.

[7] J. Mbongue, F. Hategekimana, D. Tchuinkou Kwadjo, D. Andrews, C. Bobda, FPGAVirt: A Novel Virtualization Framework for FPGAs in the Cloud, in: CLOUD'18, ACM, San Francisco, USA, 2018, pp. 862–865.

[8] S. Sunkavilli, Z. Zhang, Q. Yu, New security threats on fpgas: From fpga design tools perspective, in: ISVLSI'21, IEEE, Tampa, USA, 2021, pp. 278–283.

[9] Z. Wan, J. Zhang, Y. Wang, K. Liu, Y. Pan, T. Huang, Rethinking Congestion Management in Intra-Host Networks, IEEE Communications Magazine 63 (4) (2025) 184–191.

[10] J. Chaudhuri, K. Chakrabarty, Diagnosis of malicious bitstreams in cloud computing FPGAs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 42 (11) (2023) 3651–3664.

[11] L. Rosa, L. Foschini, A. Corradi, Empowering cloud computing with network acceleration: A survey, IEEE Communications Surveys & Tutorials 26 (4) (2024) 2729–2768.

[12] M. Khazraee, A. Forencich, G. C. Papen, A. C. Snoeren, A. Schulman, Rosebud: Making FPGA-Accelerated Middlebox Development More Pleasant, in: ASPLOS'23, ACM, Vancouver, Canada, 2023, p. 586–605.

[13] A. T. de Oliveira Filho, E. Freitas, P. R. do Carmo, D. F. Sadok, J. Kelner, Measuring the impact of SR-IOV and virtualization on packet round-trip time, Computer Communications 211 (2023) 193–215.

[14] J. Li, F. Yu, M. Ma, W. Liu, Y. Wang, H. Wu, H. He, RISC-V-Based GPGPU With Vector Capabilities for High-Performance Computing, VLSI'25 33 (8) (2025) 2239–2251.

[15] Y. Chang, Z. Guo, RosebudVirt: a high-performance and partially reconfigurable FPGA virtualization framework for multitenant networks, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 33 (1) (2025) 298–302.

[16] J. Mbongue, F. Hategekimana, D. T. Kwadjo, D. Andrews, C. Bobda, Fpgavirt: A novel virtualization framework for fpgas in the cloud, in: CLOUD'18, IEEE, San Francisco, USA, 2018, pp. 862–865.

[17] J. C. Vega, M. Ewais, A. Leon-Garcia, P. Chow, Ffive: An fpga framework for interactive vnf environments., in: FCCM'21, IEEE, Virtual Event, 2021, p. 263.

[18] Y. Xiao, A. Hota, D. Park, A. DeHon, Hipr: High-level partial reconfiguration for fast incremental fpga compilation, in: FPL'22, IEEE, Belfast, United Kingdom, 2022, pp. 70–78.

[19] M. Mandava, P. Reckamp, D. Chen, Nimblock: Scheduling for fine-grained fpga sharing through virtualization, in: ISCA'23, ACM, Orlando, USA, 2023, pp. 1–13.

[20] H. Li, Y. Dang, G. Sun, G. Liu, D. Shan, P. Zhang, {LemonNFV}: Consolidating heterogeneous network functions at line speed, in: NSDI'23, USENIX, Boston, USA, 2023, pp. 1451–1468.

[21] B. Seyoum, D. Giri, K.-L. Chiu, B. Natter, L. Carloni, Pr-esp: An open-source platform for design and programming of partially reconfigurable socs, in: DATE'23, IEEE, Antwerp, Belgium, 2023, pp. 1–6.

[22] Y. Zhou, M. Wilkening, J. Mickens, M. Yu, Smartnic security isolation in the cloud with s-nic, in: EuroSys'24, Athens, Greece, 2024, p. 851–869.

[23] A. Vaishnav, K. D. Pham, D. Koch, A survey on fpga virtualization, in: FPL'18, IEEE, Dublin, Ireland, 2018, pp. 131–1317.

[24] W. Chen, Z. Zhang, X. Zhang, Q. Shen, Y. Yarom, D. Genkin, C. Yan, Z. Wang, Hyperhammer: Breaking free from kvm-enforced isolation, in: ASPLOS'25, ACM, Rotterdam, Netherlands, 2025, p. 545–559.

[25] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, R. Tessier, The Future of FPGA Acceleration in Datacenters and the Cloud, ACM Transactions on Reconfigurable Technology and Systems 15 (3) (2022) 1–42.

[26] C. Wulf, M. Willig, D. Göhringer, A Survey on Hypervisor-Based Virtualization of Embedded Reconfigurable Systems, in: FPL'21, IEEE, Dresden, Germany, 2021, pp. 249–256.

[27] B. Seyoum, M. Pagani, A. Biondi, G. Buttazzo, Automating the design flow under dynamic partial reconfiguration for hardware-software co-design in FPGA SoC, in: SAC'21, ACM, Virtual Event, Korea, 2021, p. 481–490.

[28] A. Li, D. Wentzlaff, PRGA: An Open-Source FPGA Research and Prototyping Framework, in: FPGA'21, ACM, Virtual Event, USA, 2021, p. 127–137.

[29] K. Vipin, S. A. Fahmy, FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications, ACM Computing Surveys 51 (4) (2018) 1–39.

[30] G. Dessouky, A.-R. Sadeghi, S. Zeitouni, SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities, in: EuroS&P'21, IEEE, Vienna, Austria, 2021, pp. 487–506.

[31] A. Dörflinger, M. Albers, B. Kleinbeck, Y. Guan, H. Michalik, R. Klink, C. Blochwitz, A. Nechi, M. Berekovic, A comparative survey of open-source application-class RISC-V processor implementations, in: CF'21, ACM, Virtual Event, Italy, 2021, p. 12–20.

[32] S. Cirici, M. Paolino, D. Raho, SVFF: An Automated Framework for SR-IOV Virtual Function Management in FPGA Accelerated Virtualized Environments, in: CITS'23, Genoa, Italy, 2023, pp. 1–6.

[33] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, J. Van Dyken, The Stratix™ 10 Highly Pipelined FPGA Architecture, in: FPGA'16, ACM, Monterey, USA, 2016, p. 159–168.

[34] P. Zou, Z. Lin, X. Shi, Y. Wu, J. Chen, J. Yu, Y.-W. Chang, Time-Division Multiplexing Based System-Level FPGA Routing for Logic Verification, in: DAC'20, San Francisco, USA, 2020, pp. 1–6.

[35] S. Kasai, Y. Osana, A driver-based approach for DMA transfer between FPGA-GPU, in: CANDARW'22, Himeji, Japan, 2022, pp. 103–108.

[36] C. Johansson, An FPGA-based Ethernet Switch, in: TELFOR'21, Belgrade, Serbia, 2021, pp. 1–4.

[37] J. Ruan, Y. Chang, K. Zhang, K. Shi, M. Chen, Y. Bao, Increasing Flexibility of Cloud FPGA Virtualization, in: FPL'22, Belfast, United Kingdom, 2022, pp. 350–357.

[38] X. Chen, N. K. Jha, A 3-D CPU-FPGA-DRAM Hybrid Architecture for Low-Power Computation, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24 (5) (2016) 1649–1662.

[39] T. Fischer, M. Rogenmoser, T. Benz, F. K. Gürkaynak, L. Benini, FlooNoC: A 645-Gb/s/link 0.15-pJ/B/hop Open-Source NoC With Wide Physical Links and End-to-End AXI4 Parallel Multistream Support, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 33 (4) (2025) 1094–1107.

[40] G. Liu, J. Garside, S. Furber, L. A. Plana, D. Koch, Asynchronous interface FIFO design on FPGA for high-throughput NRZ synchronisation, in: FPL'17, Ghent, Belgium, 2017, pp. 1–8.

[41] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, O. Mutlu, PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM, ACM Transactions on Architecture and Code Optimization 20 (1) (2022).

[42] G. Baldini, I. Amerini, Online Distributed Denial of Service (DDoS) intrusion detection based on adaptive sliding window and morphological fractal dimension, Computer Networks 210 (2022) 108923.