一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2025-59,DC2025-131(2026-03)

# End-to-End Implementation of IOMMU-Based DMA Isolation for Trusted Execution Environments on RISC-V

Yanqing LIU†, Haocheng XU†, Hidetoshi URANAMI†, Akihiro SAIKI†, and Keiji KIMURA†

† Faculty of Science and Engineering, Waseda University
Ohkubo 3–4–1, Shinjuku-ku, Tokyo, 169–8555 Japan

E-mail: †yantsing@fuji.waseda.jp, ††xvhaocheng@akane.waseda.ac.jp, †††reinsirk@ruri.waseda.ac.jp,
††††saiki@kasahara.cs.waseda.ac.jp, †††††keiji@waseda.jp

**Abstract** Trusted Execution Environments (TEEs) require strict memory isolation, yet DMA-capable accelerators can bypass CPU-side protection mechanisms. IOMMUs have been used to protect against illegal DMA memory accesses by I/O devices. Recent TEEs support IOMMUs to ensure secure IO channels, while there remains considerable scope to investigate secure and efficient TEEs that integrate secure IO devices. However, there is no open-source research hardware platform to investigate it. This work presents an end-to-end implementation of IOMMU-based DMA isolation on a RISC-V platform, integrating a real accelerator (NVDLA) under Linux using the Chipyard framework. Evaluation using an FPGA demonstrates correct isolation, fault containment, and manageable overhead, highlighting the IOMMU as a practical foundation for RISC-V TEEs.

**Key words** Trusted Execution Environment, IOMMU, DMA isolation, RISC-V, SoC architecture

## 1. Introduction

Trusted Execution Environments (TEEs) provide isolation guarantees for sensitive code and data in the presence of untrusted system software and peripheral devices. Conventional TEE designs primarily enforce isolation on CPU-initiated memory accesses via privilege separation and page-based virtual memory protection. However, as well as server platforms, modern embedded and edge platforms increasingly rely on DMA-capable accelerators for performance and energy efficiency, including GPUs/NPUs and domain-specific AI accelerators [3], [17]. Once configured, a DMA controller can issue memory requests that bypass CPU-side checks and access physical memory directly, potentially violating TEE memory boundaries. In practice, insufficient DMA protection has enabled real-world attacks that undermine otherwise strong isolation guarantees [13].

IOMMUs address this gap by extending translation and permission checks to device-initiated accesses [2], [8], [16]. Recent TEEs, which cover IO devices, support IOMMUs to ensure secure IO channels [1], [7], [15]. However, these works have had limited real-world experience, and there is considerable scope for investigating TEEs that support secure IO devices.

While mature IOMMUs exist in proprietary ecosystems (e.g., Intel VT-d and Arm SMMU [2], [8]), end-to-end evaluations on open architectures such as RISC-V remain limited. The challenges are not purely architectural in accelerator-rich systems [6]: heterogeneous SoC interconnects (e.g., AXI peripherals integrated with TileLink-based memory systems), evolving specifications, and full-stack integration effort spanning RTL, firmware, kernel, and device drivers collectively increase deployment friction [12], [18], [19].

This work presents a deployable, Linux-enabled RISC-V IOMMU system-on-chip (SoC) implemented in Chipyard and evaluates whether IOMMU-enforced DMA isolation can serve as a practical foundation for TEE-oriented assumptions in accelerator-rich RISC-V SoCs. The evaluation emphasizes deployability, functional correctness, and incremental overhead under realistic system conditions, rather than maximizing standalone accelerator throughput.

### 1.1 Contributions

This paper makes the following contributions:

- **An end-to-end FPGA deployment of an open RISC-V IOMMU.** An open-source RISC-V IOMMU [20] RTL is integrated into a Chipyard-generated SoC and validated under Linux on an FPGA.
- **Full-stack Linux enablement for DMA isolation.** The IOMMU is connected to standard Linux IOMMU and DMA frameworks, enabling transparent IOVA-based DMA for a real accelerator.
- **Experimental evaluation using a DMA-capable AI accelerator.** Functional correctness, fault containment, and workload-dependent overhead are evaluated using NVDLA inference workloads.

## 2. Background and Related Work

### 2.1 TEE Isolation and the DMA Attack Surface

TEE mechanisms ensure a trusted program execution environment by isolating sensitive software components from the rich OS and third-party drivers. Representative designs include lightweight hypervisor-assisted isolation (e.g., TrustVisor) [14], enclave-style TEEs (e.g., Intel SGX) [5], as well as VM-based confidential computing approaches such as AMD SEV and Intel TDX [1], [7]. Beyond CPU-side threats, DMA-capable devices create a distinct attack surface because device accesses may bypass the CPU MMU entirely. Markettos et al. show that DMA-related gaps can lead to practical compromises even when CPU-side protections are in place [13].

### 2.2 IOMMUs in Proprietary vs. Open Ecosystems

Commercial platforms provide IOMMUs such as Intel VT-d and Arm SMMU, but these implementations are tied to specific SoCs, and software stacks are typically opaque to researchers [2], [8]. RISC-V standardizes an open IOMMU specification [16], enabling

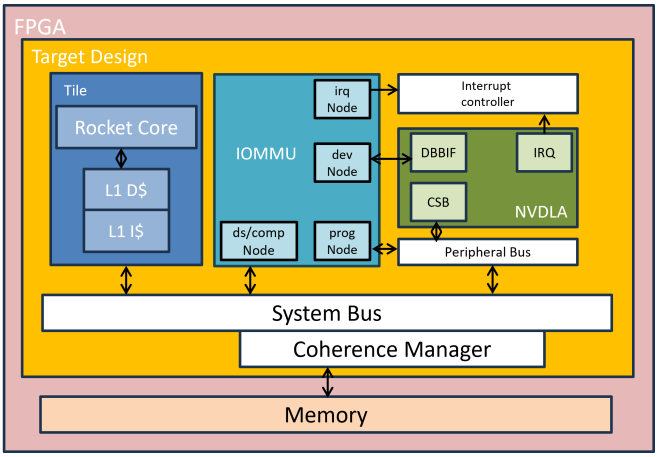


Fig. 1 Target FPGA-based heterogeneous RISC-V system integrating a CPU core, a RISC-V IOMMU, and an NVDLA accelerator.

transparent implementations and system-level integration studies. This paper builds on an open RTL implementation from ZeroDay-Labs [20] and evaluates its end-to-end behavior in a Linux-enabled deployment.

### 2.3 Accelerator-Rich SoCs and AI Inference Workloads

Hardware accelerators are increasingly used for deep learning inference and other domain-specific workloads [4], [9], [10]. While accelerators improve performance, their DMA engines introduce security and correctness risks if translation and isolation are not enforced at the system boundary. This motivates evaluating IOMMU mechanisms using realistic accelerator execution rather than purely synthetic microbenchmarks.

### 2.4 Chipyard, Diplomacy, and Linux Integration

Chipyard is a composable framework for generating RISC-V SoCs [18]. It uses Diplomacy for parameterized, type-safe hardware composition across a TileLink-based memory hierarchy [19]. In Linux-enabled deployments, DMA isolation relies on correct integration between device-tree descriptions, the Linux IOMMU subsystem, and the DMA mapping APIs [11], [12]. End-to-end evaluation, therefore, requires consistency across hardware interfaces, software bindings, and driver behavior.

## 3. System Overview

The target prototype presented in this paper is an FPGA-based heterogeneous RISC-V platform implemented in Chipyard. As shown in Fig. 1, the system integrates a RISC-V processor core, a shared memory subsystem, a RISC-V IOMMU, and a DMA-capable accelerator (NVDLA) within a unified SoC environment. The key design principle is that *all device-initiated memory accesses from the accelerator must traverse the IOMMU before reaching DRAM*.

To clarify system roles, we distinguish three paths that together form the end-to-end deployment:

- **Control/configuration path (CPU → IOMMU).** The CPU programs IOMMU state via MMIO (e.g., enabling translation, configuring page-table roots, and managing command/fault queues).
- **Device-to-memory data path (Device → IOMMU → DRAM).** All NVDLA DMA reads/writes are routed through the IOMMU, which enforces translation and permissions.
- **Fault/notification path (IOMMU → OS).** Translation failures and permission violations are reported via fault queues and interrupts, enabling softwarediagnosis and response.

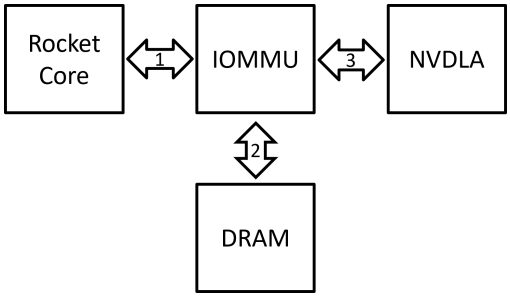


Fig. 2 Rocket–IOMMU–NVDLA–DRAM wiring example on the Chipyard-generated SoC. The control plane configures the IOMMU via MMIO, while the data plane enforces translation on accelerator DMA.

Fig. 2 illustrates this end-to-end wiring at the SoC level.

## 4. Proposed Approach and Implementation

### 4.1 End-to-End DMA Isolation Model

The proposed system places the IOMMU directly on the DMA path between accelerators and system DRAM, ensuring that all device-initiated memory accesses undergo translation and permission enforcement. Conceptually, each DMA request is expressed using an I/O virtual address (IOVA) in a device-visible address space. The IOMMU associates requests with a device identity and a translation domain (e.g., via device context and directory structures defined by the RISC-V IOMMU architecture [16]).

Upon receiving a DMA request, the IOMMU performs page-table-based address translation and access control checks. Only requests that map to valid physical addresses with appropriate permissions are forwarded to memory. Invalid or unauthorized accesses are blocked and reported via a fault-handling path that includes fault queues and interrupts delivered to the operating system. This mechanism allows software to detect, diagnose, and respond to isolation violations, providing a critical enforcement point for protecting TEE memory regions from untrusted or misconfigured peripherals.

### 4.2 IOMMU Microarchitecture and Mechanisms

The integrated IOMMU follows a queue- and IOTLB-based design. Fig. 3 shows the IOMMU BlackBox organization of the ZeroDayLabs IOMMU [20]. The major mechanisms relevant to end-to-end deployment are:

- **IOTLB.** A translation cache storing permission-tagged mappings for recently used IOVA pages. Hits avoid page-table walks and reduce latency on DMA hot paths.
- **Page-table walker.** On IOTLB misses, the IOMMU traverses device-directory and context structures, performing multi-level page walks (Sv39) to resolve translations.
- **Command/Fault queues.** Software submits control commands (e.g., invalidations) via a command queue (CQ), and the IOMMU reports faults via a fault queue (FQ) and interrupts.

### 4.3 Hardware Integration on Chipyard

The IOMMU RTL is integrated as a SystemVerilog BlackBox and composed through Chipyard's Diplomacy framework [18], [19]. The target SoC combines AXI-based peripherals/accelerators with a TileLink memory hierarchy; therefore, protocol bridges and buffering stages are inserted on both the data and control planes.

a ) Data-plane integration (AXI → TileLink).

The IOMMU master ports toward memory are bridged into the TileLink memory system via AXI–TileLink protocol conversion, as shown in Fig. 4. This ensures that device-initiated DMA traffic

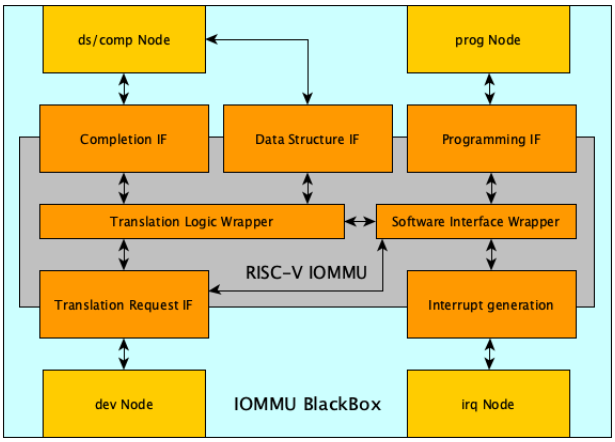


Fig. 3 Organization of the IOMMU BlackBox and its external interfaces. The IOMMU is encapsulated as a BlackBox module exposing four interfaces: a data/compute (ds/comp) interface for DMA traffic, a programming (prog) interface for CPU-side configuration, a device node (dev node) for identifying DMA-capable devices, and an interrupt (irq node) interface for fault and event notification. AXI–TileLink protocol bridges are used to integrate the AXI-based IOMMU with the TileLink-based Chipyard memory system.

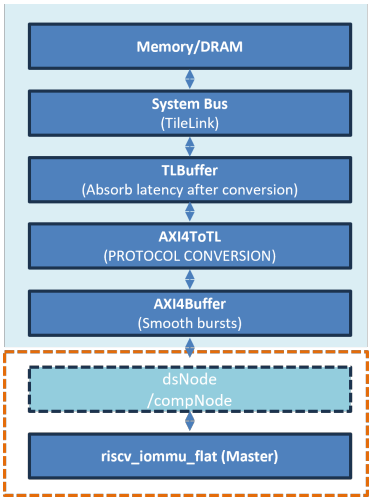


Fig. 4 IOMMU data-path connection to the TileLink memory system via AXI–TileLink protocol bridges.

is translated by the IOMMU and then correctly injected into the TileLink coherent memory system.

b ) Control-plane integration (TileLink → AXI).

The CPU MMIO programming interface is connected through TileLink toward the IOMMU register interface via TileLink–AXI conversion, as shown in Fig. 8. This allows Linux to program IOMMU registers and queue base pointers using standard MMIO access patterns.

### 4.4 Software Enablement and Linux Binding

On the software side, the system leverages the standard Linux DMA and IOMMU frameworks [11], [12] to manage device mappings and enforce isolation policies. Device discovery and IOMMU attachment are described through the device tree, enabling the OS to associate the accelerator with the appropriate IOMMU instance and to select IOVA-based DMA mapping.

During execution, the device driver allocates DMA buffers and establishes IOVA-to-physical mappings before invoking the accelerator. When buffers are released, the corresponding mappings are torn down, and IOMMU invalidation operations are issued to prevent stale translations from being reused. Kernel logs and debug traces confirm that mappings are created, consumed by the accelerator during execution, and later invalidated, demonstrating correct end-to-end coordination between software-managed mappings and hardware-enforced translation.

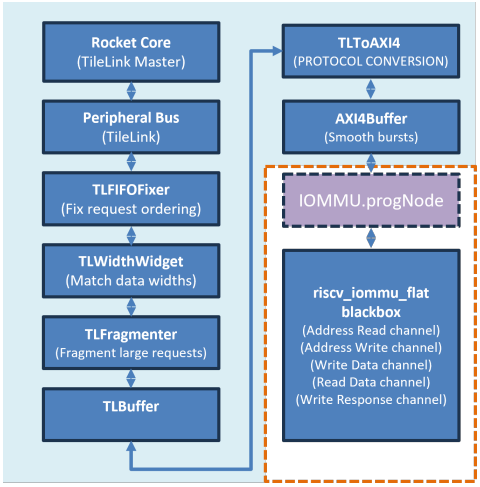


Fig. 5 Control-plane integration of the IOMMU programming interface with the CPU via TileLink.

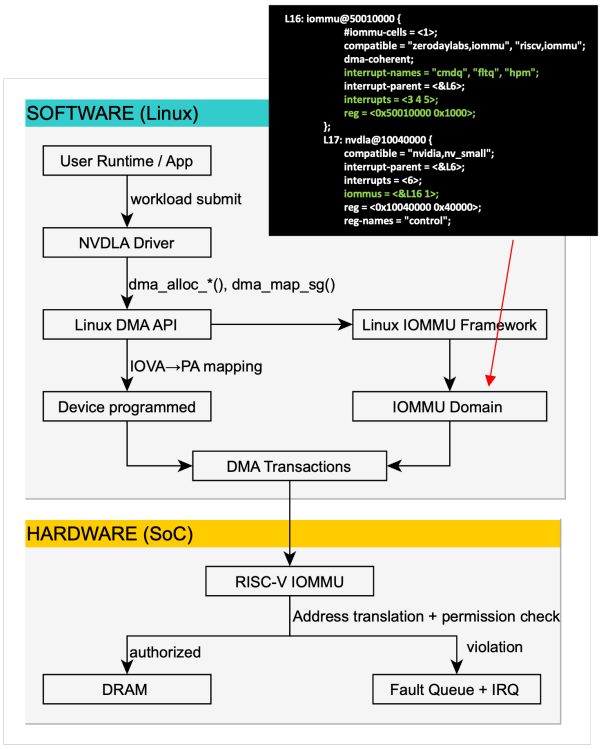


Fig. 6 Software enablement workflow from device discovery to DMA mapping and IOMMU invalidation.

### 4.5 Design Rationale for TEE-Oriented Isolation

While this paper does not introduce a new threat model, the system is explicitly designed to support a common TEE assumption: *TEE memory must remain inaccessible to untrusted peripherals and drivers*. The IOMMU enforcement point is therefore placed on the DMA boundary rather than within the accelerator, enabling:

- **Per-device isolation.** DMA permissions can be restricted to

Table 1 Target System Configuration

| Item | Description |
|---|---|
| CPU core | Single Rocket core |
| CPU clock frequency | 100 MHz |
| Memory interface clock | 100 MHz |
| Accelerator | NVDLA Small |
| IOMMU | ZeroDayLabs RISC-V IOMMU (RTL) [20] |
| FPGA board | Xilinx VCU118 |
| Toolchain | Vivado 2019.2 |
| Operating system | Linux (RISC-V) |

Table 2 Software Stack and RTL Versions

| Component | Version |
|---|---|
| Linux kernel | v6.15 (riscv/iommu next-version) |
| Boot firmware | OpenSBI v1.2 |
| Chipyard | v1.8.1 |
| IOMMU RTL | ZeroDayLabs RISC-V IOMMU (latest) |
| Accelerator software | NVDLA runtime |

Table 3 IOMMU Hardware Configuration Parameters

| Item | Description |
|---|---|
| Addressing | Sv39 IOVA, 4 KiB pages [16] |
| IOTLB | Hardware-managed, 4 entries |
| Command queue | 4096 entries |
| Fault queue | 8192 entries |
| Translation domain | DID-based device domain |

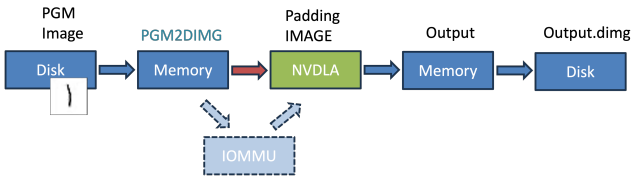


Fig. 7 NVDLA evaluation path from user-space runtime to DRAM through the Linux DMA mapping and the IOMMU.

only the buffers explicitly mapped for the accelerator.

- **Fault observability.** Misconfigurations or malicious DMA attempts surface as faults via the IOMMU's reporting path.
- **Compatibility with existing Linux stacks.** The approach aligns with existing IOMMU/DMA abstractions, improving deployability.

## 5. Evaluation

### 5.1 Experimental Setup

All experiments are performed on an FPGA-deployed RISC-V system generated using Chipyard. Table 1 summarizes the target system configuration, and Table 3 summarizes key IOMMU parameters. The evaluation path for NVDLA inference is shown in Fig. 7.

**5.1.1** Methodology.

We evaluate: (i) functional correctness of translation and isolation, (ii) fault containment and observability, and (iii) workload-dependent overhead. For overhead measurements, we run each workload with IOMMU disabled (baseline) and enabled (translation + permissions enforced).

Two representative inference workloads are selected for evaluation: *LeNet-mnist* and *ResNet50-small*. These workloads are chosen to expose different compute–data-movement characteristics on the accelerator. LeNet-mnist is a lightweight convolutional neural network with small input tensors and short execution phases, resulting in frequent DMA transfers and repeated buffer setup and teardown. In contrast, ResNet50-small is a deeper, more compute-intensive model, in which accelerator execution is dominated by computation, and DMA operations are amortized over longer processing intervals.

Both workloads are executed on the NVDLA accelerator using the standard NVDLA runtime under Linux. Prior to execution, the user-space runtime allocates input, output, and intermediate buffers through the Linux DMA mapping APIs. These buffers are mapped into an I/O virtual address (IOVA) space and accessed by the accelerator via DMA. Input preprocessing, including tensor layout preparation and padding, is performed by the runtime and driver stack, after which the accelerator fetches input data and network parameters exclusively through DMA transactions.

Time measurement is intentionally restricted to the DMA data path between system memory and the NVDLA accelerator to isolate the overhead introduced by IOMMU-enforced address translation and permission checks. Fault behavior is validated via kernel logs and IOMMU fault queue reporting during negative test cases.

To improve reproducibility and interpretability, measurements are collected only after the system has reached a steady state. The first run is treated as a warm-up and excluded to mitigate cold-start effects such as initial page-table walks and IOTLB coldness. All values reported in this section are obtained from steady-state runs under an identical system configuration.

### 5.2 Correctness and Isolation

Functional validation is performed using both positive and negative test cases.

The positive case preserves that the DMA accesses appropriately mapped pages. The accelerator is granted access only to explicitly mapped memory regions. Under this configuration, accelerator execution completes successfully without generating IOMMU faults, confirming correct translation, permission enforcement, and end-to-end DMA functionality.

The negative case entails intentionally omitting or invalidating DMA mappings. Under these conditions, the IOMMU detects translation failures, blocks unauthorized accesses, and reports faults through the fault queue and interrupt mechanisms.

We validate multiple representative misconfiguration classes: (i) **unmapped IOVA** (no mapping installed), (ii) **permission mismatch** (e.g., missing write permission), (iii) **domain/context mismatch** (e.g., incorrect DID or context format inconsistency), and (iv) **stale translation usage** after forced invalidation. From the user-visible perspective, such failures manifest as accelerator timeouts or aborted executions, consistent with stalled DMA controllers waiting on unresolved translations rather than errors in the compute pipeline. These results confirm that isolation violations are reliably detected and contained.

### 5.3 Overhead under Real Accelerator Workloads

Performance overhead is evaluated using NVDLA inference workloads as a representative example of realistic DMA-intensive accelerator usage. Measurements are performed with and without IOMMU enforcement to quantify incremental cost.

For the LeNet-mnist workload, enabling the IOMMU increases processing time from 3051.96 ms to 4323.50 ms (41.7% overhead). This increase is consistent with frequent short DMA transfers and repeated buffer setup and teardown operations, which amplify the impact of translation misses and invalidations along the DMA path.

In contrast, for the ResNet50-small workload, processing time increases from 80324.38 ms to 81165.29 ms (1.05% overhead). In this case, execution is dominated by computations, allowing translation overhead to be amortized across longer accelerator execution phases. Overall, these results indicate that IOMMU overhead is strongly workload-dependent and correlated with DMA access granularity and reuse patterns.

Table 4 NVDLA inference performance with and without IOMMU enforcement.

| Model | Data Size [B] | Input [ms] | Processing (w/o IOMMU) [ms] | Processing (w/ IOMMU) [ms] | Overhead [%] |
|---|---|---|---|---|---|
| LeNet-mnist | 626 | 16.32 | 3,051.96 | 4,323.50 | 41.7 |
| ResNet50-small | 1,3674 | 1,299.33 | 80,324.38 | 81,165.29 | 1.05 |

```
[    2.079537] printk: legacy console [ttySIF0] enabled
[    5.682112] riscv,iommu 50010000.iommu: forcing MSI_FLAT capabilit
y for DC layout
[    5.691870] riscv,iommu 50010000.iommu: using wire-signaled interr
upts
[    5.701332] riscv,iommu 50010000.iommu: queue #0 ready: entries=81
92 size=131072 bytes phys=0x0000000085480000 virt=(____ptrval____)
[    5.716124] riscv,iommu 50010000.iommu: queue #1 ready: entries=40
96 size=131072 bytes phys=0x00000000854a0000 virt=(____ptrval____)
[    5.786877] platform 10040000.nvdla: Adding to iommu group 0
[    5.793292] riscv,iommu 50010000.iommu: IODIR invalidate DID=0x1 t
c=0x0000000000000000 pasid=0
[    5.801587] riscv,iommu 50010000.iommu: IODIR update DID=0x1 tc=0x
0000000000000001 fsc=0x8000000000084859 ta=0x0000000000001000
[    6.021216] loop: module loaded
```

Fig. 8 The Linux boot and runtime logs show that the IOMMU is successfully initialized, command and fault queues are set up, and the NVDLA device is attached to an IOMMU domain.

### 5.4 Mechanism-Level Insights and Integration Lessons

The evaluated IOMMU implements a hardware-managed IOTLB using 4 KiB pages, domain-based translation, and permission-tagged entries. On an IOTLB miss, the IOMMU performs a page-table walk using device directory and device context structures before inserting the resolved translation into the IOTLB. Trace analysis confirms correct traversal of IOSATP, DDT, and device context structures, consistent with software-managed page tables [16].

The above observations are supported by a combined evidence chain from kernel logs (e.g., IOMMU attach/mapping events), fault queue reporting, and software-visible driver/kernel debug outputs collected during bring-up and evaluation.

This cross-layer evidence allows correlating misconfigurations with translation faults and confirming that observed accelerator failures are attributable to IOMMU-enforced translation and permission checks rather than errors in the compute pipeline.

A key integration lesson observed during bring-up is that mismatches between hardware-expected and software-programmed device context layouts can lead to persistent translation failures. In particular, disagreements regarding capability flags and extended context formats cause the IOMMU to interpret invalid control fields, resulting in translation faults and stalled DMA execution. Enforcing a consistent device context format across hardware and software resolves this issue and restores normal IOTLB fill and hit behavior.

Queue-based control mechanisms, including command and fault queues, are sufficient under the evaluated workloads. Command traffic is dominated by DMA map/unmap and invalidation activity, while fault queue activity remains negligible during correct configurations and appears primarily during intentional misconfiguration or early bring-up phases.

These observations indicate that IOMMU overhead on accelerator workloads is primarily determined by DMA access patterns and translation reuse, rather than raw data size alone. This reinforces the importance of mechanism-aware evaluation when assessing IOMMU deployment in accelerator-rich RISC-V systems.

### 5.5 Discussion and Limitations

This prototype demonstrates deployability and correctness for a real accelerator behind an IOMMU, but several limitations remain:

- **Single-accelerator evaluation.** We evaluate a single DMA device (NVDLA). Multi-device contention and scalability remain future work.
- **Limited workload set.** We report representative inference workloads that exhibit distinct DMA/compute balance; Broader models and batch sizes would improve generality.
- **FPGA constraints.** FPGA timing and memory subsystem characteristics may differ from ASIC deployments; nonetheless, the FPGA provides critical real-hardware validation not captured in simulation.
- **Interrupt/coherence scope.** The current study focuses on DMA translation and isolation semantics; interrupt mediation and multi-core contention effects (e.g., shared IOMMU resources and global invalidation pressure) are out of scope in the present single-core prototype. This is our future work.

## 6. Conclusion

This paper demonstrates a practical, end-to-end deployment of a RISC-V IOMMU on the Chipyard platform, spanning RTL integration, heterogeneous AXI/TileLink interconnect composition, Linux software enablement, and FPGA-based evaluation with a real DMA-capable accelerator. The results confirm that enforceable per-device DMA boundaries can be achieved across the hardware–software boundary and that the associated performance overhead is strongly workload-dependent.

By providing end-to-end measurements of RISC-V IOMMU enforcement cost under a Linux-driven AI accelerator workload, this study moves beyond microbenchmarks and isolated simulations. The system and integration lessons offer a reproducible reference for strengthening TEE-oriented memory isolation against DMA-capable peripherals on open RISC-V platforms.

**Acknowledgements** This work was supported by JST K Program Grant Number JPMJKP24U4, Japan.

**References**

[1] Advanced Micro Devices, Inc. Sev-tio: Trusted i/o for secure encrypted virtualization. `https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf`. Whitepaper, Accessed: 2026-02-10.

[2] Arm Ltd. Arm system memory management unit architecture specification (smmu). `https://developer.arm.com/documentation/ihi0062/latest`. Accessed: 2026-02-10.

[3] Krste Asanović et al. The landscape of parallel computing research: A view from berkeley. Technical report, EECS Department, University of California, Berkeley, 2010.

[4] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263, 2016.

[5] Victor Costan and Srinivas Devadas. Intel sgx explained. Technical Report 2016/086, IACR Cryptology ePrint Archive, 2016.

[6] John L. Hennessy and David A. Patterson. A taxonomy of accelerator architectures. *Communications of the ACM*, 62(9):54–64, 2019.

[7] Intel Corporation. Intel tdx connect architecture specification. `https://cdrdv2-public.intel.com/862706/Intel%20TDX%20Connect%20EAS%20354629%20004.pdf`. Document 354629-003-US, June 2025, Accessed: 2026-02-10.

[8] Intel Corporation. Intel virtualization technology for directed i/o (vt-d) architecture specification. `https://www.intel.com/content/www/us/en/architecture-and-technology/vt-directed-io-specification.html`. Accessed: 2026-01-21.

[9] Norman P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.

[10] Norman P. Jouppi et al. A domain-specific architecture for deep neural networks. *Communications of the ACM*, 61(9):50–59, 2018.

[11] Linux Kernel Developers. Dma api documentation. `https://www.kernel.org/doc/html/latest/core-api/dma-api.html`. Accessed: 2026-01-21.

[12] Linux Kernel Developers. Iommu subsystem documentation. `https://www.kernel.org/doc/html/latest/driver-api/iommu.html`. Accessed: 2026-01-21.

[13] A. T. Markettos et al. Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2019.

[14] Jonathan M. McCune et al. Trustvisor: Efficient tcb reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[15] RISC-V AP-TEE-IO Task Group. Cove-io v0.2.0 draft specification. `https://github.com/riscv-non-isa/riscv-ap-tee-io/releases/tag/v0.2.0`. Accessed: 2026-02-10.

[16] RISC-V International. Risc-v iommu specification. `https://github.com/riscv-non-isa/riscv-iommu`. Accessed: 2026-01-21.

[17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[18] UC Berkeley. Chipyard: A framework for composable risc-v systems. `https://github.com/ucb-bar/chipyard`. Accessed: 2026-01-21.

[19] UC Berkeley. Diplomacy: A parameterized interconnect generator for chisel/chipyard. `https://chipyard.readthedocs.io`. Accessed: 2026-01-21.

[20] ZeroDayLabs. Open-source risc-v iommu implementation. `https://github.com/zero-day-labs/riscv-iommu`. Accessed: 2026-01-21.