# XRAY: Detecting and Exploiting Vulnerabilities in Arm AXI Interconnects

Melisande Zonta
*ETH Zurich*

Nora Hinderling
*ETH Zurich*

Shweta Shinde
*ETH Zurich*

***Abstract*—The Arm AMBA Advanced eXtensible Interface (AXI) interconnect is a critical IP in FPGA-based designs. While AXI and interconnect designs are primarily optimized for performance, their security requires closer investigation—any bugs in these components can potentially compromise critical IPs like processing systems and memory. To this end, XRAY systematically analyzes AXI interconnects. Specifically, it treats the AXI interconnect as a transaction processing block that is expected to adhere to certain properties (e.g., bus and data isolation, progress). Then, XRAY employs a traffic generator that creates transaction workloads with the aim of triggering violations in the AXI interconnects. As the last piece of the puzzle, XRAY checkers automatically flag transaction traces as either compliant, errors, or warnings. Put together, XRAY comprises 13 properties, has been tested on 7 interconnects and identifies 41 violations corresponding to 41 vulnerabilities. When compared to existing approaches such as verification IPs (VIPs) and protocol checkers from commercial tools, XRAY identifies 19 known and 22 new violations. We show the security impact of XRAY by sampling 5 XRAY violations to construct 3 proof-of-concept exploits on realistic scenarios deployed on FPGA to leak intermediate data, drop transactions, and corrupt memory.**

***Index Terms*—AXI, interconnect, security, vulnerability**

## I. INTRODUCTION

The Arm AMBA AXI (Advanced eXtensible Interface) interconnect [1] is the communication backbone between components (e.g., processors, memory, and peripheral devices). Interconnects manage data flow efficiently by arbitration, bandwidth distribution, and protection against stalling or protocol violations. However, while these features are essential for performance, the security of interconnects has received relatively less attention. Most research and development efforts have concentrated on optimizing stalling and bandwidth distribution [2], with some industrial IPs, such as firewalls [3] from AMD Xilinx or VIPs from Cadence [4], [5] or AMD Xilinx [6], developed to safeguard against timeouts and some fatal protocol violations. A systematic exploration of interconnect security—addressing issues such as bus sharing, memory corruption, and other potential vulnerabilities has not been investigated.

To this end, we present XRAY which analyzes the security of AXI-based interconnects. Our main insight is to first identify security properties that are critical to interconnects—data invalidation, transaction ordering, concurrency, and integrity of transaction IDs and bursts. We show that in the absence of these properties, malicious IPs that are AXI compliant but aim to exploit weaknesses in the interconnects and other IPs can bring about security exploits (e.g., leaking data, tampering writes, denial-of-service). Formulating the properties of an abstract interconnect allows XRAY to: (a) generate AXI traffic that can potentially uncover vulnerabilities by intentionally modifying transactions; and (b) synthesize a property checker which identifies if the interconnect was able to handle the malicious traffic correctly or not. We build XRAY based on these principles and then address design decisions such as traffic granularity, traffic mutation strategies and synthesis of checkers for different interconnects. Put together, XRAY is a tool that can operate on any given interconnect to identify security vulnerabilities.

To validate our approach, we select seven interconnects, including most commonly used from AMD Xilinx [7], [8] and Pulp [9] as well as non-commercial implementations [10]. We rigorously tested them using the XRAY tool. We uncovered multiple vulnerabilities. Furthermore, we exploited some of the vulnerabilities to corrupt the output of a classification task ran on an AWS F1 instance. This enabled us to bypass all protocol checkers and firewalls, ultimately demonstrating a memory corruption attack on the host side.

**Contributions.** Our main contributions are XRAY—a tool to analyze interconnect security. Our novel approach introduces security properties which guide XRAY to generate transaction traffic along with a checker that detects property violations. In our evaluation of 7 interconnects, XRAY detects 41 vulnerabilities. We use 5 vulnerabilities to show-case 3 case-studies that demonstrate security impact. We responsibly disclosed all our findings to AMD Xilinx and PULP in September 2024. XRAY is available at https://axi-security.github.io/xray.

## II. BACKGROUND

### A. AXI Overview

AXI defines a manager-subordinate interface for concurrent, bi-directional data exchanges [1]. Each AXI interface has five independent channels: Address Read (AR), Address Write (AW), Data Read (R), Data Write (W), and Write Response (B) channel. AXI supports both read and write transactions. Managers initiate transaction requests, directing them to the appropriate subordinates. The target subordinate processes these requests, either providing the requested data (read) or accepting the data and issuing an acknowledgment (write). Managers can issue multiple outstanding transaction requests meaning it allows managers to send multiple transaction requests without waiting for earlier ones to complete. AXI supports out-of-order transaction completion. The sequence of transactions is tracked using an ID signal, which ensures that transactions with the

978-3-9826741-0-0/DATE25/© 2025 EDAA

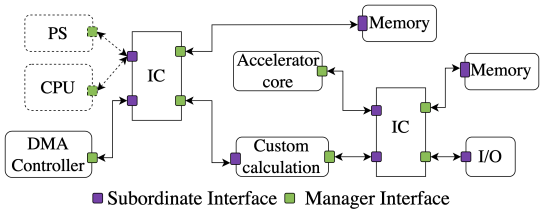


Fig. 1: Example of FPGA design with interconnects.

same ID are completed in order, while those with different IDs can be processed out of order. This adds flexibility but also introduces complexity to the interconnect and the connected subordinates, as maintaining the correct order of transactions is challenging. AXI operates in two modes, AXI-lite and AXI-full. AXI-full provides all AXI features including multiple outstanding transactions, read interleaving, and burst transfers. AXI-lite supports only single transactions where each request transfers a single data word. In AXI-full, burst transactions allow a single address request to read or write up to 256 data words, significantly increasing data throughput.

### *B. Interconnects*

AXI-based architecture is designed to facilitate efficient communication between various IP cores through a structured hierarchy of managers, an interconnect, and secondaries. Fig. 1 shows commonly used managers and subordinates. Interconnect acts as a communication hub, connecting multiple managers to multiple subordinates. The interconnect has subordinate interfaces for each controller and manager interfaces for each peripheral, connecting controllers to subordinate interfaces and peripherals to manager interfaces.

**Interconnect Capabilities.** The interconnect offers protocol compliance, data routing, arbitration, and isolation using a structure composed of arbiters, routers, and buffers. Fig. 2 shows a generalized view of AXI interconnects [11]. Interconnects separate requests from acknowledgments. It places read and write addresses, along with the data, in one buffer and stores write acknowledgments and read data in another. Buffers temporarily hold transactions, such as a write request from M1 or a read response from S2, to manage congestion or when the receiving component is not ready. This ensures smooth data flow and handles differences in processing speed between components. The arbiter decides the order in which multiple managers (e.g., M1 or M2) access a shared resource, such as a subordinate memory, prioritizing transactions based on predefined rules such as round-robin or fixed-priority. After the arbiter selects a transaction, the router forwards it to the correct destination (S1 or S2) by analyzing the transaction's address. Likewise, it routes acknowledgments back to the manager, using the transaction ID (if available) to identify its destination. The interconnects ensure compatibility between managers and subordinates by handling protocol conversions when needed (e.g., from AXI-lite manager to AXI-full subordinate). The interconnects support burst transfers with variable transaction lengths, burst types, and data width.

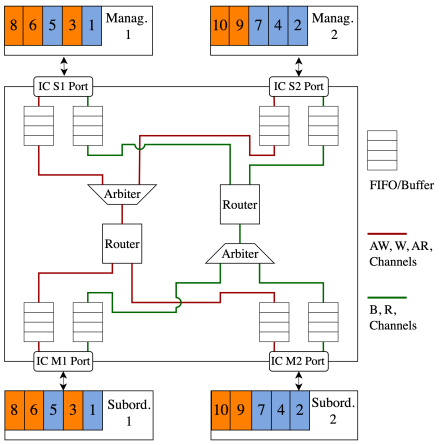


Fig. 2: Interconnect structure.

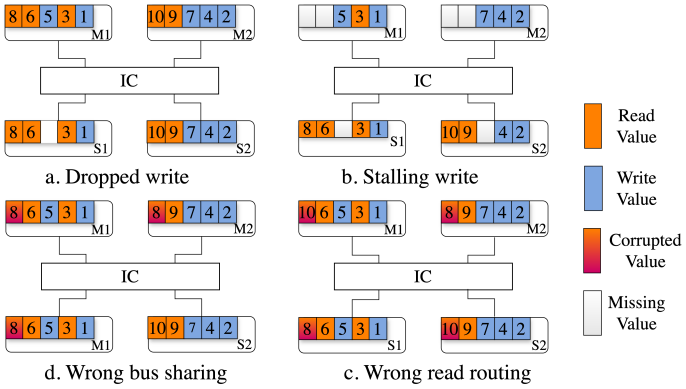


Fig. 3: Examples of interconnect behavior.

## III. Motivation

**Threat Model.** In a system with $m$ managers and $n$ subordinates connected via an interconnect, we assume at least one subordinate is not AXI compliant. The interconnect, implemented in Verilog, SystemVerilog, or VHDL, is expected to route transactions between potentially non-compliant IPs while maintaining its own AXI compliance. However, it may have functional issues, such as mishandling transactions, inadequate isolation, or failure to recover from protocol violations. These vulnerabilities could expose the system to risks like data leakage, corruption, or denial-of-service attacks. Concretely, we assume one of the managers, while being AXI compliant, is completely controlled by the adversary and aims to mount attacks on the interconnect, subordinates, and other IPs.

We take a concrete AXI interconnect and explain how it can deviate from the expected behavior. Consider a system with two subordinates, of which one may not be AXI-compliant, connected to two managers via the AMD Xilinx AXI interconnect [7]. One of the managers could potentially be malicious, with the ability to access sensitive data of the other manager or transmit incorrect information to a subordinate. Fig. 2 illustrates two managers, M1 and M2, successfully completing read and write transactions, with values routed correctly and processed

sequentially. In this scenario, M1 serves as the victim, while M2 could be malicious, and subordinates S1 and S2 might be faulty. Next, we present example scenarios to demonstrate potential security vulnerabilities M1 can exploit.

**Dropped Write.** In Fig. 3a., S1 does not sample the address or data due to an incorrect timing in transaction ordering. When used with M1, this causes a write transaction from M1 to S1 to be dropped, thus not writing value 5 to S1. It is the interconnect's job to prevent such faulty implementations of S1 from impacting correctness of transactions. Specifically, the interconnect's buffering and ordering mechanism should have prevented the drop.

**Stalling Write.** Fig. 3b. shows M1 and M2 operating normally until a write transaction from M1 stalls while processing the value 5. This intentional stall by M1 subsequently blocks M2 from making progress after writing the value 4 to S2 resulting in a denial of service (DoS). The subordinate fails to time out after not receiving an acknowledgment. The interconnect's arbitration mechanism should have prevented this.

**Wrong Read Routing.** Fig. 3c. shows that the read transactions values 8 and 10 get wrongly routed between M1 and M2. S1 and S2 do not correctly support the transaction IDs, and the router does not verify if the returned ID matches an outstanding transaction, allowing out-of-order processing but without confirming if ACKs belong to the correct transactions.

**Wrong Bus Sharing.** In Fig. 3d., M1 initiates the final read transaction, the read data 8 is not only returned to M1 but also appears on M2's read data bus, indicating unintended sharing. A failure from S1 to clear data after the transaction completes results in sharing. The interconnect's routing and isolation mechanisms should have prevented the sharing.

As an example, we analyze here the root cause of the bus sharing due to the interconnect, specifically the interface signals as shown in Fig. 3d.–4. In this simplified example, M1 sends `data` to the address `addr` of S1. While it is normal to see the address and data on the interface connected to S1, it is unexpected to see `data` appear on the interface connected to M2. Interestingly, we observe that while $ic\_m1\_rdata\_i$ fails to clear its bus when the signal $ic\_m1\_rvalid\_i$ goes low at t=7, proper clearing does occur for $ic\_s1\_rdata\_o$ at t=9. This indicates an initial issue with the S1 interface failing to clear data when it becomes invalid, although the interconnect attempts to resolve the problem on the output side. The complication arises when the data also appears on the $ic\_s2\_rdata\_o$ bus from t=8 to t=9, indicating unintended bus sharing which could lead to the leakage of sensitive data from M1 to M2. While S1 might be expected to clear the data, it is ultimately the interconnect's responsibility to make sure that such bus sharing does not occur. In fact, interconnects like AMD Xilinx SmartConnect [8] and PULP crossbar [12] handle this correctly by clearing the data once it is not in use.

## IV. XRAY OVERVIEW

We present a systematic approach to detect the above described types of vulnerabilities that can arise in interconnects using XRAY. Given an interconnect, XRAY first analyzes it to capture representative traffic. Then it mutates the traffic to simulate a wide range of realistic scenarios and executes the interconnect under this traffic. Lastly, XRAY defines oracles to detect wrong behaviors. XRAY represents the oracles as assertion checks and places them between the managers, subordinates, and interconnect. The checkers determine whether each output trace is benign or indicates a potential violation.

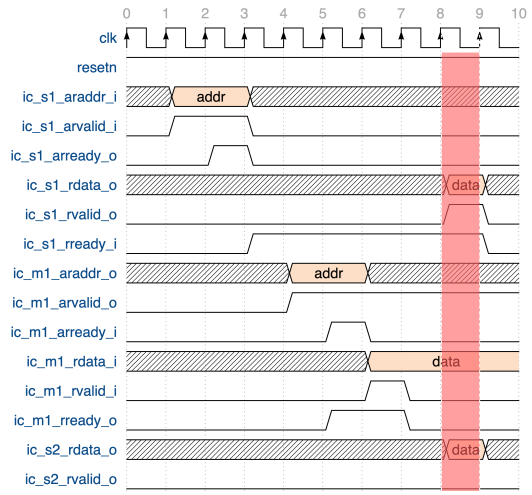


Fig. 4: **Bus sharing example.** The interface naming refers to Fig.2.

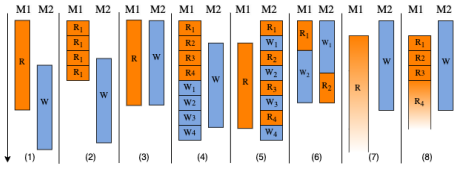


Fig. 5: **Example of traffic from two managers.** Scenario 1 shows manager 1(M1) initiating a read transaction while manager 2(M2), slightly delayed, performs a write. In Scenario 2, M1 executes a burst read followed by M1's write transaction. Scenario 3 illustrates concurrent read and write transactions from M1 and M2, respectively. Scenarios 4 and 5 depict rapid read or write transactions (spamming) from M1 or M2, while the other manager carries out normal transactions. Scenario 6 explores out-of-order transactions. Scenario 7 involves a stalled read from M1 blocking M2's transactions, and Scenario 8 shows a stalled burst read from M1 similarly blocking M2. These scenarios can be reversed between managers and applied to different transaction types.

XRAY needs to make three key decisions to realize the above approach. The first decision is at what granularity to operate. AXI protocol defines transactions, channels, and signals. XRAY's threat model treats the interconnect as a black box, with managers controlling only transactions, so it focuses on transactions exclusively. The second decision is how to mutate the traffic. As shown in Fig. 5, traffic consists of sequences of read and write transactions with varying timings. XRAY focuses on key AXI protocol features, adjusting parameters like which manager initiates transactions, transaction types (read, write), and modes (normal or burst). Finally, XRAY needs oracles to identify vulnerabilities in the output traces generated by the mutated traffic. To achieve this, we define the properties to capture the expected behavior of an interconnect.

**Existing Properties.** We reviewed both industrial Verification IPs (VIPs) from sources like Xilinx and Arm, and non-commercial tools (e.g., eXpect [13], Gisselquist [10]). We used

| P# | Category | Sub Category | Transaction | Property |
|---|---|---|---|---|
| 1 | Reset | - | R/W | Valid signals should be cleared upon reset |
| 2 | Data Invalidation | Reset | R/W | Address and data should be cleared upon reset |
| 3 | | Data Clearance | R/W | Values should be cleared after usage |
| 4 | | Data Isolation | R/W | Data of one manager shouldn't be present on another manager bus |
| 5 | Ordering | Data Ordering | R | The data should follow the associated address |
| 6 | | Data Timeout | R | The transaction should be cancelled if the ack does not follow the address |
| 7 | | Data/Address Timeout | W | Data and address are active at the same time |
| 8 | | Ack Ordering | W | Ack should come after address and data have been transmitted |
| 9 | | Ack Timeout | W | The transaction should be cancelled if the ack does not follow the data |
| 10 | Concurrency | - | R/W | A read and a write transaction are issued at the same time |
| 11 | Burst Transfer | Bursts Number | R/W | The number of burst should map the issued length |
| 12 | | Last Burst | R/W | The last signal should be issued only at the last burst transfer |
| 13 | ID | - | W | The acknowledgment ID should match the request ID |

TABLE I: XRAY **Properties.** Detailed description of our properties.

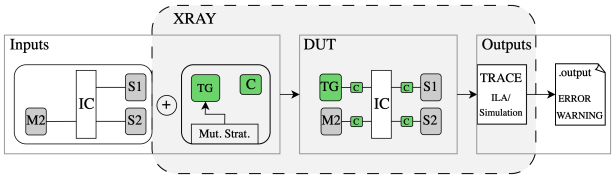


Fig. 6: XRAY **Tool.** XRAY takes as input the interconnect under analysis, along with M2, S1, and S2. XRAY has two components: a traffic generator (TG) with mutation strategies and a checker (C) connected to each interconnect interface. The design under test (DUT) runs in simulation or on FPGA, where XRAY generates a report of warnings and errors for each trace.

| P# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VIP/PC | [✓] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | [✓] | ✓ |
| Severity | E | E | E | E | W | E | W | W | E | W | E | W | E |
| LoC | 14 | 17 | 17 | 16 | 70 | 50 | 50 | 70 | 50 | 20 | 20 | 19 | 19 |

TABLE II: **Property Overlap.** ✓ indicates others VIP/PC cover the property, ✗ indicates a new property, and [✓] shows partial coverage. E flags security concerns, while W indicates concerning behaviors. LoC represents the number of lines of code needed to implement each distinct property (e.g., property 2 is replicated 6 times, but the count reflects only a single instance).

the properties from these works as a base to capture problematic interconnect behaviors (see Tab. I and Tab. II). However, existing set of properties are not sufficient to detect interconnect violations, which lead to issues outlined in Section III.

XRAY **Properties.** XRAY introduces new properties not covered by existing VIPs which are specific to interconnects, as shown in Tab. I. We summarize them below:

*Reset*: System signals must be cleared after a reset.

*Data Invalidation*: Outdated or unnecessary values must be cleared after use, preventing persistent data from causing errors. This invalidation could be triggered at different stages of transactions, both during reset and normal operation.

*Ordering*: The sequence of requests (address and data) and acknowledgments must be correct to ensure that transactions occur in the expected order. This is critical for data integrity and synchronization. XRAY includes timeout properties to ensure that transactions are discarded if they stalled for too long, preventing system hangs.

*Concurrency*: Simultaneous issuance of read and write transactions must preserve transaction completion and data isolation.

*Burst Transfer*: Data bursts must be properly handled, and completed without loss.

*ID*: Acknowledgment IDs must match the corresponding request IDs. This is vital for tracking and validating transactions within the system. Mishandling the IDs can lead to improper routing of transactions.

## V. XRAY TOOL

We build two XRAY components as summarized in Fig. 6.

### A. Traffic Generator

XRAY generates AXI traffic and monitors the interconnect's response across various scenarios, including reset, read, write, concurrent transactions, out-of-order operations, stalling, and spamming. It varies the transaction types, address/data widths, and increment steps. In full mode, it also changes burst types and lengths. For concurrency, XRAY modifies in addition the interconnect's address range. For spamming, it also tests buffer handling by varying buffer length. For out-of-order scenarios in full mode, it involves adjusting all of the above parameters for in-flight transactions, while in lite mode, XRAY tests outstanding transactions. For stalling, it blocks certain signals at various transaction stages, including the final burst.

### B. XRAY Checker

Translating the safety properties from Tab. I into synthesizable SystemVerilog assertions often requires a transformation into more complex concrete units like control logic or state machines. Most can be handled with simple if-else conditions, but as shown in Tab. II, the ordering properties require more complex implementations due to the temporal logic. Tracking out-of-order transactions, especially when enforcing ordering constraints, necessitates introducing various internal data structures. XRAY implements a lite and full mode version of its checker supporting both modes of the connected interface.

## VI. EVALUATION

### A. Setup

We selected a diverse set of 7 interconnects ranging from industrial IPs provided by Xilinx such as Smartconnect (XS) and Interconnect (XI) and by open-source platforms such as Pulp (PU) and Gisselquist (GC). Tab. III outlines the features that each interconnect advertises in their documentation. Most interconnects promote their protocol compliance, burst transfer capabilities, and arbitration features. However, none mention handling timeouts standalone. A setup can comprise IPs which use different AXI modes, and interconnects ensure they can operate with each other. For the interconnects that support this auto-configuration, we tested all possible combinations of manager, interconnect, and subordinate in lite and full modes.

We consider a setup with two managers and subordinates (M1/S1 and M2/S2), allowing M1 and M2 to connect to either subordinate one at a time, enabling XRAY to detect routing, isolation, or interconnect issues. M1 is XRAY's traffic generator. Both M2 and the subordinates (S1, S2) use the same AXI implementations for a fair comparison, extracted

| Interconnects /Features | AXI Compliance | Burst Transfer | Buffer Config | Arbit-ration | Security Isolation | Time Out | Protocol Bridging | Auto Config |
|---|---|---|---|---|---|---|---|---|
| Pulp Xbar (PU) [12] | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Gisselquist Xbar (GC) [14] | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Xilinx Interconnect (XI) [7] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Xilinx Smartconnect (XS) [8] | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ |

TABLE III: **Interconnect Features.** Combines both the lite and full versions.

via Vivado [15], [16]. All are written in Verilog/SystemVerilog, with the subordinate emulating a small memory using at least four built-in registers. Gisselquist [10] and eXpect [13] demonstrated that the AMD Xilinx subordinate implementation is not AXI-compliant although it is commercially used in hardware designs hence this selection for our test setup. Following the mutation strategy in Section IV, we simulate 37844 test cases for interconnect for lite and 171828 for interconnect for full. We parallelized the experiment for the 7 interconnects which took about 10 days. We selected error-generating test cases, shown in Tab. IV, and executed them on a VCU118 [17]. For the concurrency scenario, we also tested on a Zynq ZCU102 [18] to simulate a cloud environment using Vitis [19], where the processing system (PS) represents the host VM. Finally, after the local cloud simulation, we ran one test on AWS F1. We assess the impact of the deployed checkers on FPGA area and power. The lite checker uses 94 LUTs, 42 Flip-flops and consumes 0.210W while the Full checker uses 395 LUTs, 517 Flip-flops and 0.628W. Lastly, we compare XRAY with VIPs and PC from AMD Xilinx [5], [20] and Cadence [5].

### B. Summary of Vulnerabilities

Tab. IV shows the results for each of the 7 interconnects.

**Vulnerability Types.** XRAY identifies 9 types of vulnerabilities, of which DoS, bus sharing, and incorrect routing were explained in Section III. Additionally, it detects two types of memory corruption vulnerabilities: (a) corruption during the final write burst transfer, leading to an extra write when the indicator of the last burst transfer (`wlast`) signal is not properly raised, (b) corruption when the interconnect issues concurrent read and write transactions. Then, buffer overflow is caused by improper handling of internal buffers within the interconnect. Lastly, read data leakage occurs when the bus is not cleared after the data becomes invalid.

**Detected Vulnerabilities.** XRAY identifies 41 total vulnerabilities. Due to space limitations, we will highlight only a few notable results from Tab. IV. Overall, $XI_F$ is vulnerable to all identified issues, while XS has the fewest vulnerabilities, with $PU_F$ falling in between. This is probably because XI offers the highest level of user configurability, making it more prone to incorrect implementations or conflicting requirements, whereas XS manages many aspects automatically, reducing errors. All interconnects are vulnerable to DoS attacks due to the lack of internal timeout mechanisms, despite the arbiter's role in preventing resource monopolization. However, the propagation of DoS from a manager to a subordinate can be mitigated, as seen in $XI_L$'s default setting, which permits only one transaction at a time. XS is not vulnerable to last burst vulnerabilities, likely due to its built-in logic that tracks and ensures the correct number of burst transfers. All interconnects operating in full mode are vulnerable to memory corruption caused by

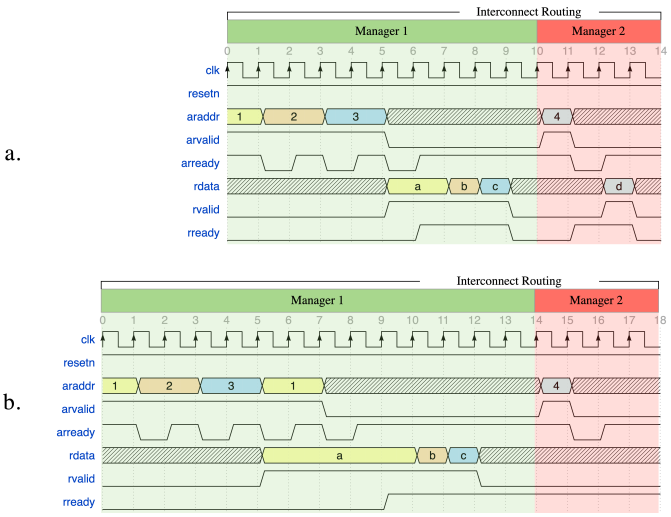


Fig. 7: **Buffer Overflow.** a. Depicts normal scenario b. Illustrates the vulnerability. We show each manager's perspective with $GC_L$ as interconnect. For brevity, the diagram shows an example with a buffer length of 3.

concurrent read and write operations, a non-deterministic issue that is difficult to resolve without adding significant logic overhead. 12 vulnerabilities are tied to specific unexpected parameter settings. For example, GC's `Low_Power_Mode`, when activated, prevents issues like bus sharing and data leakage. In a similar way, for $XI_L$ and $XI_F$, enabling the `Data_Fifo` parameter in the configuration prevented these vulnerabilities. Additionally, 8 vulnerabilities in $XI_F$ and XS are linked to specific manager/subordinate implementations configurations. For instance, when the manager is in full mode and the subordinate is in lite mode, protocol conversion by the interconnect led to protocol bridging issues in 5 cases.

**Comparison to VIP/PC.** As summarized in Tab. IV, of the 41 vulnerabilities, 19 are detected by VIPs, while 22 go undetected because of the lack of properties. Although VIPs include timeout properties that flag all 16 DoS issues, they cannot fully evaluate arbiter functionality since they only operate between a single manager and subordinate interface. Interestingly, VIPs can still detect some cases, like buffer overflows, where a dropped transaction could lead to a DoS, triggering timeout mechanisms.

### C. Exploiting Vulnerabilities

Of the 41 vulnerabilities detected by XRAY, we picked 5 and present 3 end-to-end exploits. For the remaining 35 vulnerabilities, we confirmed that XRAY reported vulnerabilities are true positives and unique (i.e. no duplicates) by re-executing the setup with the corresponding traffic and manual analysis.

**Buffer Overflow.** M1 and M2 are connected to S1 acting as memory via $GC_L$. M2 is an additional IP, while M1 is a monitor IP regularly polling the memory and potentially untrusted. In this case M1 triggers outstanding transactions. Fig. 7a shows M1 making 3 read requests to addresses `1`, `2`, `3` at t=0, 1, 3, with the subordinate responding by sending values `a`, `b`, and `c` at t=5, 7, 8. At t=10, M2 requests a read from address 4, receiving the value `d` at t=12. Fig. 7b illustrates M1 sending

| Interconnects/ Vulnerabilities | DoS Read | | DoS Write | | DoS Write Last Burst | | Memory Corruption Last Burst | | Wrong routing ack | | Unallowed Bus Sharing | | Read Data Leakage | | Buffer Overflow | | Memory corruption concurrency | | Total # Vuln. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pulp lite ($PU_L$) | ⑥ | ✓ | ⑨ | ✓ | | | | | ④ | ✗ | ④ | ✗ | ④ | ✗ | ⑤ | ✗ | No | | 6 |
| Pulp full ($PU_F$) | ⑥ | ✓ | ⑨ | ✓ | ⑫ | ✓ | ⑪ | ✗ | No | | No | | No | | No | | ⑩ | ✗ | 5 |
| Gisselquist lite ($GC_L$) | ⑥ | ✓ | ⑨ | ✓ | | | | | No | | ◇4 | ✗ | ◇3 | ✗ | ⑤ | ✗ | No | | 5 |
| Gisselquist full ($GC_F$) | ⑥ | ✓ | ⑨ | ✓ | ⑫ | ✓ | ⑪ | ✗ | No | | ◇4 | ✗ | ◇3 | ✗ | No | | ⑩ | ✗ | 7 |
| Xilinx Interconnect lite ($XI_L$) | No | | ⑨ | ✓ | | | | | No | | ◇4 | ✗ | ◇3 | ✗ | No | | No | | 3 |
| Xilinx Interconnect full ($XI_F$) | ◇6 | ✓ | 9 | ✓ | 12 | ✓ | 11 | ✗ | ⑬ | ✓ | ◇4 | ✗ | ◇3 | ✗ | 6 | ✓ | 10 | ✗ | 9 |
| Xilinx Smartconnect (XS) | 6 | ✓ | 9 | ✓ | No | | No | | No | | ◇4 | ✗ | ◇4 | ✗ | ◇6 | ✓ | 10 | ✗ | 6 |

TABLE IV: XRAY **Results.** Greyed-out cells represent vulnerabilities that cannot occur in lite interconnects; numbers correspond to the properties triggered by the vulnerabilities, as listed in Tab. I; circle indicates vulnerabilities that hold unconditionally; diamond denotes vulnerabilities that occur only under specific parameter combinations; square marks vulnerabilities present in certain manager/subordinate mode combinations (lite/Full); ✓and ✗indicate whether the vulnerabilities were detected by AMD Xilinx and Cadence VIPs and protocol checker.

4 requests: the same 3 as before, plus a request to address 1. At this point, the buffer of issued transactions is full, causing additional transactions to be dropped on S1's side as a malicious stalling of read ready signal propagates to it. This leads to a chain reaction, where M2's read is not processed due to this stalling, resulting in a DoS for both managers and data corruption, as the result is incorrect. This dropped read indicates that ultimately M1 exploits a vulnerability in the subordinate but the interconnect also failed to handle the overflow correctly. XRAY's ordering property detected this and flagged the error.

**Concurrency.** We simulate a scenario where a user accelerates a classification task using AWS cloud FPGA. The user has access to a host VM, enabling them to upload their design onto the FPGA. In addition to including 2 AMD Xilinx IPs like a small memory component, and an XS interconnect, the user integrates a firewall that guarantees no stalling or unauthorized access to protect the PS and the custom logic from each other. To perform arithmetic operations necessary for classification, the user employs a pre-built arithmetic (ALU) IP without access to its underlying RTL. In this scenario, the attacker controls the ALU IP and continuously initiates write transactions to perform computations. Their goal is to alter the output on the host side without alerting the user to the corruption of their program. We exploit a vulnerability identified by XRAY in XS, which occurs when the interconnect processes concurrent read and write transactions, leading to memory corruption on the subordinate's side and compromising execution integrity. XRAY's concurrency property flagged this. As shown in Fig. 8, when the host sends a sequence of read transactions to execute the classification task (from addresses 0 and 4), and the addition IP simultaneously initiates a write transaction to address 8 with value b, the interconnect mishandles the interleaving of these transactions while propagating the non AXI-compliant memory subordinate behavior. Consequently, b is written to 0, causing the host to read incorrect data when accessing the address again. This erroneous execution occurs without any warnings from the shell or the firewall integrated into the custom logic. As a result, the user ends up with corrupted data for their task and remains unaware of the issue. While we assume the ALU IP is attacker-controlled, the triggered traffic remains AXI-compliant and could potentially occur accidentally during benign operations, though it may not be deterministic in our exploit.

**Wrong Bus Sharing.** We recreate the scenario described in Fig. 3 from Section III, where M1 and M2 are connected to S1 via $PU_L$. In this case, M2 is an addition IP untrusted, while M1

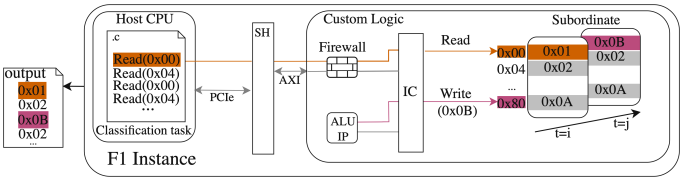


Fig. 8: **AWS F1 setup.** Host VM connected to an FPGA VCU118 with an AWS shell directly integrated. The custom logic has XS as interconnect connecting the shell and an ALU IP to the subordinate acting as memory. An AMD Xilinx Firewall IP is placed between the shell and the interconnect.

reads sensitive data from the small memory. The vulnerability allows M2 to gain access to M1's sensitive data on the bus.

## VII. Related Work

Interconnects have been studied for risks of DoS and stalling. New interconnects address this with fairness in arbitration of resource allocation or predictability for real-time design [2], [11], [21]–[23]. Adding wrappers and modules to existing interconnects is another effective solution to ensure specific system requirements (e.g., access control mechanism or traffic regulation) for real-time systems [24], [25]. XRAY focuses on security challenges such as transaction integrity or data leakage.

Fern et al. highlight hardware Trojan attacks on TrustZone [26], [27], emphasizing the need for interconnect security. Fuzzing or CVE analysis [28]–[30] can uncover CPU core vulnerabilities or malicious hardware [31]–[34]. These methods focus on known attack vectors but may overlook subtle issues or create unrealistic, non-compliant traffic. XRAY uses targeted and compliant traffic to uncover vulnerabilities.

Verification IPs and protocol checkers are popular, with offerings from AMD Xilinx [3], [6], [20], Cadence [4], [5], and Synopsys [35]. XRAY reasoning stems from our properties to find new vulnerabilities. eXpect captures AXI protocol functional and security specifications [13] but does not reason about interconnects. eXpect uses signal-level interactions between managers and subordinates while XRAY uses transactions. More importantly, eXpect does not reason about 8 of the 13 (P3, P4, P6, P7, P9, P10, P11, P13) XRAY interconnect properties.

## VIII. Conclusion

XRAY systematically analyzes the security of AXI interconnects, uncovering 41 vulnerabilities across 7 interconnects, including 22 that were previously unidentified. We selected 5 of these vulnerabilities to develop 3 end-to-end exploits.

## REFERENCES


[1] *AMBA AXI Protocol Specification*, ARM, July 2019, ARM IHI 0022G.

[2] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? Restoring fairness in axi interconnects for fpga socs," *ACM Transactions on Embedded Computing Systems (TECS)*, 2019.

[3] AMD Xilinx, "AXI Xilinx Firewall," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi-firewall.html.

[4] Cadence, "AXI Cadence Formal Verification IP," Accessed: Sept. 21, 2024. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/verification-ip/formal-vip/amba-arm.html#axi-protocols.

[5] ——, "AXI Cadence Simulation Verification IP," Accessed: Sept. 21, 2024. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/verification-ip/simulation-vip/amba/amba-axi.html.

[6] AMD Xilinx, "AXI Xilinx Verification IP," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi-vip.html.

[7] ——, "AXI Xilinx interconnect implementation," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_interconnect.html.

[8] ——, "AXI Xilinx smartconnect implementation," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_smartconnect.html.

[9] Pulp Platform, "Open Source Platform," Accessed: Sept. 21, 2024. [Online]. Available: https://www.pulp-platform.org/.

[10] Dan Gisselquist, "Using a formal property file to verify an AXI-lite peripheral," Accessed: Sept. 21, 2024. [Online]. Available: https://zipcpu.com/formal/2018/12/28/axilite.html.

[11] Z. Jiang, K. Yang, N. Fisher, I. Gray, N. C. Audsley, and Z. Dong, "Axi-ic$^{\mathrm{rt}}$ : Towards a real-time axi-interconnect for highly integrated socs," *IEEE Transactions on Computers*, vol. 72, no. 3, pp. 786–799, 2023.

[12] PULP, "AXI PULP crossbar implementation," Accessed: Sept. 21, 2024. [Online]. Available: https://github.com/pulp-platform/axi/blob/master/src/axi_xbar.svl.

[13] M. Zonta-Roudes, A. Meza, N. Hinderling, L. Deutschmann, F. Restuccia, R. Kastner, and S. Shinde, "eXpect: On the Security Implications of Violations in AXI Implementations," in *ACM/IEEE ICCAD*, 2024.

[14] AXI Full Crossbar, "AXI Full Crossbar ZipCPU implementation," Accessed: Sept. 21, 2024. [Online]. Available: https://github.com/ZipCPU/wb2axip/blob/master/rtl/axixbar.v.

[15] AMD Xilinx, "Vivado," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[16] ——, "Vivado Axi Peripheral Creation," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/video/hardware/creating-an-axi-peripheral-in-vivado.html.

[17] ——, "VCU118," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vcu118.html.

[18] ——, "ZCU102," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html.

[19] ——, "Vitis," Accessed: Sept. 21, 2024. [Online]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html.

[20] ——, "AXI Xilinx Protocol Checker," Accessed: Sept. 21, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_protocol_checker.html.

[21] F. Restuccia and R. Kastner, "Towards zero-trust hardware architectures in safety and security critical system-on-chips," in *Real-Time Intell. Edge Comput. Workshop (RAGE) Co-Located 61th Design Autom. Conf.(DAC)*, 2024.

[22] A. Meza, F. Restuccia, R. Kastner, and J. Oberg, "Safety verification of third-party hardware modules via information flow tracking," in *Real-Time Intell. Edge Comput. Workshop (RAGE) Co-Located 59th Design Autom. Conf.(DAC)*, 2022.

[23] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *ACM/IEEE DAC*, 2020.

[24] F. Restuccia, A. Meza, and R. Kastner, "Aker: A design and verification framework for safe and secure soc access control," in *ACM/IEEE ICCAD*, 2021.

[25] T. Benz, A. Ottaviano, R. Balas, A. Garofalo, F. Restuccia, A. Biondi, and L. Benini, "Axi-realm: A lightweight and modular interconnect extension for traffic regulation and monitoring of heterogeneous real-time socs," in *IEEE DATE*, 2024.

[26] N. Fern, I. San, C. K. Koç, and K.-T. Cheng, "Hardware trojans in incompletely specified on-chip bus systems," in *IEEE DATE*, 2016.

[27] ARM, "Trustzone," Accessed: Sept. 21, 2024. [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-a.

[28] K. Laeufer, J. Koenig, D. Kim, J. Bachrach, and K. Sen, "Rfuzz: Coverage-directed fuzz testing of rtl on fpgas," in *ACM/IEEE ICCAD*, 2018.

[29] S. Canakci, L. Delshadtehrani, F. Eris, M. B. Taylor, M. Egele, and A. Joshi, "Directfuzz: Automated test generation for rtl designs using directed graybox fuzzing," in *ACM/IEEE DAC*, 2021.

[30] R. Zhang, C. Deutschbein, P. Huang, and C. Sturton, "End-to-end automated exploit generation for validating the security of processor designs," in *MICRO*, 2018.

[31] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *IEEE S&P*, 2016.

[32] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating uci: Building stealthy and malicious hardware," in *IEEE S&P*, 2011.

[33] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "HardFails: Insights into Software-Exploitable hardware bugs," in *USENIX Security*, 2019.

[34] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE S&P*, 2010.

[35] Synopsys, "AXI Synopsys Verification IP," Accessed: Sept. 21, 2024. [Online]. Available: https://www.synopsys.com/verification/verification-ip/amba/amba-axi.html.