eXpect: On the Security Implications of Violations in AXI Implementations

Melisande Zonta-Roudes
ETH Zurich

Andres Meza
University of California San Diego

Nora Hinderling
ETH Zurich

Lucas Deutschmann
RPTU Kaiserslautern-Landau

Francesco Restuccia
University of California San Diego

Ryan Kastner
University of California San Diego

Shweta Shinde
ETH Zurich

Abstract

The Arm Advanced eXtensible Interface (AXI) protocol is a widely used on-chip interconnect for processors, accelerators, memories, and other IPs. Any bugs in the AXI implementations pose a security risk to the chip's correctness. Buggy or non-compliant third-party IPs can use AXI implementation bugs to bypass the security mechanisms of the whole system. Identifying AXI implementation bugs is challenging because the incomplete specifications allow room for implementation-specific behavior in performant designs. eXpect is a systematic approach for analyzing AXI implementations to detect functional and security violations. We use eXpect to test 7 implementations of varying complexity, including the ones from AMD Xilinx and RISC-V PULP. We identified 135 property violations. We sampled 10 of them to show 7 exploits demonstrating that an attacker can use these bugs to trick victim IPs. Our exploits achieve outcomes such as using stale data, skipping reads and writes, leaking intermediate data, and reading and writing attacker-controlled data to attacker-controlled addresses. We evaluated our exploits in realistic scenarios deployed on FPGA. We show that AMD Xilinx protocol checker IPs miss 5/7 of our exploits.

ACM Reference Format:
Melisande Zonta-Roudes, Andres Meza, Nora Hinderling, Lucas Deutschmann, Francesco Restuccia, Ryan Kastner, and Shweta Shinde. 2024. eXpect: On the Security Implications of Violations in AXI Implementations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676844

1 Introduction

The Arm Advanced eXtensible Interface (AXI) protocol is a specification for system-on-chip (SoC) communication [14]. It consists of interfaces such as AXI-Lite and AXI-Full and is a part of the Advanced Microcontroller Bus Architecture (AMBA) specification suite [11]. AXI connects Intellectual Property (IP) cores using a manager/subordinate model [47], [12]. It uses handshakes to coordinate communication via read-and-write transactions.

The AXI protocol intentionally maintains a level of ambiguity, which can lead to the development of divergent implementations of the same protocol, potentially resulting in IPs operating on contradictory assumptions. These discrepancies can result in functional and security issues, potentially compromising the system's integrity. For instance, an AXI implementation might inadvertently grant sensitive data access to unintended IPs that use stale data [30], [40]. Furthermore, an implementation could introduce memory corruption, jeopardizing data integrity and system reliability. Additionally, such discrepancies may result in execution integrity problems, undermining the system's trustworthiness. Addressing these challenges requires comprehensive validation and adherence to protocol specifications to mitigate the risks of divergent implementations.

eXpect is a tool to detect specification violations in AXI implementations. eXpect models the AXI protocol as properties that capture the timing relationship that can be expressed as SystemVerilog Assertions (SVAs). We capture the specifications into sets of properties to check adherence to AXI. Then we identify gaps within the existing specifications and add properties that reason about security. This comprehensive approach results in eXpect model composed of 13 functional and 17 security properties regrouped into 20 properties spanning 29 signals across unbounded time cycles. We demonstrate the effectiveness of eXpect by testing diverse AXI implementations that produce counterexamples for violated properties. We illustrate how these violations manifest as exploitable vulnerabilities on real-world setups on FPGAs. Next, we conduct a comparative analysis between eXpect and commercial verification IPs. Finally, we propose patches for eXpect-detected violations.

Results. We use eXpect to test 7 implementations varying from simple state machines, to use-case specific, to production versions from AMD Xilinx and RISC-V PULP. eXpect detects a total of 135 violations across these 7 implementations, ranging from 15-26 violations per implementation. We select 10 violations to showcase 7 exploitable vulnerabilities. They demonstrate how the implementation bugs can be used to compromise security-sensitive IPs (both in manager and subordinate roles). Our vulnerabilities exploit issues such as stalling, dropping, silent failures, and reordering transactions. We test realistic setups with functional IPs and interconnects that act as intermediaries between IPs and the AXI bus itself. Additionally, we test a range of IPs from AMD Xilinx that developers can integrate into their designs to act as Verification IPs, facilitating ongoing transaction checks and violation flagging [8], [7], [6]. We examine if their violations align with those reported by eXpect.

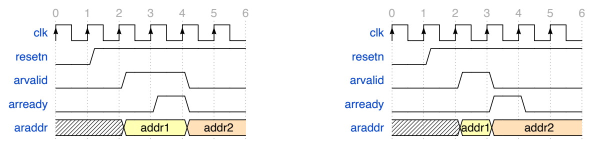

Figure 1: (a) Expected specifications behavior (b) Observed behavior

```
1. always @( posedge S_AXI_ACLK )
2.   begin
3.     if (~axi_arready && S_AXI_ARVALID)
4.       begin
5.         // indicates that the
6.         // subordinate has accepted
7.         // the valid read address
8.         axi_arready <= 1'b1;
9.         // Read address latching
10.         axi_araddr  <= S_AXI_ARADDR;
11.       end
12.     else
13.       begin
14.         axi_arready <= 1'b0;
15.       end
16.     end
17.   end
```

```
1. On every rising edge of clk:
2.   if (not arready and arvalid):
3.     // indicates that the
4.     // subordinate has accepted
5.     // the valid read address
6.     arready = 1;
7.     // Read address latching
8.     araddr = addr2;
9.   else:
10.     arready = 0;
11.   end if
12.end
```

Figure 2: (a) Verilog implementation of a part of the read transaction AXI subordinate secondary. The lower case signals indicate subordinate inputs and the upper case are manager inputs (subordinate outputs) (b) Pseudo code.

We report that the majority of eXpect vulnerabilities remain undetected, with only 2 out of 7 being identified by the AMD Xilinx IPs. Finally, we measure the impact of our proposed fixes to the original implementations.

Contributions. Our three main contributions are:
1. eXpect is the first model that attempts to capture not only the functional but also new security semantics of AXI at an intra-transaction level.
2. eXpect tests 7 implementations and reports 28 functional and 107 security violations. We build 7 exploits to showcase the security impact.
3. We identify verification gaps in existing industrial verification IPs and suggest remedies for their AXI implementations.

We disclosed our findings to PULP and AMD Xilinx on January 2024. eXpect is available at https://axi-security.github.io/expect.

2 Motivation

We take a concrete AXI implementation and explain how it violates the specifications and leads to security vulnerabilities.

2.1 Example

Consider a security-sensitive IP connected as a subordinate via an AXI bus to a buggy manager IP. The manager IP may access sensitive data or send wrong data to the subordinate IP, such as a memory, posing security risks like data leakage or memory corruption. For example, consider the AXI read address channel, which transmits the address from the manager to the subordinate IP. The subordinate IP should perform checks before allowing the manager to access any addresses. But the subordinate IP is not AXI-compliant so it does not do these checks. If the manager exploits AXI implementation bugs, it can lead to vulnerabilities, e.g., an address is checked and disallowed, but the manager still accesses it.

Expected Behavior. The read address channel of the read transaction uses three signals: arvalid and araddr are driven by the manager interface, and arready is driven by the subordinate. As stipulated in the Arm specifications [14], arvalid signifies the presence of a valid address on the bus. When asserted, it indicates the managers' request for a valid transaction. An assertion of arready implies the subordinate is ready to sample the address request from the manager. The coordination between the manager and the subordinate materializes when both arready and arvalid of the address read channel are concurrently asserted for one clock cycle, resulting in the transfer of the address. This handshake mechanism ensures a synchronized and reliable data transfer between the manager and the subordinate. To ensure a correct handshake, the specifications dictate a crucial requirement: "When arvalid is asserted, it must remain asserted until the rising clock edge following the assertion of the arready signal by the subordinate." Fig. 1.a illustrates this requirement where $addr_2$ is accurately sampled during the process. In this case, the Xilinx AXI-Lite subordinate implementation abides by an active low reset.

Observed Behavior. Fig. 2.a shows Verilog code for a synchronous always block that is sensitive to a positive edge of the clk signal. On a positive edge of clk, the block checks: (a) if the AXI read interface is not ready to accept a new address request (arready); (b) if a new AXI read address is available at the interface (arvalid). If both conditions are met, it sets arready to indicate that the subordinate has accepted the valid read address and latches the read address (araddr). If the conditions are not met, it sets arready to 0, indicating that the subordinate is not ready for a read operation. Thus the code snippet does not satisfy AXI requirements as described above. On line 3 highlighted in Fig. 2.a, arready should be high for at least one clock cycle and arvalid should remain high for one more clock cycle after arready is asserted. Hence, the code allows the timing diagrams shown in Fig. 1.b which shows that a handshake fails to occur, but due to the address latching in the implementation, $addr_1$ will reach the subordinate. However, since the address handshake (arready and arvalid both high) does not occur, no data is read. Consequently, $addr_1$ is sampled, contrary to the expected behavior. This poses a security risk as the system may operate based on outdated or erroneous data from $addr_1$ instead of $addr_2$ which will never be sampled. Moreover, no data handshake will follow because of the absence of $addr_1$ handshake leading to in a denial of service from the view of the subordinate.

2.2 Setup & Problem

In RTL development, testing and validation often involve the use of System Verilog Assertions (SVA) integrated directly into the RTL. These assertions aim to identify protocol violations and functional bugs to ensure the overall performance of the design. However, it is not always feasible to test and validate all the IPs as some IPs originate from various sources and providers. Vendors usually provide encrypted IPs for integration into trusted design. Hence, our threat model considers a generally unintentional buggy AXI implementation. Three reasons support this assumption: (a) the IP does not exhibit any bugs under a normal setting but will fail in corner case scenarios; (b) the encryption makes it difficult to analyze the IP for bugs; or (c) the IP is involuntarily non-compliant to the AXI protocol. Finally, even if the IPs are subject to validation and testing, it may not be sufficient to catch all security violations. In summary, we assume that at least one IP in the system does not adhere to AXI specifications. Then we show that if the system has an attacker-controlled IP, it can exploit AXI bugs in the victim IP leading to security violations (e.g., data leakage or corruption, DoS).

2.3 Solution Insight

The example from Fig. 1 emphasizes the need to extract specifications and derive properties from them. The specifications cited in the example dictate that "`arvalid` must remain asserted until the rising clock edge following the assertion of the `arready` by the subordinate." We can define a corresponding formal property that ensures the stability of `arvalid` until the appropriate address is sampled during the handshake of the read address channel. Hence if there was a tool that could check if a given implementation satisfied this property or not, we could detect specification violations. To this end, eXpect captures both functional and security specifications, in form of properties. For a given implementation, eXpect either deems the property to be satisfied or generates a counterexample that demonstrates the violation with a concrete input.

3 Background

eXpect captures the full AXI-Lite and burst mode of AXI-Full specification. We present the building blocks of AXI protocol [14].

Channels Description. The AXI protocol executes two types of transactions: read and write. AXI employs five distinct channels allowing for parallel read and write transactions, as illustrated in Fig. 3. The address channels convey both address and protection information for both transactions through the read and write address channel. In read transactions, data from memory or registers are transmitted back to the manager through the read data channel. In write transactions, the write data channel transports the data to be written to the target address. Additionally, a dedicated acknowledgment channel for write transactions transfers acknowledgment responses from the subordinate to the manager. We reason about all signals in AXI-Lite. For AXI-Full, we concentrate on the subset of signals listed below since all the implementations that we tested only support these signals. We categorize the signals according to channel c, where c is a prefix replaced by AR, R, AW, W, or B, depending on the channel:

Handshaking signals: AXI uses `cready` and `cvalid` for handshaking between manager and subordinate. `cvalid` indicates valid information, while `cready` shows readiness to accept data.

Transaction and control signals: Address and data channels carry different information, denoted by `caddr` and `cdata` while protection signals (`cprot`) ensure data transfer security.

Response signals: Response signals (`cresp`) on data channels and acknowledgment channels indicate transaction outcomes.

Burst signals: Signals `clen`, `csize`, `cburst`, and `clast` optimize data transfers, specify transfer count, address size, burst type, and conclusion of bursts, respectively.

Transactions. In a read transaction (refer to Fig. 3.a), while the system is not under reset (`reset = 0`), the manager initiates the process during the read address channel handshake (`arvalid` and `arready` high). It transmits the address (`araddr`) and protection information (`arprot`) on the channel. This prompts the subordinate to retrieve data from the provided address. The subordinate processes the request and transmits data during the read data channel handshake from the specified memory or registers to the manager. It places the data (`rdata`) on the read data channel along with the

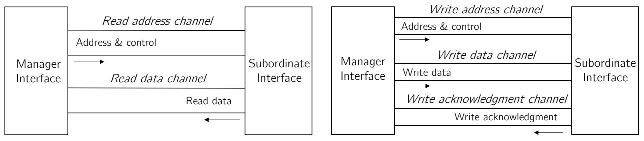

Figure 3: AXI-Lite transactions (a) Read (b) Write; For AXI-Full, data will be transferred in bursts

response signals (`rresp`) and acknowledgment. After receiving the data, the manager processes the data for further use. In this case, the specifications maintain proper channel ordering [14], ensuring that the activation of the read address channel precedes the corresponding data one. In a write transaction (Fig. 3.b) outside of reset, the manager initiates the process by transmitting the write address and protection information (`awaddr` and `awprot`) via the write address channel handshake (`awvalid` and `awready`). In the write data channel handshake (`wvalid` and `wready`), the manager sends the data (`wdata`) on the data channel without a specific ordering between the channels. The subordinate then writes the data to the correct location, and acknowledges the write transaction's status through `bresp` on the write acknowledgment channel.

Burst Transactions. In an AXI read transaction in burst mode, the manager defines the burst by setting parameters `arlen`, `arsize`, and `arburst`. Unlike a non-burst read, the subordinate accesses consecutive addresses within the burst range, fetching and sequentially placing data (`rdata`) on the read data channel. The subordinate asserts `rlast` to signify the end of the burst transfer. In an AXI write transaction under burst mode, the burst parameters (`awlen`, `awsize`, and `awburst`) are defined similarly, and address processing and last data transfer indication operate the same as in a read transaction. The `wlast` signal marks the end of the burst transfer.

4 eXpect Model

We highlight our insights, present eXpect properties, and explain the security implications of property violations.

4.1 Challenges and Approach

We consider potential models and reason for our choices.

Timing. Hardware protocol analysis primarily emphasizes a transactional perspective instead of timing [29], [46]. A timing agnostic analysis can miss issues for data arrival times and handshakes, which are crucial for system synchronization and reliability. This can result in an incomplete protocol behavior and security gaps. Thus eXpect models timing requirements within a transaction.

Granularity. As outlined in Section 3, the AXI protocol operates at handshakes and channels granularity. On one hand, monitoring signal values helps analyzing handshakes, ensuring accurate timing. On the other hand, channel-level modeling is useful for concerns like channel ordering to capture overall event sequences. eXpect analyses at the channel level rather than at the transaction level i.e, signal relationships within channels and channel ordering to capture intra-transaction properties.

Handling Ambiguity. Translating hardware protocols specifications into properties requires handling specifications ambiguities. They may arise from vague descriptions, differing interpretations, or incomplete specifications. SVAs require a level of clarity and unambiguity that may not align with the nature of hardware protocol specifications. To address this, eXpect focuses on the mandatory requirements, leaving aside the ambiguous statements that do not endanger the system's functional correctness and reliability.

Table 1 illustrates two common types of statements found in the specification. The first example e.g.1 demonstrates a mandatory requirement leaving little room for interpretation. However, some implementations deviate from this requirement, as depicted in Interpretation 1. Interpretation 2 aligns with our deemed correct interpretation from which we derive the SVA. The second example e.g.2 shows a recommendation where no specific interpretation takes precedence, as both behaviors are deemed acceptable. While we typically opt not to model e.g.2 type, as it is very ambiguous, in this case, we use it as a baseline for refining the base model into a more secure version, implementing Interpretation 2 in an SVA. Implementations adhering to Interpretation 1 are compliant with the specifications but may pose a security risk; hence, we recommend compliance with Interpretation 2. Our specification refinement in the second scenario establishes a precise channel ordering, ensuring the accurate association of data with addresses.

4.2 Functional and Security Properties

In eXpect, we manually extract functional properties from the specifications by translating them into timing properties as SVAs. We summarize our properties below and list them in Table 2.

Base. We first capture functional properties.

Channel Ordering (P_{1-2}). In a read transaction, the data channel should become active after the address handshake. For the write, the acknowledgment should follow the data and address handshake.

Stability (P_{3-5}). For both transactions, the specification mandates correct sampling of address, control information and acknowledgments. Hence, the AXI protocol mandates that these signals should stay stable until the corresponding channel handshakes completion.

Reset mechanism (P_{6-7}). AXI supports both asynchronous and synchronous reset mechanisms to reset the interface and restore the system to a known state. The specification only states the reset of the valid signals of each channel.

Security. Next, we add the security properties to eXpect.

Sensitive Data Stability (P_{8-9}). Although the specifications mandate the stability of certain signals, they do not encompass all sensitive signals, such as data and response across every channel.

Error handling (P_{10}). AXI maintains data transaction integrity and reliability via its response signals, enabling the manager to ascertain transaction outcomes and respond accordingly. The encoding of these signals indicates transaction validity, so eXpect includes properties to verify this correlation.

Strict Ordering (P_{11-13}). The specification dictates mandatory sequences for some handshakes and signals, but there is significant ambiguity and flexibility concerning others. eXpect refines these requirements. In write transactions, eXpect ensures dependable execution by stipulating that data transmission must follow the transmission of the address specifying its intended write location. After sampling the associated address and writing the data, there should be an acknowledgment. We apply this to the burst mode too, making sure the acknowledgment follows the last data transfer.

Advanced Reset mechanism (P_{14-15}). The reset mechanism does not cover the clearing of all signals, so eXpect completes it.

Stale data invalidation (P_{16}). A low valid signal on the read data channel indicates that the read data becomes invalid. Consequently, eXpect clears associated data when this valid signal is low.

Strict Ordering in burst mode (P_{17-20}). eXpect adapts the strict ordering properties to the burst mode.

4.3 Effects of Properties Violations

Direct violations of the above properties can lead to diverse categories of undesirable effects: (a) Ordering properties violations can lead to stale data usage which violate integrity of subsequent transactions because it uses outdated information; (b) Stability properties violations can result in dropped transactions leading to incomplete or lost data transfers (c) Ordering properties violations introduce unintended memory alterations when potential write and read operations use incorrect addresses (d) Stability and ordering properties violations can also lead to stuck transactions.

5 eXpect Tool

eXpect has 30 properties, which we test on 7 implementations.

5.1 eXpect Implementation

We implement eXpect using the Questa simulation and verification tool [37]. Given a set of properties and an implementation, it classifies properties as being violated or proven. When violated, it produces a counter-example trace. We analyze the trace's violated property. Next, we execute the counterexample input in a Vivado test bench alongside other signals from the implementation to validate reproducibility and investigate the impacts [9]. We categorize the impacts as functional or security-related and develop exploits.

5.2 Evaluation Targets

Selection of AXI Implementations. The AXI protocol is implemented in Verilog, VHDL, and SystemVerilog. We examine open-source (e.g., RISC-V, personal projects) and proprietary implementations, the latter involving Verilog extraction by creating axi4 peripheral in Vivado[10]. We test a simple manager or subordinate IP and use a dedicated testbench to confirm functional validity of the AXI implementation. Table 3 shows our test subjects and their respective complexities. We deliberately choose implementations with varying complexity (line count) to explore potential correlations with the number of counterexamples.

1. SM [20] and SF [21] are open-source implementations showing an interpretation of the specifications as a state machine and a simplified version of the AXI-Lite protocol's interface, respectively.
2. XS_L [4] and XM_L [3] are AMD Xilinx AXI-Lite subordinate and managers implementations extracted using Vivado respectively. XS_F [2] and XM_F [1] are AXI-Full counterparts. The subordinate emulates a small memory system, utilizing four built-in registers.
3. PU is part of PULP, an open-source RISC-V project [35].

Implementations Viability. We select these implementations by testing their functionality, read and write transactions with one data transfer at a time and in burst mode through simulation. We only pick implementations that pass these tests successfully, confirming their operability and basic functionality. During this selection, our objective is not to induce edge cases or expose non-compliance

Table 1: Example of specifications, interpretations, and corresponding SVA

e.g.	AXI Specifications	Interpretation 1	Interpretation 2	SVA
1	"When asserted, `arvalid` must remain asserted until the rising clock edge after the subordinate asserts `arready`"	`arvalid` can be unasserted when `arready` is asserted	`arvalid` needs to still be asserted when `arready` is asserted	assert property (@(posedge `clk`) disable iff (`arvalid` && `arready`) (!`reset` && `arvalid` && !`arready`) ⇒ `arvalid`);
2	"The subordinate can assert `wready` before `awvalid` or `wvalid`, or both, are asserted"	`wready` is asserted before `awvalid`	`awvalid` and `awready` must be asserted before asserting `wready`	assert property (@(posedge `clk`) (!`reset` && `awvalid` && `awready`) ⇒ !`wready`);

Table 2: eXpect Properties

P #	c.	Property.	
1	-	`rvalid` cannot be asserted simultaneously with `arvalid` and `arready`	
2	-	`bvalid` cannot be asserted simultaneously with `wvalid` and `wready`	
3	all	`cvalid` must be high until the rising clock edge after `cready` is asserted	
4	AR,AW	`caddr`	must remain stable until the rising clock edge after `cready` is asserted
5		`cprot`	
6	AR,AW,W	`cvalid`	should be nulled out during synchronous or asynchronous reset
7	R,B	`cvalid`	
8	R,W	`cdata`	must remain stable until the rising clock edge after `cready` is asserted
9	R,B	`cresp`	
10	R,B	`cresp`=00 if the transaction is successful (AXI-LITE specific)	
11	-	Once `araddr` has been sampled, `rdata` should eventually follow	
12	-	Once `awaddr` has been sampled, `wdata` should eventually follow	
13	-	The acknowledgment of the write transaction should follow the associated address and data	
14	R,B	`rdata`, `cresp`	should be nulled out during synchronous or asynchronous reset
15	AR,AW	`caddr`, `cprot` `wdata`	
16	-	Data must be cleared if `cvalid` is deasserted	
17	R,W	`clast` must be asserted at the last data transfer while `cvalid` is asserted	
18	-	Same as P_{11}-P_{13} in burst mode with added dependency to `clast`	
19			
20			

Table 3: Implementations details with number of lines (LOC)

Implementations	Name	Language	AXI role	LOC
State-machine [20]	SM	System Verilog	Subordinate	76
Subordinate for FPGAs [21]	SF	Verilog	Subordinate	153
Xilinx Lite Subordinate [4]	XS_L	Verilog	Subordinate	404
Xilinx Lite Manager [3]	XM_L	Verilog	Manager	433
PULP [35]	PU	System Verilog	Subordinate	463
Xilinx Full Subordinate [2]	XS_F	System Verilog	Subordinate	619
Xilinx Full Manager [1]	XM_F	System Verilog	Manager	906

with specifications. The selected subordinate implementations undergo functional reset and read/write traffic generator tests, while the manager ones are assessed with resets and traffic responses. Although `cprot` is part of the AXI specifications, none of the implementations have any logic for the protection signals.

6 Evaluation

We first analyze the eXpect violations and then build exploits. We compare eXpect to verification IPs and patch the violations.

6.1 Analysis of Reported Violations

We execute eXpect on 7 implementations and report the summary of the violations in Table 4. We observe 135 violations and 50 proven properties across the 7 implementations. There is no correlation between the number of violations and implementation complexity. The implementations with more features trigger a higher number of property violations than SM and SF with omitted features like access control. As for XM_L and XM_F, the number of violations is 21 and 26, respectively, which can be explained by poor manager implementation. We also observe that certain properties are

Table 4: Results for violations for 7 implementations where ✗represents the properties violated and ✓the proven ones

Prop. Category	P #	c.	SM	SF	XS_L	XM_L	PU	XS_F	XM_F
Channel Ordering	P_1	-	✓	✗	✗	✗	✓	✓	✗
	P_2	-	✓	✗	✓	✗	✓	✗	✗
Stability	P_3	AR	✗	✗	✗	✗	✗	✗	✗
		R	✗	✓	✓	✗	✗	✓	✗
		AW	✗	✗	✗	✗	✗	✗	✗
		W	✗	✗	✗	✗	✗	✗	✗
		B	✗	✓	✓	✗	✗	✓	✗
	P_4	AR	✗	✗	✗	✗	✗	✗	✗
		AW	✗	✗	✗	✗	✗	✗	✗
	P_5	AR	✗	✗	✗	✓	✗	✗	✓
		AW	✗	✗	✗	✓	✗	✗	✓
Reset mechanism	P_6	-	✓	✗	✓	✓	✓	✓	✓
	P_7	-	✓	✗	✓	✓	✓	✓	✓
Sensitive data stability	P_8	R	✗	✓	✓	✗	✗	✗	✗
		W	✗	✗	✗	✗	✗	✗	✗
	P_9	R	✓	✗	✓	✗	✓	✓	✗
		B	✓	✓	✓	✗	✓	✓	✗
Error handling	P_{10}	R	✓	✗	✓	✗	✓	✓	✗
		B	✓	✗	✓	✗	✓	✓	✗
Strict ordering	P_{11}	-	✗	✗	✗	✗	✓	✗	✗
	P_{12}	-	✗	✗	✗	✗	✓	✗	✗
	P_{13}	-	✗	✗	✗	✗	✗	✗	✗
Advanced reset mech	P_{14}	-	✗	✗	✗	✗	✗	✗	✗
	P_{15}	-	✗	✗	✗	✗	✗	✗	✗
Stale data	P_{16}	-	✗	✗	✗	✗	✗	✓	✗
Last burst check	P_{17}	R	NA	NA	NA	NA	NA	✗	✗
		W	NA	NA	NA	NA	NA	✗	✗
Strict ordering in burst mode	P_{18}	-	NA	NA	NA	NA	NA	✗	✗
	P_{19}	-	NA	NA	NA	NA	NA	✗	✗
	P_{20}	-	NA	NA	NA	NA	NA	✗	✗
Total ✗/Total ✓	-	-	17/8	21/4	15/10	21/4	15/10	20/10	26/4

consistently violated across all implementations. P_3(AW, W), P_4, and P_8(W) are not enforced at all according to the specifications; these properties primarily pertain to the stability of information (address, data). On the other hand, properties P_{13}, P_{14} as well as P_{20} and P_{19}, P_{15}-P_{17} are part of our refined model as security properties so it is not surprising that they have not been included in most implementations. P_{16}, which is violated in the AMD Xilinx AXI-Lite implementation (XS_L), has been added in the AXI-Full version (XS_F). Moreover, PU [36] unlike the other implementations enforces strict ordering in read transactions between the address and data channels and between address/data and acknowledgment channels in write transactions. Overall, the PULP implementation demonstrates higher consistency in address and data latching. PU adheres correctly to the address latching from the specifications.

6.2 From Violations to Vulnerabilities

We take the counter-examples generated by eXpect and manually analyze them. We create a testbench to confirm the behavior and run it in simulation using Vivado. We refine our initial traffic generators and responders to generate synthesizable design sources. We then load these design sources onto Virtex UltraScale+ FPGA VCU118. Next, our goal is to craft exploits using single or multiple property violations to bring about effects such as dropping transactions, memory corruption, using stale data, and DoS.

Setups Under Test. We select three manager-subordinate setups shown in Fig. 4. In (a) and (b), either the manager or the

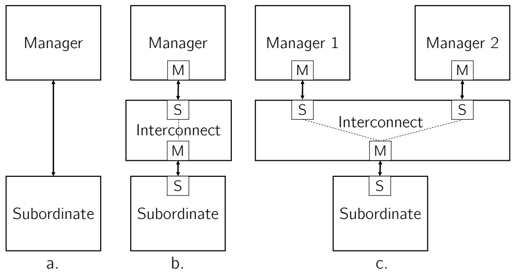

Figure 4: Three setups of communication between a manager and a subordinate

Table 5: Mapping vulnerabilities to scenarios and implementations

Vulnerability #	Setup #	Impl. Mgr	Impl. Sub.	Interconnect
1	1	NA	XS_L, PU	Xilinx[5]
2.a	2	NA	XS_L, PU	NA
2.b	1	NA	XS_L	NA
3,4,7	1	XM_F	XS_F	NA
5,6	3	XM_L	XS_L	Custom

subordinate is attacker-controlled, while the other components (manager/subordinate and interconnect) are the victims and their implementations remain unchanged. In (c), Manager 1 is regarded as benign, while Manager 2 is considered buggy or non-compliant. We develop a *custom interconnect* that follows two key principles. First, it ensures atomicity of access by separately granting access to each manager. This means that when Manager 1 attempts to access the subordinate, Manager 2 will receive only null data. Second, the interconnect will transfer access to Manager 2 only once the handshakes for the read address/data channel or write address/data channel and write acknowledgment for Manager 1 are completed. Table 5 shows the configuration for each exploit.

6.3 Dropped Transactions

eXpect detects stability and ordering property violations for all implementations. This can lead to dropped read and write transactions. To demonstrate their impact, we use a non-compliant hence untrusted AXI manager, which is an IP performing additions, connected to a victim subordinate acting as memory.

Vulnerability 1: Silent transaction dropping in write on PU. As highlighted in Table 5, the attack scenario is Fig. 4.a with the addition IP manager communicating directly to the subordinate. We exploited PU but our approach is applicable to XS_L.

Violation: In Fig. 5, the manager writes the data `06` to address `10` at t=2, `08` to `20` at t=4 and `0e` to `30` at t=7. The problem arises because only two out of the three handshakes for sampling the three addresses and data occur (t=2, t=7). This observation implies that address `20` and data `08` are not being sampled by the subordinate (highlighted as red section in Fig. 5.b) which violates $\mathtt{P}_4\mathtt{(AW)}$.

Consequence: To perform the addition, the manager requests to read the data from addresses `10`, `20` and `30`. The data read is `06` and `0c` from `10` and `30` respectively. However, at t=17, nothing is read from address `20` (`00`). Hence, the manager sums up the values which results in `12` instead of `1a`.

Vulnerability 2: Silent transaction dropping in read. In violation 2.a the setup is Fig. 4.b connecting the manager addition IP to the subordinate through AMD Xilinx crossbar while in violation 2.b the manager is directly connected to the subordinate Fig. 4.a.

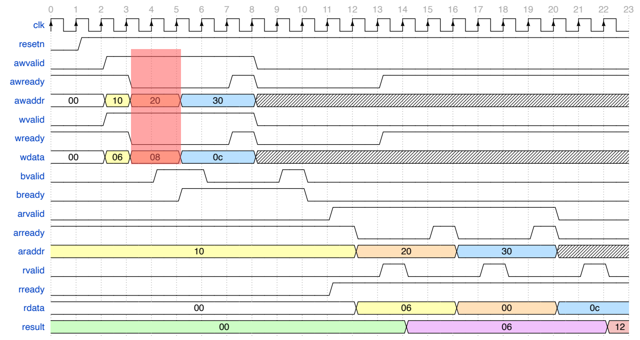

Figure 5: Dropped write transaction for PU

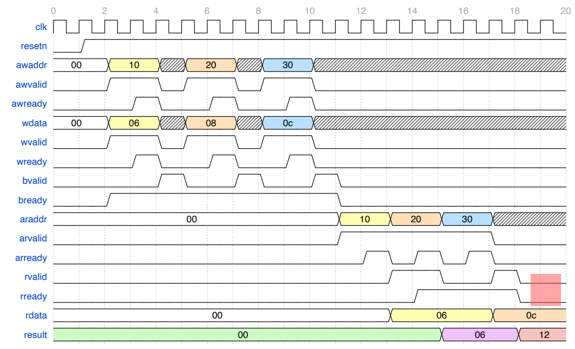

Figure 6: Dropped read transaction for XS_L

Violation 2.a: We exploit PU. The manager proceeds to do similar write transactions as in Vulnerability 1. Then, the manager proceeds to execute three sequential read requests from each address at t=14, 16, and 18, respectively. During this process, the accesses from `10` and `30` clearly display the contained data `06` and `0c` respectively. The problem arises from the read address request from `20` which is disregarded because of the absence of a handshake and the address which should have stayed the same until `arready` is high according to $\mathtt{P}_4\mathtt{(AR)}$. This approach is also applicable to XS_L.

Violation 2.b: We exploit XS_L. In Fig. 6, the manager proceeds with the same sequential write transactions as before. Subsequently, three reads are requested. At time t=12, the handshake for address `10` occurs correctly and at t=14 for `06`. However, the handshake for address `20` coincides with the first data handshake, resulting in the second data handshake occurring after the third address handshake. Consequently, the second address is overwritten before the second data handshake, which violates the read channel ordering property $\mathtt{P}_{11}$. It leads to only `0c` from `30` being read missing `08` from `20`.

Consequence: The manager, the addition IP, proceeds to sum up the data after the transactions' acknowledgments. The sum is `12` instead of `1a`, hence the arithmetic operation is corrupted.

6.4 Unintended Memory Alterations

eXpect counterexamples reveal memory corruption issues when requesting parallel read and write transactions or burst write.

Vulnerability 3: Concurrent read and write transactions for XS_F. The attack scenario is Fig. 4.a with the non-compliant hence untrusted AXI manager connected to the victim subordinate. In Fig. 7, the manager simultaneously initiates a write and a read.

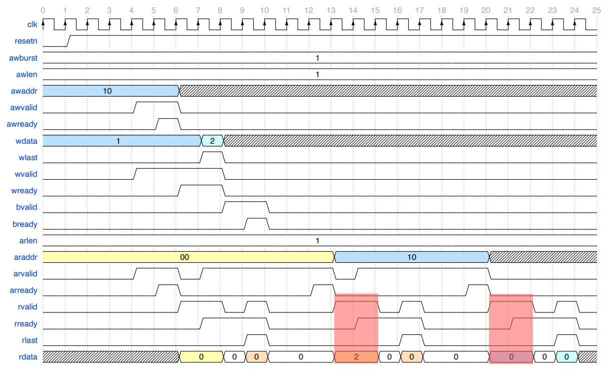
Figure 7: Wrong read data due to memory corruptions for XS_F

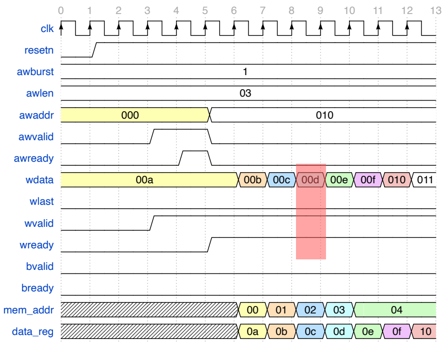
Figure 8: Memory corruption after an non-allowed write transaction for XS_F

Violation: Specifically, the manager aims to write the data `1` to address `10` while also requesting a read from address `00` at t=4. However, despite the AXI protocol being designed to accommodate such scenarios, these transactions are mishandled by the subordinate. At t=6 and t=9, no data is stored in address `00`, as it is solely read during the transaction. Subsequently, the manager performs another read from address `00` at t=12, it reads `2` at t=13 and `0` at t=16. This is correlated to the violation of $\mathtt{P}_8(\mathtt{R})$. The problem is that no write operation has been executed on address `00`. Then at t=19, when address `10` is latched for the final read transaction, both readouts from address `10` yield `0` despite expecting `1` and `2`.
Consequence: Simultaneous read and writes cause a violation, which results in memory mishandling. This causes memory corruption, with value 2 stored at address 0 instead of 10, and value 1 dropped.
Vulnerability 4: Memory corruption in XS_F. The setup is Fig. 4.a with the non-compliant manager connected to the victim.
Violation: In Fig. 8, the manager initiates a burst write transaction of length 4 with an incrementing burst type. The manager wants to write the data `00a`, `00b`, `00c` and `00d` at address `000` as shown by the handshake at t=4 but wants to perform another write as indicated by a new address `010` on the bus at t=5. The problem arises from the subordinate never sending the acknowledgment handshake and not raising `wlast` signal, which violates the strict ordering P_{19}.
Consequence: Unfortunately, the write transaction to address `000` does not perform the correct number of write as shown by the internal registers `mem_addr` and `data_reg` since data is being written to address `004`, which is one more increment than the length specified.

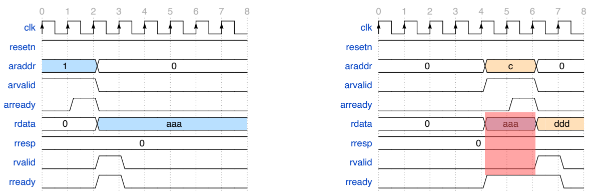
Figure 9: Confidential data breach vulnerability discovered in XS_L (a) Benign manager (b) Untrusted manager

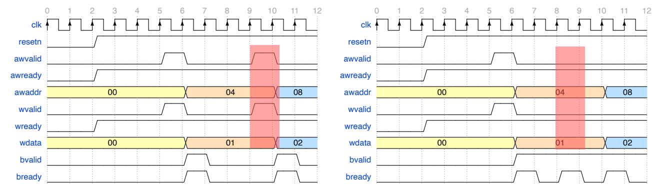
Figure 10: Write transaction in XM_L (a) followed by another write transaction (b) not followed by write transaction due to a DoS caused by `bvalid`

6.5 Stale Data

eXpect identifies violations of stale data clearing. If an AXI non-compliant IP manages to intercept data intended for a benign manager due to improper channel ordering, it can read protected information or trick the victim into using stale data.
Vulnerability 5: Read transaction data leak on XS_L. We use the setup shown in Fig. 4.c and Table 5. One of the two managers is considered trusted and the other AXI non compliant hence untrusted and they both communicate with a benign subordinate. Here we exploit XS_L but this is also applicable for PU.
Violation: Looking at the benign manager in Fig. 9.a, at t=1 `arvalid` and `arready` are high, ready to sample the address 1, then at t=2 `rvalid` and `rready` also become high and sample the data `aaa` to be returned from address 1. Then the non-compliant manager (Fig. 9.b) proceeds to a read request by sampling address `c` at t=5. The problem comes at t=4 where the data available to the untrusted manager is actually `aaa` (highlighted as red in Fig. 9.b) rather than `ddd` due to the data on the subordinate side not being cleared out when `rvalid` went low. No data should have been transmitted ($\mathtt{P}_{18}$), preventing the untrusted manager from receiving it.
Consequence: Hence, data remains on the bus, posing a potential security breach in the scenario where the untrusted manager can then access sensitive data from the benign manager.

6.6 Denial of Service (DoS)

eXpect reports counterexamples for ordering properties leading to stuck transactions because the buggy IP fails to assert a valid or ready signal subsequently blocking any forthcoming transactions.
Vulnerability 6: Failure on write transaction on XM_L. The attack scenario is Fig. 4.b with an AXI non-compliant subordinate and the manager is benign and being attacked.
Violation: Fig. 10.a shows a manager and subordinate completing a write transaction (from t=5 to t=7). The manager initiates a second write transaction (at t=10) (highlighted as red in Fig 10.a). However, the subordinate can prevent this second write transaction from occurring by keeping the `bvalid` signal high. The manager does not initiate another write transaction (evidenced by `awvalid` remaining

low after t=6 in Fig. 10.b). This is a result of the buggy subordinate not lowering its `bvalid` signal at the next positive edge of the clock. *Consequence*: Because of the subordinate's incorrect behavior, the manager suffers from a DoS preventing it from issuing any requests.

Vulnerability 7: Failure on write transaction on XS_F. This exploit arises from the same situation as Vulnerability 3 but has an additional consequence. The attack scenario is Fig. 4.a and is displayed in Fig. 8. After t=5 the address `010` is not processed since the previous transaction did not return any acknowledgment (`bvalid` never high), violating P_{20}, which leads to a DoS for both parties.

6.7 Effectiveness of Verification IPs

Although several vendors, such as Cadence and Synopsys, also offer verification IPs, we do not currently have access to them. Therefore, our focus is exclusively on AMD Xilinx IPs, which target their implementations ensuring that the transactions are functionally correct. AMD Xilinx offers a verification IP (VIP) [8] and a protocol checker (PC) [7]. The AXI VIP verifies managers and subordinates in custom RTL designs and the protocol checker core monitors for violations. The VIP incorporates the same assertions as the protocol checker. They have over 100 rules on all aspects of AXI-Full. While many of these rules entail value or range checks of the signals, there is some overlap with the properties of eXpect. eXpect flags all violations detected by the VIP for AXI-Lite and burst mode AXI-Full in the scope of our analysis. Moreover, eXpect identified 10 violated properties that were missed by AMD Xilinx, like the ordering properties and the advanced reset mechanism. We also tested if AMD Xilinx VIP can detect our exploits. Of the 8 violations among the 7 vulnerabilities, 5 are undetected and only 3 detected by the VIP. The VIP enforces timing checks between `cvalid` and `cready` signals to preempt potential stalling but does not cover all the eXpect ordering properties. Hence, we notice that the protocol checker is able to catch the dropped transaction vulnerabilities derived from the deviation of stability properties, like Vulnerability 1 and 2.a, but Vulnerability 2.b, which is based on a channel ordering requirement, is undetected. Since the data invalidation property is absent from the rules, none of the IPs are able to detect the stale data issues. As for memory corruption exploits, while vulnerability 3 triggered by the read data stability property is covered in the VIP, the more sophisticated concurrent read and write exploit managed to evade the VIP's detection. Vulnerability 6 triggers the VIP's alert because it incorporates timeout checks detecting this DoS. However, vulnerability 7 is not detected despite causing a DoS, indicating incomplete coverage of their DoS properties.

6.8 Fixing Vulnerabilities

We patch some of the identified vulnerabilities and retest the implementation with eXpect which does not report further counterexamples. Additionally, we synthesize the patched implementation on our FPGA to ensure that the fixes did not affect functionality. We focus on patching vulnerabilities required for exploits 2 (violation 2.b), 4 and 5, as these vulnerabilities were part of the undetected ones by the Xilinx Verification IPs. We do not patch the remaining 2 (#3 and #7) because they require extensive fixes at the transaction level. Table 6 shows the resource consumption of the patched implementations compared to the baseline. The fixes for the read channel ordering in AXI-Lite (P_{11}) and the additional write in AXI-Full do not require significant logic additions; constraining the address latching fixed the problem by limiting the number of memory writes to the transferred length. However, addressing data invalidation (P_{16}) necessitates extensive changes: allocating a new output register for memory read data, incorporating additional combinational logic to clear this register when the data is no longer valid. This lead to a significant increase in logic levels.

Table 6: Patch Results for Previously Listed XS_L and XM_L Vulnerabilities

Resources	AXI-Lite			AXI-Full	
	Baseline	Patch #2b	Patch #5	Baseline	Patch #4
FFs/LUTs	48/170	48/170	79/170	94/74	97/74
Logic Levels	1	1	5	5	3
Power (in W)	0.103	0.103	0.104	0.105	0.105

7 Related Work

Gisselquist Technology LLC. reported that a read transaction in XS_F cannot start as long as a write transaction is ongoing thus breaking functionality [16]. eXpect confirms this but further reports that concurrent read and write lead to a vulnerability as exploited in #3.

Fern et al. highlights hardware trojans attacks against Trustzone using the incomplete specifications of the AXI protocol like the undefined behavior of non active valid signals [19], [13]. It emphasizes the need for a thorough interconnect security verification. Formal verification approaches for confidentiality [33] or integrity [30] of SoC systems address a specific threat model (e.g., integration of malicious third-party IP). While these methods can exhaustively verify the system's security with respect to the corresponding threat model, they omit AXI implementation correctness, unlike eXpect.

Protocol compliance often focuses on functional correctness and not security [34]. Meza et al. uncovered and solved a security threat in a commercial-grade hardware root of trust leveraging property-based security verification and information flow tracking [32]. Restuccia et al. introduced access control systems for modern AXI-based platforms utilizing property-based verification [41], [42], [43]. Kastner et al. and Deutschbein et al. automated security property generation from existing CVEs [18],[27]. These are complementary and can be checked against eXpect.

On-chip interconnect can be extended to enforce availability[38], [39], [25], [31] or access control policies [23],[15]. eXpect applies beyond a specific interconnect, availability, and IP settings. Software analysis such as fuzzing [24], [45], [28], [26], symbolic execution, testing [50], and exploit generation are applied to hardware [49]. They can detect bugs in CPU cores or other IPs and also detect buggy or malicious hardware [48], [44], [17], [22]. This requires checking if the behavior is correct or not. Prior works rely on heuristics, IP-specific, and CVE-based oracles and then build the analysis frameworks to identify violations. eXpect properties can be used for such analyses of AXI behavior.

8 Conclusion

We presented eXpect, a tool that captures the AXI specification and security semantics such as data invalidation, address stability, channel ordering with 30 properties. eXpect detects 135 violations in 7 implementations for range of complexity from simple state-machines to state-of-the-art from AMD Xilinx. We pick 10 violations to build 7 exploits that demonstrate the security impact of eXpect-detected violations.

References

[1] AMD Xilinx. AXI-Full Xilinx manager implementation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi/axi4_ip.html.

[2] AMD Xilinx. AXI-Full Xilinx secondary implementation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi/axi4_ip.html.

[3] AMD Xilinx. AXI-Lite Xilinx manager implementation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi/axi4_ip.html.

[4] AMD Xilinx. AXI-Lite Xilinx secondary implementation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi/axi4_ip.html.

[5] AMD Xilinx. AXI Xilinx crossbar implementation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_interconnect.html.

[6] AMD Xilinx. AXI Xilinx Firewall. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi-firewall.html.

[7] AMD Xilinx. AXI Xilinx Protocol Checker. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_protocol_checker.html.

[8] AMD Xilinx. AXI Xilinx Verification IP. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi-vip.html.

[9] AMD Xilinx. Vivado. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[10] AMD Xilinx. Vivado Axi Peripheral Creation. Accessed: July 29, 2024. [Online]. Available: https://www.xilinx.com/video/hardware/creating-an-axi-peripheral-in-vivado.html.

[11] ARM. AMBA. Accessed: July 29, 2024. [Online]. Available: https://developer.arm.com/Architectures/AMBA.

[12] ARM. Cortex Core. Accessed: July 29, 2024. [Online]. Available: https://developer.arm.com/documentation/100095/0003/.

[13] ARM. Trustzone. Accessed: July 29, 2024. [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-a.

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

[15] Brunel, J., Pacalet, R., Ouaarab, S., and Duc, G. Secbus, a software/hardware architecture for securing external memories. In *IEEE International Conference on Mobile Cloud Computing, Services, and Engineering* (2014).

[16] Dan Gisselquist. Using a formal property file to verify an AXI-lite peripheral. Accessed: July 29, 2024. [Online]. Available: https://zipcpu.com/formal/2018/12/28/axilite.html.

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

[18] Deutschbein, C., Meza, A., Restuccia, F., Gregoire, M., Kastner, R., and Sturton, C. Toward hardware security property generation at scale. *IEEE S&P* (2022).

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

[20] Github. AXI-Lite secondary implementation. Accessed: July 29, 2024. [Online]. Available: https://github.com/mmxsrup/axi4-interface/blob/master/axi4-lite/axi_lite_slave.sv.

[21] Github. AXI-Lite secondary implementation. Accessed: July 29, 2024. [Online]. Available: https://gitlab.com/suoglu/axi-lite-slave/-/blob/main/Sources/ip_repo/axi_lite_slave_1.0/hdl/axi_lite_slave_v1_0.v.

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

[23] Huffmire, T., Prasad, S., Sherwood, T., and Kastner, R. Policy-driven memory protection for reconfigurable hardware. In *ESORICS* (2006).

[24] Hur, J., Song, S., Kwon, D., Baek, E., Kim, J., and Lee, B. Difuzzrtl: Differential fuzz testing to find cpu bugs. In *IEEE S&P* (2021).

[25] Jiang, Z., Yang, K., Fisher, N., Gray, I., Audsley, N. C., and Dong, Z. Axi-ic-rt: Towards a real-time axi-interconnect for highly integrated socs. *IEEE Transactions on Computers* (2022).

[26] Kande, R., Crump, A., Persyn, G., Jauernig, P., Sadeghi, A.-R., Tyagi, A., and Rajendran, J. TheHuzz: Instruction fuzzing of processors using Golden-Reference models for finding Software-Exploitable vulnerabilities. In *USENIX Security* (2022).

[27] Kastner, R., Restuccia, F., Meza, A., Ray, S., Fung, J., and Sturton, C. Automating hardware security property generation. In *DAC* (2022).

[28] Laeufer, K., Koenig, J., Kim, D., Bachrach, J., and Sen, K. Rfuzz: Coverage-directed fuzz testing of rtl on fpgas. In *ICCAD* (2018).

[29] Mattarei, C., Mann, M., Barrett, C., Daly, R. G., Huff, D., and Hanrahan, P. CoSA: Integrated Verification for Agile Hardware Design. In *FMCAD* (2018).

[30] Mehmedagić, D., Fadiheh, M. R., Müller, J., Antón, A. L. D., Stoffel, D., and Kunz, W. Design of Access Control Mechanisms in Systems-on-Chip with Formal Integrity Guarantees. In *USENIX Security* (2023).

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

[32] Meza, A., Restuccia, F., Oberg, J., Rizzo, D., and Kastner, R. Security verification of the opentitan hardware root of trust. *IEEE S&P* (2023).

[33] Müller, J., Fadiheh, M. R., Antón, A. L. D., Eisenbarth, T., Stoffel, D., and Kunz, W. A Formal Approach to Confidentiality Verification in SoCs at the Register Transfer Level. In *DAC* (2021).

[34] Nguyen, M. D., Thalmaier, M., Wedler, M., Bormann, J., Stoffel, D., and Kunz, W. Unbounded Protocol Compliance Verification Using Interval Property Checking With Invariants. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2008).

[35] PULP. AXI-Lite Pulp secondary implementation. Accessed: July 29, 2024. [Online]. Available: https://github.com/olofk/axi_node/blob/master/axi_regs_top.sv.

[36] Pulp Platform. Open Source Platform. Accessed: July 29, 2024. [Online]. Available: https://www.pulp-platform.org/.

[37] Questa. Questa property checking. Accessed: July 29, 2024. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/formal-verification/property-checking/.

[38] Restuccia, F., Biondi, A., Marinoni, M., and Buttazzo, G. Safely preventing unbounded delays during bus transactions in fpga-based soc. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2020).

[39] Restuccia, F., Biondi, A., Marinoni, M., Cicero, G., and Buttazzo, G. Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc. In *DAC* (2020).

[40] Restuccia, F., and Kastner, R. Cut and forward: Safe and secure communication for fpga system on chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[41] Restuccia, F., Meza, A., and Kastner, R. Aker: A design and verification framework for safe and secure soc access control. In *ICCAD* (2021).

[42] Restuccia, F., Meza, A., Kastner, R., and Oberg, J. A framework for design, verification, and management of soc access control systems. *IEEE Transactions on Computers* (2022).

[43] Siddiqui, F., Hagan, M., and Sezer, S. Pro-active policing and policy enforcement architecture for securing mpsocs. In *IEEE International System-on-Chip Conference (SOCC)* (2018).

[44] Sturton, C., Hicks, M., Wagner, D., and King, S. T. Defeating uci: Building stealthy and malicious hardware. In *IEEE S&P* (2011).

[45] Trippel, T., Shin, K. G., Chernyakhovsky, A., Kelly, G., Rizzo, D., and Hicks, M. Fuzzing hardware like software. In *USENIX Security* (2022).

[46] Xing, Y., Lu, H., Gupta, A., and Malik, S. Leveraging processor modeling and verification for general hardware modules. In *IEEE DATE* (2021).

[47] Xiphera. PRNG AXI peripheral. Accessed: July 29, 2024. [Online]. Available: https://xiphera.com/random-number-generation/pseudorandom-number-generation/.

[48] Yang, K., Hicks, M., Dong, Q., Austin, T., and Sylvester, D. A2: Analog malicious hardware. In *IEEE S&P* (2016).

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

[50] Zhang, R., and Sturton, C. Transys: Leveraging common security properties across hardware designs. In *IEEE S&P* (2020).