

Cross-party formal verification for hardware trust

by

Zhaoxiang Liu

B.S., University of Electronic Science and Technology of China, 2019

M.S., Rutgers University, 2021

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Mike Wieggers Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2026

Abstract

Modern computing systems increasingly depend on hardware as the foundation of system trust. Yet establishing that trust has become substantially more difficult in a semiconductor ecosystem defined by extensive third-party IP reuse, complex system-on-chip integration, tight hardware–software coupling, and globally distributed design, manufacturing, and distribution processes. In such an environment, security threats are no longer confined to isolated implementation bugs within a single module. Instead, vulnerabilities may arise from malicious RTL modifications, unintended interactions among integrated IPs, exploitable dependencies between software and hardware, counterfeit devices introduced into the supply chain, and confidentiality risks during pre-purchase IP evaluation and exchange. Although formal verification remains one of the most powerful approaches for establishing design assurance, existing methods often operate at only a single abstraction level, lose important RTL semantics through synthesis or translation, or implicitly assume trusted relationships among the parties involved. These limitations motivate the need for a broader framework capable of reasoning about hardware trust across both technical abstraction boundaries and organizational trust boundaries.

This dissertation addresses that need through a unified perspective on cross-party formal verification for hardware trust. The central premise is that hardware trust must be established progressively across expanding boundaries: first among interacting IPs within an integrated system, then across the hardware–software interface, next across the physical supply chain, and finally across mutually untrusted parties engaged in IP exchange. Rather than treating these settings as unrelated problems, this dissertation shows that they are connected by a common challenge: security-relevant behavior propagates across components, clock cycles, abstraction layers, and stakeholders in ways that conventional verification frameworks do not fully capture. To address this challenge, the dissertation combines formal reasoning, static analysis, causal modeling, RTL security instrumentation, and cryptographic protocols into a coherent methodology for analyzing, verifying, and preserving trust in modern hardware ecosystems.

The dissertation first focuses on trust within integrated SoC platforms, where malicious behavior may emerge not from the standalone operation of a single IP core, but from its interaction with other

components through shared buses and interconnects. To address this challenge, it develops If-Tracker, an RTL-level static information-flow verification framework that operates directly on hardware descriptions by parsing them into abstract syntax trees and constructing cycle-accurate data-flow graphs. Unlike approaches that merely determine whether a path exists, If-Tracker also extracts the conditions under which information can propagate along that path, enabling the detection of subtle threats such as bus eavesdropping, covert leakage paths, and malicious interconnect manipulation, including TrustZone-related tampering. Because the analysis preserves RTL structure and maps suspicious flows back to source code locations, it improves both the precision and interpretability of SoC-scale security verification. In doing so, this part of the dissertation demonstrates that preserving high-level RTL semantics is essential for scalable and explainable formal analysis of integrated hardware systems.

Building on this foundation, the dissertation then extends the notion of trust across the hardware–software boundary, where many hardware vulnerabilities become relevant only when they can be triggered by particular software instructions or firmware behaviors. Existing approaches often struggle in this setting because they either translate hardware into software representations that lose structural fidelity or rely on verification tools that are not designed to expose causal relationships between software actions and hardware failures. To overcome these limitations, the dissertation proposes Microscope, a framework based on a Hardware Structural Causal Model (HW-SCM) that captures multi-cycle causal dependencies among software-controlled inputs, internal hardware signals, and security-critical outcomes. Through a dedicated domain-specific language and SMT-based incremental solving, Microscope infers instruction patterns capable of activating hardware bugs and vulnerabilities. This allows verification to move beyond simple property checking toward causal explanation: the framework not only determines whether a vulnerability is reachable, but also identifies how it can be exercised from the software side. As a result, this work bridges formal verification, exploitability analysis, and test generation, offering a practical way to reason about software-exploited hardware weaknesses.

The dissertation next broadens the scope of hardware trust from logical behavior to device authenticity in the supply chain. Even when a design has been thoroughly verified, its value as a root of trust is undermined if counterfeit or tampered integrated circuits can be inserted during fabrication, distribution, or deployment. To address this problem, the dissertation presents Wing-RTL, a lightweight RTL-level attestation framework that automatically embeds watermarks into finite state machines and integrates them with an on-chip watermark controller. This enables a secure attesta-

tion protocol through which a verifier can confirm the authenticity of a device without exposing the internal watermark itself. By emphasizing automation, minimal hardware overhead, and compatibility with existing SoC infrastructure, Wing-RTL shows that trust must be preserved not only in design logic, but also in the provenance and authenticity of the physical device that implements that logic. In this sense, the dissertation expands formal hardware assurance beyond design-time verification into the broader lifecycle of hardware deployment and supply-chain security.

Finally, the dissertation considers a zero-trust setting in which the hardware asset itself is the subject of commercial exchange. In modern semiconductor development, IP vendors and IP users often need to interact before a legal agreement is finalized: the user wants to verify that an IP core satisfies required properties, while the vendor must protect the design from disclosure. At the same time, the user may also wish to keep their intended application, verification property, or purchasing interest confidential. To address these conflicting requirements, the dissertation proposes BlindMarket, an end-to-end framework for verifiable, confidential, and traceable IP core distribution in zero-trust environments. BlindMarket combines customized hardware-to-formula translation, oblivious design selection, secure two-party computation, privacy-preserving SAT solving, hardware-aware optimization through hw-ppSAT, and ledger-based authorization for traceability. This framework enables users to verify design properties before acquisition without learning the full IP implementation, while also preventing the vendor from learning the user’s selection and verification intent. By doing so, the dissertation extends formal verification into a new role: not only as a technique for proving correctness, but also as a mechanism for enabling secure and privacy-preserving interaction among mutually untrusted stakeholders in the hardware ecosystem.

Taken together, these contributions advance a single overarching argument: hardware trust is no longer a property that can be established at one level, by one party, or at one stage of the design flow. It must instead be verified across the full chain of interactions through which hardware is described, integrated, exercised, authenticated, exchanged, and ultimately relied upon. By progressing from inter-IP verification to hardware–software causal inference, to device attestation, and finally to privacy-preserving IP distribution, this dissertation develops a layered methodology for establishing trust in contemporary hardware systems.

Cross-party formal verification for hardware trust

by

Zhaoxiang Liu

B.S., University of Electronic Science and Technology of China, 2019

M.S., Rutgers University, 2021

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Mike Wiegiers Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2026

Approved by:

Major Professor
Xiaolong Guo

Copyright

© Zhaoxiang Liu 2026.

Abstract

Modern computing systems increasingly depend on hardware as the foundation of system trust. Yet establishing that trust has become substantially more difficult in a semiconductor ecosystem defined by extensive third-party IP reuse, complex system-on-chip integration, tight hardware–software coupling, and globally distributed design, manufacturing, and distribution processes. In such an environment, security threats are no longer confined to isolated implementation bugs within a single module. Instead, vulnerabilities may arise from malicious RTL modifications, unintended interactions among integrated IPs, exploitable dependencies between software and hardware, counterfeit devices introduced into the supply chain, and confidentiality risks during pre-purchase IP evaluation and exchange. Although formal verification remains one of the most powerful approaches for establishing design assurance, existing methods often operate at only a single abstraction level, lose important RTL semantics through synthesis or translation, or implicitly assume trusted relationships among the parties involved. These limitations motivate the need for a broader framework capable of reasoning about hardware trust across both technical abstraction boundaries and organizational trust boundaries.

This dissertation addresses that need through a unified perspective on cross-party formal verification for hardware trust. The central premise is that hardware trust must be established progressively across expanding boundaries: first among interacting IPs within an integrated system, then across the hardware–software interface, next across the physical supply chain, and finally across mutually untrusted parties engaged in IP exchange. Rather than treating these settings as unrelated problems, this dissertation shows that they are connected by a common challenge: security-relevant behavior propagates across components, clock cycles, abstraction layers, and stakeholders in ways that conventional verification frameworks do not fully capture. To address this challenge, the dissertation combines formal reasoning, static analysis, causal modeling, RTL security instrumentation, and cryptographic protocols into a coherent methodology for analyzing, verifying, and preserving trust in modern hardware ecosystems.

The dissertation first focuses on trust within integrated SoC platforms, where malicious behavior may emerge not from the standalone operation of a single IP core, but from its interaction with other

components through shared buses and interconnects. To address this challenge, it develops If-Tracker, an RTL-level static information-flow verification framework that operates directly on hardware descriptions by parsing them into abstract syntax trees and constructing cycle-accurate data-flow graphs. Unlike approaches that merely determine whether a path exists, If-Tracker also extracts the conditions under which information can propagate along that path, enabling the detection of subtle threats such as bus eavesdropping, covert leakage paths, and malicious interconnect manipulation, including TrustZone-related tampering. Because the analysis preserves RTL structure and maps suspicious flows back to source code locations, it improves both the precision and interpretability of SoC-scale security verification. In doing so, this part of the dissertation demonstrates that preserving high-level RTL semantics is essential for scalable and explainable formal analysis of integrated hardware systems.

Building on this foundation, the dissertation then extends the notion of trust across the hardware–software boundary, where many hardware vulnerabilities become relevant only when they can be triggered by particular software instructions or firmware behaviors. Existing approaches often struggle in this setting because they either translate hardware into software representations that lose structural fidelity or rely on verification tools that are not designed to expose causal relationships between software actions and hardware failures. To overcome these limitations, the dissertation proposes Microscope, a framework based on a Hardware Structural Causal Model (HW-SCM) that captures multi-cycle causal dependencies among software-controlled inputs, internal hardware signals, and security-critical outcomes. Through a dedicated domain-specific language and SMT-based incremental solving, Microscope infers instruction patterns capable of activating hardware bugs and vulnerabilities. This allows verification to move beyond simple property checking toward causal explanation: the framework not only determines whether a vulnerability is reachable, but also identifies how it can be exercised from the software side. As a result, this work bridges formal verification, exploitability analysis, and test generation, offering a practical way to reason about software-exploited hardware weaknesses.

The dissertation next broadens the scope of hardware trust from logical behavior to device authenticity in the supply chain. Even when a design has been thoroughly verified, its value as a root of trust is undermined if counterfeit or tampered integrated circuits can be inserted during fabrication, distribution, or deployment. To address this problem, the dissertation presents Wing-RTL, a lightweight RTL-level attestation framework that automatically embeds watermarks into finite state machines and integrates them with an on-chip watermark controller. This enables a secure attesta-

tion protocol through which a verifier can confirm the authenticity of a device without exposing the internal watermark itself. By emphasizing automation, minimal hardware overhead, and compatibility with existing SoC infrastructure, Wing-RTL shows that trust must be preserved not only in design logic, but also in the provenance and authenticity of the physical device that implements that logic. In this sense, the dissertation expands formal hardware assurance beyond design-time verification into the broader lifecycle of hardware deployment and supply-chain security.

Finally, the dissertation considers a zero-trust setting in which the hardware asset itself is the subject of commercial exchange. In modern semiconductor development, IP vendors and IP users often need to interact before a legal agreement is finalized: the user wants to verify that an IP core satisfies required properties, while the vendor must protect the design from disclosure. At the same time, the user may also wish to keep their intended application, verification property, or purchasing interest confidential. To address these conflicting requirements, the dissertation proposes BlindMarket, an end-to-end framework for verifiable, confidential, and traceable IP core distribution in zero-trust environments. BlindMarket combines customized hardware-to-formula translation, oblivious design selection, secure two-party computation, privacy-preserving SAT solving, hardware-aware optimization through hw-ppSAT, and ledger-based authorization for traceability. This framework enables users to verify design properties before acquisition without learning the full IP implementation, while also preventing the vendor from learning the user’s selection and verification intent. By doing so, the dissertation extends formal verification into a new role: not only as a technique for proving correctness, but also as a mechanism for enabling secure and privacy-preserving interaction among mutually untrusted stakeholders in the hardware ecosystem.

Taken together, these contributions advance a single overarching argument: hardware trust is no longer a property that can be established at one level, by one party, or at one stage of the design flow. It must instead be verified across the full chain of interactions through which hardware is described, integrated, exercised, authenticated, exchanged, and ultimately relied upon. By progressing from inter-IP verification to hardware–software causal inference, to device attestation, and finally to privacy-preserving IP distribution, this dissertation develops a layered methodology for establishing trust in contemporary hardware systems.

Table of Contents

List of Figures	xiv
List of Tables	xvii
Acknowledgements	xix
Preface	xxi
1 Introduction	1
1.1 Motivation.	2
1.2 Contribution	3
1.2.1 Inter-IP Malicious Modification Detection through Static Information Flow Tracking	3
1.2.2 Microscope: Causal Inference Crossing the Hardware and Software Boundary	4
1.2.3 WING-RTL: Attestation through Automatic FSM Watermark Generation and Integration on RTL	4
1.2.4 BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero-Trust Settings	5
1.3 Dissertation Organization	5
2 Inter-IP Malicious Modification Detection through Static Information Flow Tracking	7
2.1 Introduction	7
2.2 Background	8
2.2.1 Abstract Syntax Tree	8
2.2.2 Malicious Modifications in SoC Platform	8
2.2.3 Security Formal Verification on SoC	9

2.2.4	Formal Verification Tools for Security Property Checking	10
2.3	Methodology	11
2.3.1	Adversarial Model	11
2.3.2	Workflow	11
2.3.3	Information Flow Policy	12
2.3.4	Locating Threats in RTL Code	13
2.3.5	Case Study 1: Eavesdropping Attack	14
2.3.6	Case Study 2: TrustZone Attack	16
2.3.7	Case Study 3: Counter-based Trojan Attack	17
2.4	Comparison with Commercial Tools and Discussion	18
2.5	Conclusion	19
3	Microscope: Causal Inference Crossing the Hardware and Software Boundary	20
3.1	Introduction	20
3.2	Background	21
3.2.1	Structural Causal Model	21
3.2.2	SMT Solving.	21
3.3	Hardware Structural Causal Model	22
3.3.1	HW-SCM definition	22
3.3.2	HW-SCM Graph Example	23
3.4	Microscope	24
3.4.1	Microscope Overview	24
3.4.2	Domain-Specific Language for HW-SCM	25
3.4.3	Assertion Development	27
3.5	Experimental Results	27
3.5.1	Experiment Settings	27
3.5.2	Threat Model and Heuristic Assertions Development	28
3.5.3	Results and Analysis	30
3.6	Related Work	31

3.6.1	Comparison with Existing Works	32
3.6.2	Relationship between Information Flow Tracking and Microscope	33
3.7	Conclusion and Future Work	33
4	WING-RTL: Attestation through Automatic FSM Watermark Generation and Integration on RTL	35
4.1	Introduction	35
4.2	Background	36
4.2.1	Watermarking Schemes	36
4.2.2	Attestation	37
4.3	WING-RTL Design	37
4.3.1	Adversarial Model	37
4.3.2	Primitives for Operation	38
4.3.3	Protocol Operation	38
4.4	WING-RTL Implementation	40
4.4.1	Addition of the Watermark	40
4.4.2	Enrollment	41
4.4.3	Testing Platform	41
4.5	Experimental Results	44
4.5.1	Overhead of Watermarking Scheme	44
4.5.2	Scheme Resiliency	45
4.6	Related Works and Comparison	47
4.6.1	Watermarking Scheme	47
4.6.2	Hardware Attestation	47
4.6.3	Comparison with Previous Work	48
4.7	Conclusion	48
5	BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero- Trust Settings	49

5.1	Introduction	49
5.2	Preliminaries	51
5.2.1	SAT-based Hardware Verification	51
5.2.2	Cryptographic Tools	51
5.3	Related Work	53
5.4	Threat Model	54
5.5	BlindMarket System Design	54
5.5.1	BlindMarket Workflow	54
5.5.2	Customized IP and Property Translations	57
5.5.3	Oblivious Design Selection	59
5.5.4	Secure Design Verification	59
5.5.5	hw-ppSAT	61
5.5.6	License Authorization	64
5.6	Design Pruning	65
5.7	Experiment	66
5.7.1	Performance Evaluation	66
5.7.2	Communication Overhead	69
5.7.3	Comparison with Non-private Baseline	70
5.8	Conclusion & Discussion	72
6	Conclusion and Future Work	73
6.1	Conclusion	73
6.2	Future Work	74
	Bibliography	76

List of Figures

2.1	Workflow of the proposed If-Tracker framework.	12
2.2	AXI4-Lite SoC benchmark	14
2.3	Propagation conditions of SHA256	15
2.4	Propagation conditions of RAM	15
3.1	Graph based HW-SCM	22
3.2	Working procedure of the proposed Microscope framework	24
3.3	Domain Specific Language	25
4.1	WING-RTL framework: designed ICs are watermarked and enrolled into a verification server. At any point during the supply chain, anybody can follow the attestation protocol to verify the authenticity of the IC. Adversaries may attempt to inject functional counterfeits into the supply chain or extract ICs from it for reverse engineering.	36
4.2	Overview of the live phase of the attestation protocol. The prover in our scheme consists of the watermarked FSM as well as the watermark controller within the integrated circuit.	39
4.3	Finite State Machine watermarking scheme. As to not interfere with the regular operation of the FSM we use a separate set of watermarking states.	41
4.4	Simplified view of the SoC used as test platform. We add the watermark to the finite state machine of the AES module.	42

5.1	End-to-end workflow of secure IP verification and licensing in BlindMarket, organized into four phases. (1) In the preprocessing phase, the vendor encodes each IP core into CNF and masks it using randomness, while the user encodes the verification property. (2) During oblivious design selection, the user retrieves the masked CNF of the selected design via 1-out-of- N OT, without revealing the selection to the vendor. (3) In the secure design verification phase, the vendor and user engage in a 2PC protocol to reconstruct the selected design and jointly verify it against the property using hw-ppSAT . (4) Finally, if verification succeeds, the vendor issues a certificate bound to the authorized user and selected IP, publishing it to the ledger for traceability.	55
5.2	Secure hardware design verification workflow in BlindMarket. The vendor masks each CNF-encoded design using random matrices, and the user selects a target design via oblivious transfer (OT), receiving only the masked version. The vendor and user then run a 2PC protocol to reconstruct the selected design and combine it with the user's property to form $\psi = \phi_i \wedge \neg\varphi$. The satisfiability of ψ is evaluated using hw-ppSAT under 2PC, revealing only the final result to both parties while preserving mutual confidentiality.	60
5.3	Illustration of two signal prioritization principles. In (a), the control signal ctrl1 is prioritized during branching, since assigning ctrl1 to false renders the values of a and c irrelevant to the satisfiability of the encoded design. In (b), for two control signals ctrl11 and ctrl12 , ctrl12 is prioritized, because once ctrl12 is assigned false , ctrl11 no longer influences the evaluation.	62
5.4	When the circuit is unrolled, later-stage control signals dominate earlier-stage ones. Thus enable₂ dominates reset₁ . Based on Principle 2, reset₁ dominates enable₁ . Consequently, the priority chain becomes enable₂ $\succ$ reset₁ $\succ$ enable₁	63

5.5	Illustrates the performance ratio between (generic) ppSAT and (our) hw-ppSAT across various (pruned) benchmarks. Each segment represents a specific design, with the outer boundary (ratio = 1.0) indicating the baseline solving time of ppSAT . The shaded region corresponds to hw-ppSAT ; a smaller area indicates a greater speedup. As shown, most designs experience a substantial reduction in solving time, with normalized ratios significantly below 1.0. In the best-case scenario, our method achieves a remarkable speedup with a ratio as low as 0.0098.	69
5.6	Communication overhead analysis of the proposed protocol. (a) shows the cost of the 1-out-of- 2^{depth} OT protocol as a function of design size, while (b) reports the 2PC overhead per giant step measured by garbled circuit size. Both results demonstrate near-linear scalability with respect to the design size ($n_{var} \times n_{cls}$).	71
5.7	Runtime of the non-private baseline implementation of hw-ppSAT without 2PC. The results indicate that solver runtime is largely dominated by the number of executed steps.	71

List of Tables

2.1	Security properties for addressing threats. In the table, BM: Bus Manipulation	13
2.2	Detection results and comparison with commercial tools. In the table, PE: path exists, CE: conditions extracted, PNE: path not exists, CC: Connectivity Checking, FSV: Formal Security Verification, SPV: Security Path Verification. All times given in seconds, RL: Reach Time Limit.	17
3.2	Experiment Result	30
3.3	Comparison with existing work on or1200 testbench	31
3.1	Vulnerabilities description and code example	34
4.1	Watermark overhead in select state machines when targeting an Artix 7 XCA100TCGS324 FPGA.	44
4.2	Table comparing different state-of-the-art IP protection schemes based on watermarking scheme, verification scheme, and resiliency analysis against IC counterfeit. WING-RTL watermarks IC at the pre-synthesis stage and leverages attestation to verify the provenance of the IC against Non-invasive Reverse engineering (NI-RE) attacks and Invasive Reverse Engineering (I-RE) attacks.	44
5.1	Existing methods cannot provide privacy protection for formal-based IP verification without a third party.	53
5.2	Soft IP metadata in each record.	64
5.3	Design, Line of Code, Abstract Description and Assertion. We expect UNSAT for the negation of the assertion property conjoined with the design, and SAT for the counterexample derived from the conjunction of the cover property and the design. . .	67

5.4 We summarize the end-to-end execution time of BlindMarket for 13 IP designs in both their original and pruned forms. **Var/Cls** denotes the number of variables and clauses in the verification formula ψ . **Design Size** represents the size of the encoded design transferred during OT, measured in 128-bit blocks. **OT(1/32)** reports the execution time of 1-out-of-32 OT. **Demask** measures the time required to reconstruct the selected design within the 2PC (Equation 5.1). For the SAT solving stage, **DLIS** records the number of giant steps taken under the standard **ppSAT** DLIS heuristic, and **ppSAT** reports the corresponding solving time. Similarly, **Ctrl** denotes the number of giant steps taken under **hw-ppSAT**, while **hw-ppSAT** reports the solving time using the control-flow-guided heuristic. The results show that combining design pruning with **hw-ppSAT** achieves faster verification times on 12 of 13 benchmarks. In the table, **yellow cells** represent time improvements from design pruning, and **green cells** represent time improvements from heuristic guidance. 68

Acknowledgments

I would like to express my sincere gratitude to everyone who has supported and contributed to this dissertation.

First and foremost, I am deeply thankful to my advisor, Dr. Xiaolong Guo. Reflecting on my academic journey, I truly appreciate his patience, generosity, and guidance. When I began this path, I was still developing my research skills, and he gave me the time, trust, and encouragement to grow into an independent scholar. His insightful feedback and high expectations consistently challenged me to think more critically and work with greater rigor. I feel incredibly fortunate to have had Professor Guo as my advisor, and I will always value the confidence he placed in me.

I also extend my sincere appreciation to my dissertation committee members, Dr. Caterina Scoglio, Dr. Hongyu Wu, Dr. Orlando Arias, and Dr. Don Gruenbacher, for their valuable feedback, thoughtful suggestions, and continued support. Their perspectives have significantly strengthened this work, and I am grateful for the time and care they dedicated to this process.

I am especially thankful to my colleagues, particularly Dr. Orlando Arias and Dr. Ning Luo. This work would not have been possible without their guidance, support, and willingness to share their expertise. I am particularly grateful to Dr. Orlando Arias, one of my earliest collaborators. His patience, professionalism, and encouragement were invaluable during the early stages of my research, when I was still learning how to navigate academic work. As an early mentor, he not only provided technical insight but also gave me confidence as I grew into this field. I also sincerely appreciate his willingness to serve as my outside committee member despite his many commitments.

I am equally grateful to Dr. Ning Luo for her rigor, professionalism, and support throughout our collaboration. In the later stages of my research, her careful feedback and sharp attention to detail helped me identify weaknesses, refine my ideas, and improve the overall quality of this dissertation. I deeply appreciate her honesty, precision, and commitment to high scholarly standards.

I would like to express my heartfelt thanks to my family, especially my mother. Although she could not assist me academically, she has always been my emotional anchor. She is gentle, caring, and unwaveringly supportive. During moments of frustration and uncertainty, she comforted me and gave

me strength; during moments of success, she reminded me to stay grounded and humble. Her love and understanding sustained me through many challenges, and I will always be grateful for her presence in my life.

I would also like to thank Dr. Jie Liu and Dr. Shen Yan, who first introduced me to academic research and served as my earliest mentors. Their guidance laid the foundation for my academic journey. I am also grateful to Dr. Zhenyuan Liu for her companionship and support, and for being part of such a meaningful and enriching experience.

This dissertation is not merely an endpoint, but a reflection of the community that supported my growth—academically, emotionally, and personally. To everyone who has contributed along the way, I offer my deepest thanks for helping turn this journey into reality.

Preface

This dissertation, “Cross-Party Formal Verification for Hardware Trust,” is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering at Kansas State University. The research presented herein was conducted under the supervision of Professor Xiaolong Guo.

To the best of my knowledge, this work is original, except where appropriate acknowledgments and references have been provided. The dissertation is based on my previously published or submitted work in peer-reviewed conferences and journals, and is incorporated here with appropriate attribution.

Publications Related to This Dissertation

1. Zhaoxiang Liu, Orlando Arias, Weimin Fu, Yier Jin, and Xiaolong Guo, “Inter-IP Malicious Modification Detection through Static Information Flow Tracking,” in *Design, Automation and Test in Europe Conference (DATE)*, 2022, pp. 600–603.
2. Orlando Arias, Zhaoxiang Liu, Xiaolong Guo, Yier Jin, and Shuo Wang, “RTSEC: Automated RTL Code Augmentation for Hardware Security Enhancement,” in *Design, Automation and Test in Europe Conference (DATE)*, 2022, pp. 596–599.
3. Zhaoxiang Liu, Kejun Chen, Dean Sullivan, Orlando Arias, Raj Gautam Dutta, Yier Jin, and Xiaolong Guo, “Microscope: Causality Inference Crossing the Hardware and Software Boundary from Hardware Perspective,” in *Proceedings of the 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 933–938.
4. Zhaoxiang Liu, Orlando Arias, Dean Sullivan, Raj Dutta, and Xiaolong Guo, “A Temporal Causality Model for SoC-Scale Security Formal Verification at the Hardware-Software Boundary,” in *Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, 2024.
5. Ruochen Dai, Zhaoxiang Liu, Orlando Arias, Xiaolong Guo, and Tuba Yavuz, “DTjRTL: A Configurable Framework for Automated Hardware Trojan Insertion at RTL,” in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2024.

6. Ruochen Dai, Zhaoxiang Liu, Orlando Arias, Xiaolong Guo, and Tuba Yavuz, “Evaluating the Effectiveness of Hardware Trojan Detection Approaches at RTL,” in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2025, pp. 250–260.
7. Zhaoxiang Liu, Ning Luo, Samuel Judson, Raj Gautam Dutta, Xiaolong Guo, and Mark Santolucito, “Poster: BlindMarket: A Trustworthy Chip Designs Marketplace for IP Vendors and Users,” in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024, pp. 5048–5050.
8. Zhaoxiang Liu, Samuel Judson, Raj Dutta, Mark Santolucito, Xiaolong Guo, and Ning Luo, “BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero-Trust Settings,” in *2026 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2026.

Chapter 1

Introduction

Modern computing systems increasingly rely on hardware as the root of trust, yet establishing that trust has become substantially more difficult in a semiconductor ecosystem shaped by extensive third-party IP reuse, large-scale system-on-chip (SoC) integration, and fast product cycles^{1;2}. Formal verification remains one of the most rigorous approaches for establishing design assurance, and recent progress in SAT-based reasoning has made exhaustive analysis increasingly attractive for hardware security evaluation³⁻⁷. However, the trust problem faced by modern hardware is no longer confined to a single module or a single stage of the lifecycle. Security-relevant behavior may emerge only after IP integration, under particular software interactions, after fabrication and distribution, or when mutually untrusted parties attempt to evaluate and exchange designs.

At the same time, existing verification workflows often reason about hardware through representations that are either too low-level or too far removed from the original RTL. Some approaches depend on synthesized netlists or other transformed representations whose recovery to the original structure is difficult^{8;9}; others translate RTL designs into software-like descriptions and then rely on symbolic execution frameworks such as KLEE, which can weaken fidelity to hardware timing and structure¹⁰⁻¹². Commercial formal platforms provide powerful property checking capabilities, but they are typically optimized for functional correctness or path validation rather than for explaining cross-layer exploitability, authenticating fabricated devices, or enabling secure verification across distrustful organizational boundaries¹³⁻¹⁵. These limitations suggest that hardware trust should be treated not as a local verification task, but as a cross-boundary assurance problem.

1.1 Motivation.

The central motivation of this dissertation is that the boundary at which hardware must be trusted keeps expanding. First, in an SoC assembled from heterogeneous third-party components, malicious behavior may not appear inside any single IP when analyzed in isolation, but instead arise from interactions through buses, interconnects, and shared control logic. Existing taint-based and path-based approaches are valuable, yet they often emphasize reachability while leaving propagation conditions and malicious inter-IP coordination insufficiently characterized^{14–20}. This motivates an RTL-aware method that preserves structural semantics and reasons about how and under what conditions information propagates across integrated hardware. Yet eliminating malicious inter-IP interaction does not by itself establish trust, because vulnerabilities may still remain dormant until exercised from the software side.

Second, even when inter-IP structure is understood, many security-relevant hardware weaknesses become consequential only when they can be triggered by software or firmware. The growing impact of transient execution and related microarchitectural vulnerabilities shows that hardware assurance cannot stop at RTL connectivity or property reachability alone. Existing co-verification workflows often require translating hardware into software or manually mining counterexamples from formal tools, making it difficult to recover complete, replayable, multi-cycle trigger patterns^{9–11;13;15;21}. This motivates a causal perspective that can relate software-visible inputs to internal hardware states and security outcomes across time. And even a fully analyzed design cannot serve as a trustworthy root if the physical device that reaches deployment is not genuine.

Third, logical correctness alone is insufficient if counterfeit or cloned integrated circuits can undermine the value of a verified design after fabrication and distribution. Real-world incidents have shown that end users may bear the consequences of counterfeit devices without even realizing that a fake component has entered the supply chain²². Prior approaches to attestation and identification, including watermark-based methods, side-channel fingerprints, and FPGA-oriented schemes, demonstrate important directions but still face scalability, deployability, or platform-specific limitations^{23–25}. This motivates an attestation mechanism that can be integrated at RTL with low overhead and reused throughout the supply chain. Beyond deployed devices, however, the question of trust appears even earlier, during pre-purchase IP evaluation and exchange.

Finally, formal verification is attractive because it allows an IP user to check whether a design satisfies desired properties before acquisition, but performing that verification naively requires exposing the vendor’s proprietary design. At the same time, the property itself may reveal the user’s architecture, intent, or purchasing interest. The resulting privacy risk is therefore bidirectional, and well-known protection mechanisms such as watermarking, obfuscation, or legal and standardized protections do not fully solve confidential pre-purchase verification in a zero-trust setting^{26–36}. Meanwhile, missing provenance and traceability can create security risks that persist after distribution³⁷. This motivates a privacy-preserving verification and authorization framework that can jointly support verifiability, confidentiality, and traceability without relying on a trusted third party³⁸.

Guided by this progressive expansion of trust boundaries, this dissertation develops a unified view of cross-party formal verification for hardware trust. It proceeds from trust within integrated SoCs, to trust across the hardware–software boundary, to trust in the authenticity of deployed devices, and finally to trust in zero-trust IP exchange. Concretely, the dissertation first presents If-Tracker, an RTL-level information-flow framework for detecting malicious inter-IP interactions. It then introduces Microscope, which infers software instruction patterns capable of triggering hardware vulnerabilities. Next, it presents Wing-RTL, a lightweight RTL-level attestation framework for counterfeit detection. Finally, it proposes BlindMarket, which enables confidential, verifiable, and traceable IP distribution before acquisition. Taken together, these works argue that trustworthy hardware cannot be established at only one abstraction level, by only one party, or at only one stage of the lifecycle; it must instead be constructed progressively across integration, execution, deployment, and exchange.

1.2 Contribution

1.2.1 Inter-IP Malicious Modification Detection through Static Information Flow Tracking

- We present a scalable framework to statically identify threats in SoC platforms at RT-Level. By parsing HDL into an AST, we extract more information that can be used for tracking, identification, and localization of anomalies.
- The introduction of If-Tracker, a CAD tool which can parse Verilog HDL and automates security checks on the design. If-Tracker models an SoC into a data-flow graph and automatically infers

interactions between IP cores. If-Tracker uses this to perform checks on user-specified security properties.

- We demonstrate the efficacy of If-Tracker using an SoC benchmark equipped with AMBA-AXI bus which includes a series of concealed Trojans. This benchmark will be released as open-source to the hardware security community in³⁹. We compare the If-Tracker to commercial tools from Cadence and Synopsys.

1.2.2 Microscope: Causal Inference Crossing the Hardware and Software Boundary

- The proposed Microscope presents HW-SCM in a self-developed DSL, which models a hardware-software system from the hardware side. It effectively addresses scalability and efficiency issues by eliminating the need to convert hardware to software or low-level hardware description.
- Our approach identifies causality to determine whether a hardware threat could result from any software events. If such a relationship exists, the inferred instruction patterns can serve as a valuable resource for both patching hardware bugs and generating test input. Conversely, if no such relationship is found, the hardware vulnerability can be deemed a non-critical security threat, allowing for the conservation of resources.
- Through extensive experimentation, we demonstrate that Microscope is capable of inferring the causality of a wide range of vulnerabilities and bugs found in SoC-level benchmarks. In line with this, we have developed a corresponding EDA tool that will be made publicly available upon the paper’s acceptance.

1.2.3 WING-RTL: Attestation through Automatic FSM Watermark Generation and Integration on RTL

- The introduction of WING-RTL, a secure integrated circuit attestation scheme based on FSM watermarking to cheaply identify counterfeits. Unique to WING-RTL is the ability to automatically add a watermark to a design and connect it to a configurable watermark controller. We address the existing challenges of scalability and cost of testing. A central goal in WING-RTL is to make attestation cheap and accessible, even to end users.

- An example implementation of WING-RTL in an Arm Cortex-M3 based microcontroller showing a minuscule integration area overhead of 1.5%.
- An evaluation of WING-RTL in terms of security. We demonstrate that the attestation scheme is secure under a powerful attacker with physical access to the device.

1.2.4 BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero-Trust Settings

- We propose an end-to-end “zero-trust” framework, BlindMarket, that allows the IP user and the vendor to establish a trusted IP trading process to privately select and verify IP cores of interest before the transaction takes place without relying on any trusted third-party.
- We turn the idea of privacy-preserving verification into a practical end-to-end hardware verification framework with tailored formula generation and hardware-focused optimizations.
- We design a cryptographic ledger-based IP ownership auditing mechanism that complements BlindMarket by enabling post-trade validation of provenance and ownership. This mechanism provides verifiable traceability of IP cores across transactions.
- We implement our framework and evaluate it on real-world IP cores. The results demonstrate that our approach efficiently verifies functional properties, achieving performance improvements in 12 out of 13 benchmarks. Its effectiveness is further illustrated in Figure 5.5.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 begins at the system-integration level and presents If-Tracker, an RTL-level static information-flow tracking framework for detecting malicious inter-IP modifications in SoC platforms by analyzing propagation paths together with their enabling conditions. Chapter 3 extends the trust boundary across the hardware–software interface through Microscope, which introduces a Hardware Structural Causal Model (HW-SCM) to infer software instruction patterns capable of triggering underlying hardware vulnerabilities. Chapter 4 broadens the discussion from design-time verification to post-fabrication authenticity by presenting Wing-RTL, a lightweight RTL-level attestation framework that automatically embeds watermarking logic into finite

state machines and enables counterfeit detection throughout the hardware supply chain. Chapter 5 further considers hardware trust in a zero-trust commercial setting and introduces BlindMarket, an end-to-end framework for verifiable, confidential, and traceable IP core distribution through privacy-preserving formal verification, oblivious design selection, and ledger-based license authorization. Finally, Chapter 6 concludes the dissertation by summarizing the key contributions of this work and discussing several directions for future research.

Chapter 2

Inter-IP Malicious Modification Detection through Static Information Flow Tracking

2.1 Introduction

The semiconductor industry has an urgent demand for automated methods to detect and evaluate vulnerabilities in System-on-Chip (SoC) at the design stage¹. Interestingly, while most of the hardware designs are crafted using HDL, secure verification tools that work at the HDL level are lacking⁸. For verification purposes, designers must convert their SoCs into gate-level netlists for analysis purposes⁹. Most recently, researchers have proposed methods to convert HDL code into high-level descriptions in C++ or similar languages^{10;40}. This allows for the use of software verification tools^{11;12}. Unfortunately, there are limitations to these approaches. High-level circuit structures are lost on post-synthesis netlists as well as in high-level software descriptions. The recovery of high-level circuit structures is proven to be NP-hard and an ongoing research problem.

Another factor affecting the scalability of formal verification at the SoC layer is the lack of well-defined security properties. For example, at SoC level taint propagation is tracked via the SoC bus^{16–19}. Commercially available platforms such as Cadence JasperGold Security Path Verification (SPV)¹⁴, Synopsys VC Formal Connectivity Checking, and Synopsys Formal Security Verification (FSV)¹⁵ provide means of verifying SoC infrastructure. These tools are developed with bug/fault detection in mind, and rarely consider maliciously inserted logic. For example, JasperGold SPV performs taint propagation checking through simulation. A Trojan with a counter-based trigger would significantly reduce

the efficiency of the property checking process. Exacerbating the issue is that bus behavior is often inappropriately modeled. This can result in malicious behavior from IPs in the SoC not being properly detected²⁰.

To address the above issues, in this paper we propose If-Tracker as an automatic and efficient formal verification framework. If-Tracker works at HDL level and is capable of checking security properties at SoC-scale hardware designs. Our framework leverages static information-flow tracking (IFT) to analyze the interaction between IP cores in the same SoC bus, identifying anomalies that can bypass individual security checks. If-Tracker generates a cycle-accurate data-flow graph (DFG) from an abstract syntax tree (AST) of the hardware design. This allows us to include scope and conditional information on the DFG. Once threats are detected, If-Tracker maps the DFG back to the RTL providing the user with feedback on the exact location of any vulnerabilities.

2.2 Background

2.2.1 Abstract Syntax Tree

An abstract syntax tree (AST) is a representation of the syntactic structure of the source code. Within the AST, each node represents an operation in the source code. Tools that deal with programming languages, such as compilers, assemblers, synthesis tools, and analyzers, do not directly work on the source code. Instead, these tools often take the AST as the initial representation of the code to be processed before any transformations or analysis are performed. To generate the AST of any source code, there are three steps. First, a lexer converts elements in the source code stream into tokens. Then a grammar definition is used to process the stream of tokens ensuring that the syntactical rules are met. Lastly, the token stream is rearranged and converted into a tree structure. The final AST is generated after eliminating redundant extras. In If-Tracker we leverage an open-source Verilog parser, PyVerilog⁴¹, to help parse the Verilog source files and perform basic analyses.

2.2.2 Malicious Modifications in SoC Platform

Vulnerabilities in SoCs have been reported both in academia⁴² and in industry^{43–45}. These malicious modifications can lead to dire consequences, such as the devices malfunctioning in very noticeable ways or stealthily leaking critical security information.

Dessouky et al. summarize their findings through the organization of the 2018 Design Automation Conference competition on hardware vulnerabilities (HACK@DAC 2018)⁴². They cite a large number of vulnerabilities that can be introduced in hardware which are also exploitable in software. Some of these vulnerabilities are due to errors in the implementation of the device, such as⁴⁵ where an improperly implemented GPU firmware on NVIDIA GPUs incorrectly exposes sensitive peripheral registers, leading to a privilege escalation.

Similar, Common Weaknesses Enumeration (CWE) also summarizes some SoC level vulnerabilities. For example, CWE-1276 reports an improperly integrated IP core in an SoC which results in potential security issues such as unintended privileges by the target IP or unintended exfiltration of data by the IP⁴⁴. Such behavior can be further enhanced by the IP being malicious and containing a hardware Trojan which can then proceed to further compromise the security of the platform.

2.2.3 Security Formal Verification on SoC

Authors in⁴⁰ present a model checking based formal verification framework to check the security properties of SoC bus implementation. The gate-level netlist data of the bus is parsed to a finite state machine with the assistance of gate-level reverse engineering tools. One limitation of this method is that state-explosion issue exists during the model checking process. In¹⁶, security requirements against various attacks in a multi-CPU SoC platform are modeled as taint propagation properties. Verification strategies are presented for application scenarios like key protection, user/supervisor model separation as well as access control. However, these security requirements do not consider malicious interactions among IPs. Authors in¹⁷ propose a property-driven security framework and check design examples using JasperGold. While the framework is developed to identify the properties of SoCs security evaluation with a comprehensive coverage, the developed solution is limited to the processor or IP cores rather than the whole SoC system⁶. The two case studies demonstrated in the results are either inside the processor or in IP level.

Information-flow tracking (IFT) is a powerful approach to protect confidentiality in a hardware system by detecting the sneaky path of sensitive information leakage. IFT associates data or operations with labels/taint indicating the security levels. Various secure RTL programming languages, such as Caisson⁴⁶, Sapper⁴⁷, and SecVerilog⁴⁸, have been developed to check the noninterference property. Enforcing the noninterference policy eliminates the dependency between lower sensitive outputs and

higher sensitive inputs⁴⁹. When applying these solutions, users must learn the complex tag system and manually denote labels standing for the trust level to specify the information flow policy. QIF-Verilog is later proposed to protect the confidentiality as an alternative simplified and automatic solution⁵⁰. It extends one security label from standard Verilog to reduce the cost of learning from the developers' side. However, QIF-Verilog can only protect confidentiality and validate IFT properties in IP level.

2.2.4 Formal Verification Tools for Security Property Checking

Existing commercial tools provide security property checking based on the IFT approach, such as JasperGold Security Path Verification (SPV), Synopsys VC Formal Connectivity Checking (CC), Synopsys Formal Security Verification (FSV), Questa-SecureCheck and Radix-S^{14;15;51;52}. Several of recent formal verification frameworks were proposed based on them¹⁷⁻¹⁹. However, it is far from enough to defend a substantial amount of various kinds of security threats using these tools. Take JasperGold SPV as an example; the taint propagation performed in SPV relies on simulation. It results in the issue that malicious modifications with special structures may either bypass the verification or significantly increase the time consumption for property proving. Questa-SecureCheck performs path analysis by compiling RTL to a formal model, while Radix-S imports System IP/RTL to create a security model based on predefined security rules and validates security properties in simulation – both tools have a similar underlying security checking process to SPV. We will further demonstrate these limits associated with the JasperGold SPV in the experiment section. The other two popular Synopsys Apps, CC and SPV, are also involved.

Open-source CAD solutions are increasingly developed to formally verify properties in hardware systems. Two representative examples are Yosys⁹ and Coppelia (based on Verilator)¹⁰. The problem of Yosys is that the formal model is built based on a lower level, such as the gate-level netlist. Synthesizing the design to the gate-level netlist causes the loss of higher-level structures, which means that extra computation resources are needed to recover high-level structures. It will increase the complexity of extracting security properties and worsen scalability issues in the property proving process. In the opposite direction, by applying Verilator to convert a hardware design to C++ code, Coppelia performs security checking on the equivalent C++ program⁶. The program generated by Verilator suffers from the same issue as the netlist data file – higher-level structures are eliminated during the conversion process. Furthermore, Coppelia executes the analysis in each clock cycle, which brings in significant

computation complexity. As a result, automated formal verification tools are lacking to directly process RT-Level designs.

2.3 Methodology

2.3.1 Adversarial Model

We assume that the adversary is capable of inserting malicious logic at the design phase of the IC lifecycle. For example, a rogue agent in a third-party design house may manipulate RT-Level hardware descriptions and insert hardware Trojans or backdoors⁵³. Such Trojans or vulnerabilities can bypass security checks at IP level since they do not perform malicious behaviors as individual IPs. Instead, malicious behaviors will be triggered when these IPs are integrated into an SoC platform. Signals for triggering Trojans may come from a counter, input vectors, or certain physical conditions. Once activated, the inserted malicious logic may sniff information in the bus, cause denial-of-service (DoS) to the IPs, or block the communication channels. Furthermore, a designer may insert vulnerabilities in unintended ways during the development and integration of an SoC^{40;54;55}.

2.3.2 Workflow

We show the basic workflow of If-Tracker in Figure 2.1. We parse the HDL code of an SoC design into an abstract syntax tree (AST). We navigate the AST tracking assignments, as well as the scope and conditions of those assignments. The assignments and their dependencies are then merged into a single, directed data-flow graph (DFG).

The directed graph contains the connections of all signals in the RTL code. Supported by the directed DFG, it becomes straightforward and convenient to analyze the relations among signals. Furthermore, the directed graph consists of signal nodes and temporary nodes. Signal nodes represent variables declared in the HDL code. Temporary nodes are auto-generated by our tool, which represent the semantics of statements affecting the interaction of signals, such as Boolean expressions, if-statements, serial assignments, decomposition assignments, etc. A propagation path including these nodes reflects the relationship between the target node and the source node.

Getting the directed DFG, security properties or check rules are provided by specifying the taint sources and targets in the IFT. Existing IFT based approaches only check whether a taint target

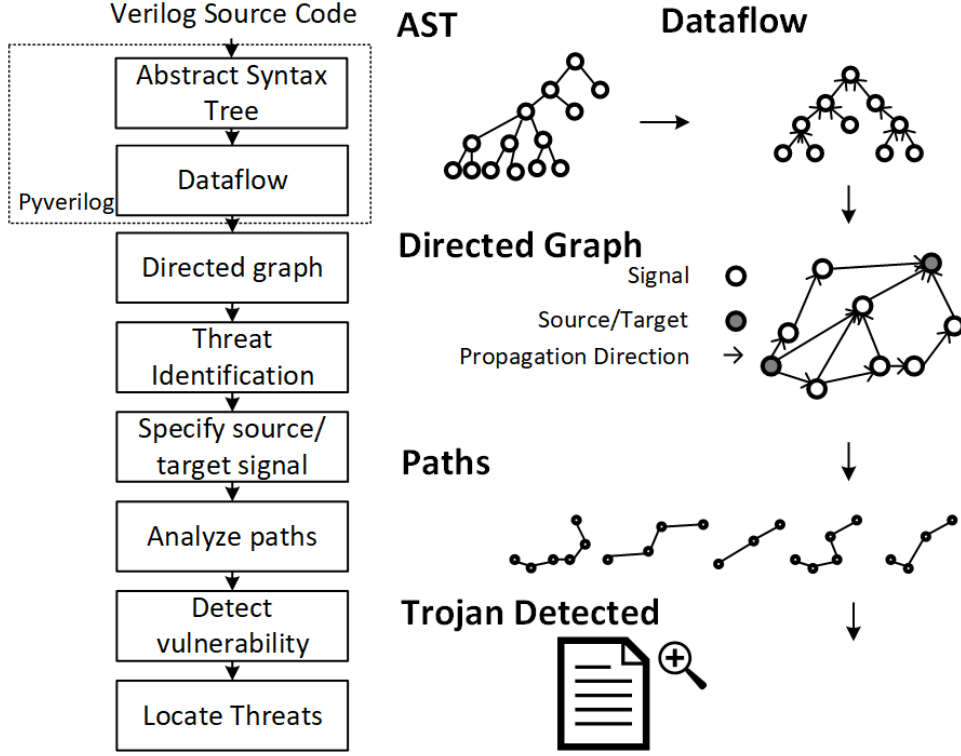


Figure 2.1: Workflow of the proposed If-Tracker framework.

can be accessed or whether the taint tag can arrive the target node. Unlike these approaches, all IFT propagation paths in the proposed framework are recorded and collected for anomaly analysis. A typical IFT propagation path includes a number of nodes. Each node on the path represents a signal with its constraints or operation required in signal propagation. Analyzing all those constrains in a path, the security rules can then be verified. More details on the verification process will be demonstrated in the experiment section. After identifying any malicious path, the proposed framework will then locate the threats in the original RTL code, as shown in Section 2.3.4. We develop the If-Tracker to automate the above procedures.

2.3.3 Information Flow Policy

Information flow policies are defined to specify the taint source, target as well as taint propagation rules. The traditional taint propagation emphasizes that the source signal with taint can/cannot be propagated to the destination under certain predefined rules. The conditions of signal transfer, such as if-statement, are not analyzed sufficiently during the propagation. Based on the AST, our framework is capable of considering the semantics of taint propagation. That is, we consider all conditions under which this taint is propagated.

Table 2.1: Security properties for addressing threats. In the table, BM: Bus Manipulation

Vulnerability	Malicious Modification	Security Property (Prop)
1. Bus eaves-dropping	When a master device writes to a specific IP, another malicious IP eavesdrops information illegally through the bus	1. When a master device is writing data to a specific IP through SoC bus, no other IPs should be in the status of accepting data from the bus
2. Sneaky path	When triggered, a sneaky path will leak information or breach the signal integrity	2. Reachability is checked from the master processor write output port to assets under protection
3. BM with signal replacement	Replace the sensitive signal by a constant value in the middle of the data transaction via AXI bus	3. Taints from all sources must be propagated to the targets
4. BM with extra logic insertion	Insert extra logic between the sensitive signal and the transmission target inter AXI bus	4. Taint propagation starts from targets and then propagates in an inverse way. Finally, the taints arrive in areas outside of the sources

With all propagation paths as well as their conditions, the behavior of IPs' interactions can be analyzed through validating the security properties. In this paper, we design and apply four types of information leakage properties. These four properties with vulnerabilities described in natural language are listed in Table 2.1 to address various threats/vulnerabilities. For these properties, we apply them in detecting the threats demonstrated in the following studies, where Property 1 is for case study 1, Properties 3 and 4 are for case study 2, and Property 2 is for case study 3.

2.3.4 Locating Threats in RTL Code

While PyVerilog is used as the basis of our parser, it does not provide the functionality in annotating data propagation path back to the RTL design. We modified the PyVerilog to support this function by keeping line information when generating the data-flow bindings. During the whole DFG generation using these bindings, we keep track of the collected line numbers. Upon the creation of temporary signals for data-flow analysis, the line number is propagated depending on how the set of signals are processed. For example, considering the example Verilog HDL code in Listing 2.1, the If-Tracker creates true node and false node temporary signals to simplify the assignment into `out_line`. In doing so, `out_line` receives a dependency on `reset` from line 18, and assignments on lines 19 and 21.

Listing 2.1 Example branching Verilog HDL code. If-Tracker takes into account line numbers when creating temporary signals as to represent data-flow in a more approachable way.

```

if(reset)
    out_line <= 1'b0;
else
    out_line <= in_line;

```

When providing the path output for a particular module hierarchy, If-Tracker can output line

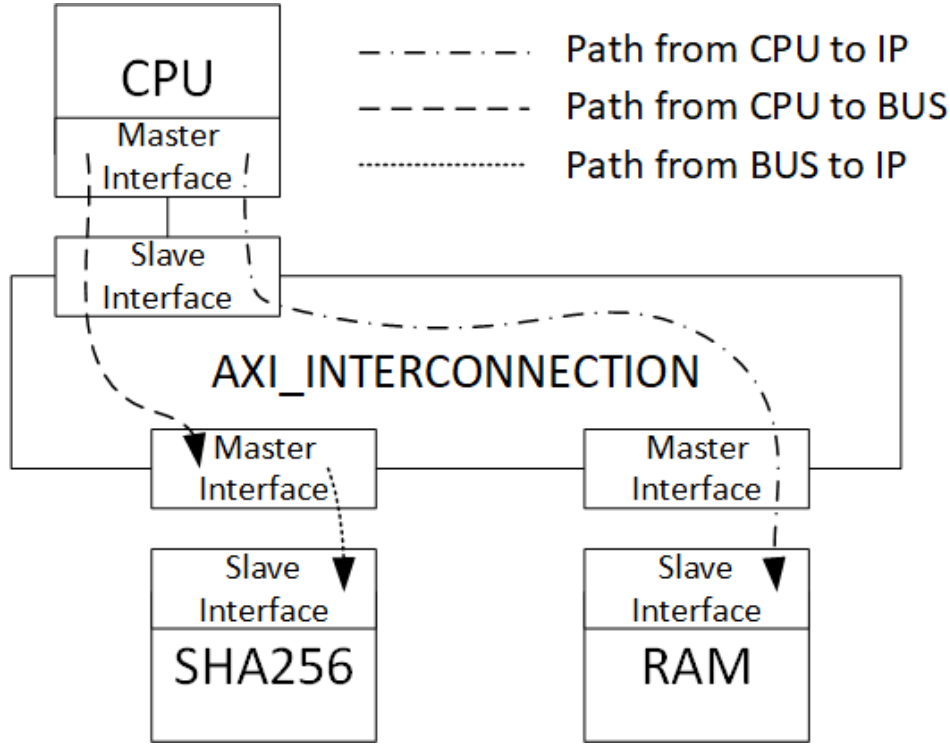


Figure 2.2: AXI4-Lite SoC benchmark

number information facilitating HDL analysis. Users are able to utilize this information to quickly identify potential errors in the design. For the purposes of quick visualization, we provide an auxiliary tool to extract the relevant portions of the code, showing snippets to the user for quick malicious code recognition in an SoC environment.

Three case studies are discussed including eavesdropping attack, TrustZone attack, and a counter-based hardware Trojan attack. For each case study, we first introduce the maliciously modified benchmark and then demonstrate how to detect the malicious behaviors using the If-Tracker framework.

2.3.5 Case Study 1: Eavesdropping Attack

AXI4-Lite SoC

Our experimental benchmark employs an AXI4-Lite based SoC with a 32 bit shared bus interconnection, which integrates an open-source pico RISC-V CPU as the master IP core and two slave IP cores, i.e., a RAM and a SHA256 crypto core. The entire system is developed using SoC benchmark in³⁹ as a primary reference. The basic architecture is shown in Figure 2.2. The CPU can read and write to both slave IP cores via the AXI4-Lite interface.

Condition₁ :	WSTRB[byte_index] == 1	=	true
Condition₂ :	slv_reg_wren	=	true
Condition₃ :	RESET == 1'b0	=	false

Figure 2.3: Propagation conditions of SHA256

Condition₁ :	WSTRB[byte_index] == 1	=	true
Condition₂ :	slv_reg_wren 1	=	true
Condition₃ :	RESET == 1'b0	=	false

Figure 2.4: Propagation conditions of RAM

Data transactions on shared bus based SoC face the risk of data being snooped by malicious third-party IPs since the shared bus is connected to all IPs. It is assumed that the RAM module contains malicious logic. A Trojan is inserted into the RAM's slave interface. As a result, the malicious RAM module constantly snoops the shared bus data and copies the data to its local.

Experiment on AXI4-Lite Benchmark

Since the hardware Trojan in RAM's slave interface eavesdrops the bus communication, it means that data needs to propagate from the CPU to both SHA256 and RAM modules at the same time. By extracting the propagation conditions and checking their conjunctions, our framework can detect this attack.

Seeing that the data leakage potentially occurs on write data channel, our experiment specifies the CPU's **WDATA** as the taint source to track the signal propagation. When a write transaction is initiated, **WDATA** will propagate taint to the SHA256 registers and RAM. The result shows four paths detected from CPU write data port to each IP module. These four paths are similar since the slave IP has to follow the AXI protocol to read data from the bus byte by byte and each path represents one byte signal propagation. Furthermore, AXI interconnect is designed with the purpose of connecting CPU and slave IPs. The detected paths just prove this functionality. Namely, the malicious behavior cannot be addressed by only checking the path existence.

In this case study, malicious modification is detected by analyzing the dependency of propagation conditions using the developed If-Tracker. Figure 2.3 lists three conditions of the taint propagation from the bus to SHA256 module. Condition₁ ensures that the byte on the bus has valid information; Condition₂ ensures that the write enable signal for the peripheral register is asserted; and Condition₃

requires that the reset signal is deactivated. If all conditions are met data can be written to the memory mapped SHA256 register. Note that signal `WSTRB` is sent from CPU. Two slave IPs receive the same signals from the CPU all the time. The `SLV_REG_WREN` signal acts as a write enable for the memory mapped register and must be asserted to enable a write operation. When routing traffic, the interconnect asserts the `SLV_REG_WREN` for the IP being accessed, selecting the desired IP. The corresponding path from bus to RAM is shown in Figure 2.4 and only the Condition₂ is different from the SHA256 path.

In our design, the propagation condition in RAM path was changed to `SLV_REG_WREN || 1'b1`, which leaves the RAM writable regardless of how the interconnect routes the write request. This results in the two paths propagation constraints being satisfied at the same time. Therefore, all propagation constraints of the two paths can be satisfied at the same time, which means that there is a data leakage from the bus.

2.3.6 Case Study 2: TrustZone Attack

Malicious Interconnection in TrustZone

Benhani et al. demonstrate three different methods of maliciously modifying the interconnection parts by tampering with the second bit of the `ARPROT` signal in⁵⁵. The first method cuts the connection of `ARPROT[1]` between the master and slave interfaces. As the result, the slave side receives a permanent `1'b0` on `ARPROT[1]`. This causes all read transactions to the core to be considered as secure. The data would be leaked from the slave side due to this non-secure visit. The other two methods utilize combinational logic on the `ARPROT[1]` by inserting an AND gate or a multiplexer. These serve as a trigger condition to arbitrarily make transactions act as if they are secure.

Experiment on AXI-Interconnect of TrustZone

The If-Tracker can detect malicious modifications to `ARPROT` in all three of the scenarios mentioned above. If no hardware Trojan is inserted, our tool finds a direct path in the 3 bit signal. However, if we tie a bit of `ARPROT` to ground to force transactions to be recognized as secure, then If-Tracker detects a second source, indicating the presence of malicious logic. If-Tracker further recognizes that a 2 bit signal is propagated from the source but a total of 3 bit reach the destination, providing yet another indication of the conflict. For the event where extra logic is added to the `ARPROT` signal through an

Table 2.2: Detection results and comparison with commercial tools. In the table, PE: path exists, CE: conditions extracted, PNE: path not exists, CC: Connectivity Checking, FSV: Formal Security Verification, SPV: Security Path Verification. All times given in seconds, RL: Reach Time Limit.

Benchmark	Source	Target	Behavior	Property	If-Tracker		CC		FSV		SPV	
					Time	Perf	Time	Perf	Time	Perf	Time	Perf
AXI4-Lite	CPU WDATA	RAM	Eavesdropping	Prop1	0.43	PE CE	1.00	PE	102	PE	26.2	PE
AXI4-Lite	Asset	Covert output	Sneaky path	Prop2	0.16	PE CE	1.00	PE	RL	N/A	RL	N/A
TrustZone	Master ARPROT	Slave ARPROT	Cut ARPROT	Prop3	0.01	PE CE	1.00	PE CE	13.0	PE	N/A	N/A
TrustZone	Master ARPROT	Slave ARPROT	And ARPROT	Prop4	0.01	PE CE	1.00	PE CE	13.0	PE	0.02	PE
TrustZone	Master ARPROT	Slave ARPROT	Mux ARPROT	Prop4	0.01	PE CE	1.00	PE CE	13.0	PE	0.02	PE

extra gate or a multiplexer, If-Tracker detects four source nodes instead of the expected single node, indicating unintended or malicious logic. By backtracking the path, If-Tracker can determine the exact source of the modifications, showing what operations are taking place and their locations in HDL source files.

2.3.7 Case Study 3: Counter-based Trojan Attack

AXI4-Lite Delayed attack

Hardware Trojans tend not to be triggered immediately, and satisfying the trigger of a Trojan may take hundreds or thousands of clock cycles. It is a concern in the dynamic analysis of circuits, where one clock cycle of circuit simulation often requires minutes or even hours. The time of demand to detect an indeterminate Trojan is unacceptable. To mimic this situation, we build a three clock cycles delay based Trojan trigger on the AXI4-Lite SoC as shown in algorithm 1. Sensitive information is leaked after a few clock cycles and a covert channel is created for the secret transmission. The framework would be tested on this model.

Algorithm 1 Three state counter trigger an attack

```

1: Every three clock cycle count++
2: if count == 10000 then
3:   out ← sensitive_data
4: end if

```

Experiment on AXI4-Lite Delayed attack

Delayed attacks set up a covert path for the transmission of sensitive data. Our method checks for the presence of a transformation channel to help determine the data leakage opportunities provided by the illegal path. The framework sets sensitive data as the source signal and an output port as the target. The framework verifies the reachability between the source and the target and the collection of

constraints on the information flowing through the node is extracted.

2.4 Comparison with Commercial Tools and Discussion

Our experiments are conducted on a dual AMD EPYC 7401 processor with 264 GiB memory. Table 2.2 summarizes how our tool performs versus commercially available offers. In the first column, we list the benchmarks mainly including AXI4-Lite SoC and TrustZone interconnection. Taint source and target are in columns 2 and 3. The malicious modification for each benchmark is shown in the fourth column. The Property column includes the utilized security properties designed in Table 2.1. The detecting results of different applications are put in the rest of columns where time consumed and performance are displayed. All threats are detected by our framework with higher efficiency.

In the eavesdropping attack from case study 1, Synopsys CC can locate the path but cannot extract the conditions. The limitation of CC is that conditions cannot be derived when the path involves flip-flops¹⁵. Synopsys FSV and Cadence SPV take longer to formally prove the reachability from the source node to target node. Since little information is involved in these paths, sophisticated malicious behavior like eavesdropping cannot be addressed by FSV and SPV. That is, only if both PE and CE are obtained from the results, the threat can be handled.

In the second row, the sneaky path attack from case study 3 is a typical taint propagation model, where Property 2 would be formally proved if the taint source reaches the target. This counter-based Trojan will only be triggered after 30000 clock cycles. As a result, both FSV and SPV reach the time/cycle limit during the proving process (we set one hour as the maximum running time). Again, due to the flip-flop issue, CC can only prove the path existence and list the path. In the developed If-Tracker, the conditions are extracted as symbols and then formally analyzed. Thus, the time consumed is not influenced by the number of counters and clock cycles.

As the the rest rows, for detecting Trust Zone attack from case study 2, both If-Tracker and CC can identify the modification. CC is also able to extract conditions since the malicious modification is the combinational logic. Even though SPV and FSV can validate the reachability or PE in most cases, for identifying the anomaly, both of them need the user to do further waveform inspection.

The results presented in this paper indicate that malicious behaviors in SoC are sophisticated so that various kinds of attacks cannot be addressed by only checking reachability or path existence. Considering the propagation conditions makes the IFT be more precise and efficient by associating multiple

propagation paths and avoiding unnecessary computation burden. Therefore, If-Tracker discovers the potential malicious tampering through extracting the propagation path topology and the conditions with a much higher performance.

2.5 Conclusion

To secure SoC designs, we present If-Tracker, an HDL level automatic IFT-based security verification framework. The tool extracts data-flow graph from the generated AST. The graph includes enough information of RT-Level hardware so that sophisticated security properties can be designed to check complex attacks including malicious interactions of IPs. Experiments performed on SoC platforms show that our framework can detect different types SoC level attacks. We compare If-Tracker with the state of the art tools and prove our tool can better address security vulnerabilities in SoC designs.

Chapter 3

Microscope: Causal Inference Crossing the Hardware and Software Boundary

3.1 Introduction

System-on-chip (SoC) designers face unprecedented security concerns due to the rise of third-party vendors in the semiconductor industry¹. As the complexity of SoC designs rises, securing a computer system requires a thorough understanding of the full software stack and the hardware architecture. SoC designers have been overwhelmed with the workload of manually diagnosing security vulnerabilities. Moreover, software-exploited hardware bugs present rigorous challenges to traditional formal methods. The emergence of transient execution attacks⁵⁶ has propelled the topic of software-hardware co-verification into the limelight. However, existing work falls short when it comes to bridging the gap between hardware and software.

For example, Coppelia¹⁰ generates software exploit for hardware bug to help engineers contextualize the threat. This work migrates hardware language to the software platform to address the co-verification issue. Specifically, the hardware Register Transfer Level (RT-level) design is first translated to C++ and then takes advantage of KLEE¹¹ for symbolic execution. Based on the security-critical assertion, the violation input sequence can be found for exploit generation. However, the software-level descriptions of hardware cannot precisely represent the behavior of the original RT-level code. This abstraction can pose a challenge to accurately analyzing the structures within the produced C++ code. Moreover, micro-architectural design can be verified as an RT-level design using open-source EDA tools such as

Yosys⁹, commercial EDA tools like Cadence JasperGold¹³, and Synopsys VC Formal¹⁵. However, these tools are not originally designed for hardware-software co-verification. While engineers can still use assertion-based verification techniques to validate potential threats, doubts remain about uncovering all interaction traces between software and hardware.

In response to these challenges, this paper presents Microscope, an innovative framework designed to infer potential software instruction patterns that expose hardware vulnerabilities. Specifically, we enhance the Structural Causal Model (SCM)²¹ with hardware features such as timing stamps, resulting in a scalable Hardware Structural Causal Model (HW-SCM). A domain-specific language (DSL) in SMT-LIB is developed to represent this HW-SCM along with predefined security properties. Subsequently, incremental SMT solving is applied to deduce all possible instructions that satisfy these properties. The effectiveness of Microscope is validated in several RISC-V SoC benchmarks.

3.2 Background

3.2.1 Structural Causal Model

A Structural Causal Model (SCM) is a mathematical framework used to describe the causal relationships between variables in a system²¹. Two set, U and V , alongside a function set defined as

$$f = \{f_x : W_x \rightarrow x | x \in V\} \quad (3.1)$$

where $W_x = U \cup V \setminus \{x\}$. V is referred to as a set of *endogenous variables* or variables within the SCM, and U is a set of *exogenous variables* or variables external to the SCM. As such, W_x is the union of all variables present in the design with the exception of x . Every endogenous variable is a descendant of a subset of exogenous variables as defined by f_x , and $\forall y \in W_x$, y is a cause of x .

3.2.2 SMT Solving.

Satisfiability Modulo Theories (SMT) solving is a powerful approach for automated verification and constraint satisfaction in software and hardware design, which supports a wide range of theories, including quantifier-free bit-vector (QF-BV), linear integer arithmetics (LIA), etc. Quantifier-Free Bit-Vector Logic (QF-BV) supports reasoning about fixed-size bit-vectors and their operations without the use of quantifiers such as Bit-wise Logical Operations, Arithmetic Operations, etc. In this paper we

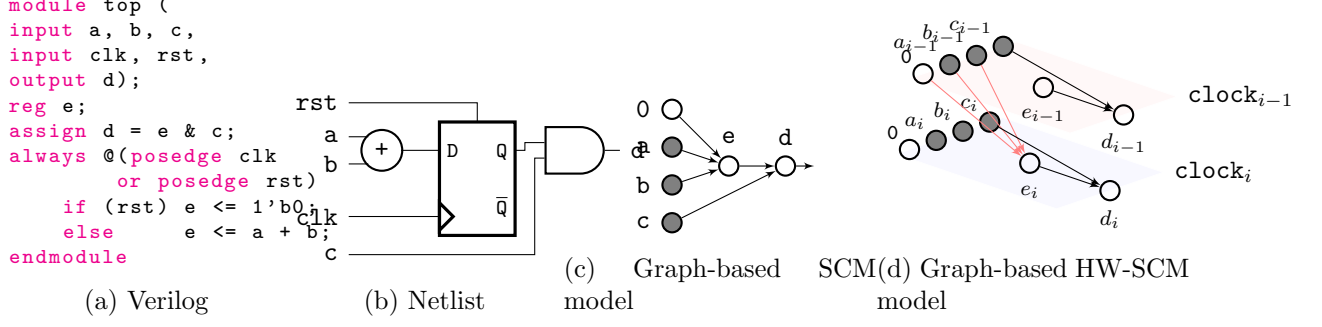


Figure 3.1: Graph based HW-SCM

use a subset of QF-BV Operations to encode the verilog into HW-SCM format.

3.3 Hardware Structural Causal Model

This section delineates the HW-SCM as a multi-layer graph model that facilitates causality inference at the hardware-software boundary. We initiate this discourse by elucidating the concept of the HW-SCM and its correlation with the traditional SCM and hardware design.

3.3.1 HW-SCM definition

HW-SCM extends the foundational concept of SCMs²¹, applying it to model software as a sequence of signals within the hardware schematic. In this context, we assume that SW_i represents the set of instructions or input signals in Clock i , while HW_i represents the set of hardware signals (excluding the inputs) in Clock i ($i \in \mathbb{N}$). To characterize the HW-SCM, we define two sets of functions:

$$f_{comb} = \{f_i : X_i \rightarrow y_i \mid y_i \in HW_i\}, \quad (3.2)$$

$$f_{seq} = \{f_i : X_{i-1} \rightarrow y_i \mid y_i \in HW_i\}, \quad (3.3)$$

Here, $X_i \subseteq (SW_i \cup HW_i) \setminus \{y_i\}$ represents a subset of signals, excluding signal y_i , from the combined set of software and hardware signals. The set f_{comb} encompasses all combinational connections within the design, while f_{seq} encompasses all sequential logic such that every cause of signal y_i , denoted as x_{i-1} , belongs to the clock cycle $i - 1$.

Following the traditional SCM model, we consider instructions and inputs to be exogenous variables

as these are external stimulus to the hardware, and signals internal to the hardware to be endogenous variables since their state is a direct response to external stimulus.

3.3.2 HW-SCM Graph Example

To illustrate the HW-SCM multi-layer graph model, we present an example as shown in Figure 3.1. The Verilog code is provided in Figure 3.1a, with its netlist representation given in Figure 3.1b. In this example, the output signal `d` depends on both signals `e` and `c`. The signal `c` is an I/O port, while signal `e` is updated by inputs `a` and `b` at the positive edge of the clock signal (`clk`). The graph representation of SCM in Figure 3.1c illustrates the signal dependencies within the design. However, since SCM is not specifically designed for hardware, it may fall short in accurately modeling certain hardware behaviors, such as timing behavior. For instance, SCM is not capable of adequately addressing delay propagation in sequential circuits.

Therefore, we develop the HW-SCM by introducing two function sets: f_{comb} and f_{seq} to fit the hardware domain-specific modeling. These function sets define different types of edges in the graph model, mapping hardware signal connections to the multi-layer structure depicted in Figure 3.1d. HW-SCM is represented as a multi-layer graph, where f_{comb} represents the connections within a layer, capturing the combinational dependencies, and f_{seq} represents the connections across layers, capturing the sequential dependencies. Hence, the hardware timing behavior can be identified through this multi-layer graph representation. Each layer represents a hardware state space within one clock cycle.

In Figure 3.1d, we present two layers: *Clock i* and *Clock i-1*. In the *Clock i* layer, the output signal d_i depends on the value of signals c_i and e_i at the same clock cycle, *Clock i*. The signal e_i in the *Clock i* layer is updated on the positive edge of the clock signal (`clk`) based on the inputs a_{i-1} and b_{i-1} at the previous clock cycle, *Clock i-1*. This means that the value of d_i at *Clock i* is determined by the inputs a_{i-1} and b_{i-1} at the previous clock cycle, *Clock i-1*, as well as the input c_i at the current clock cycle, *Clock i*. The hardware system within two consecutive time slots can be exemplified using this two-layer HW-SCM. By extending the HW-SCM into an N-layer model, we can capture the multi-clock cycle behavior of the hardware system within consecutive time slots. Moreover, the sequences of instructions from the software will be modeled as input signals with consecutive timestamps in HW-SCM. This allows for a more comprehensive representation and analysis of the hardware-software system's functionality and temporal dependencies.

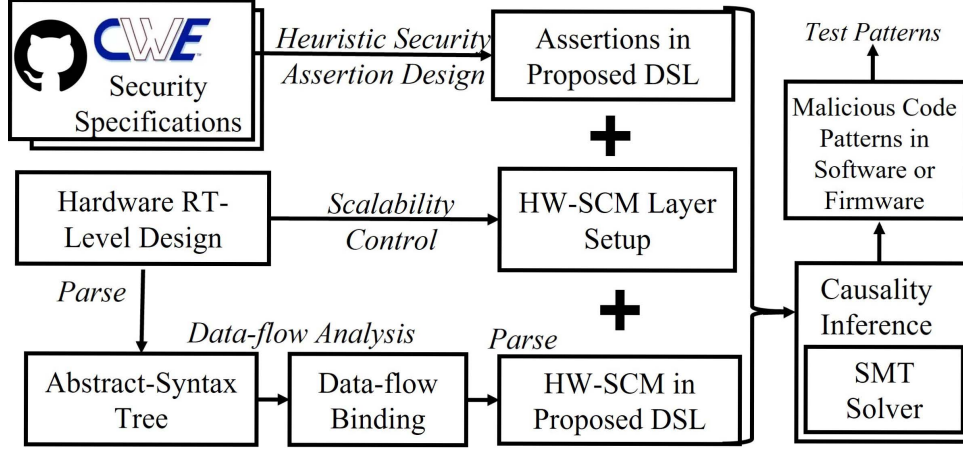


Figure 3.2: Working procedure of the proposed Microscope framework

3.4 Microscope

Microscope uses the HW-SCM as its infrastructure to enable automated hardware-software co-analysis. This section introduces Microscope and elaborates on the details of its primary steps.

3.4.1 Microscope Overview

Figure 3.2 provides a diagram of the general operational procedure of Microscope. The hardware RT-level designs are first translated into HW-SCM. In this process, information-flow tracking (IFT) is employed to traverse the abstract syntax tree (AST), which forms the basis for generating a data-flow graph. HW-SCM is then constructed based on this data-flow graph. We have designed a DSL specifically intended for representing the HW-SCM in SMT-LIB. A heuristic approach is adopted to design/obtain assertions using extant hardware databases like CWE⁵⁷ and Bugzilla¹⁰ as references. Simultaneously, it is necessary to determine the number of layers to restrict the scale of the HW-SCM. This figure hinges on the number of clock cycles the user intends to consider for the security assessment. The HW-SCM model, assertions and the number of model layers are represented utilizing the proposed DSL.

Microscope then performs causality inference using an SMT Solver. This inference involves deriving solutions that begin with the assertions at the bottom layer of the HW-SCM model. Solutions consist of same-layer inputs as defined in Equation (3.2) and higher-layer sequential inputs as defined in Equation (3.3). Consequently, the inputs from each layer are accumulated and interpreted as instructions patterns that can satisfy the assertion. This code pattern is then documented and blocked in the SMT Solver.

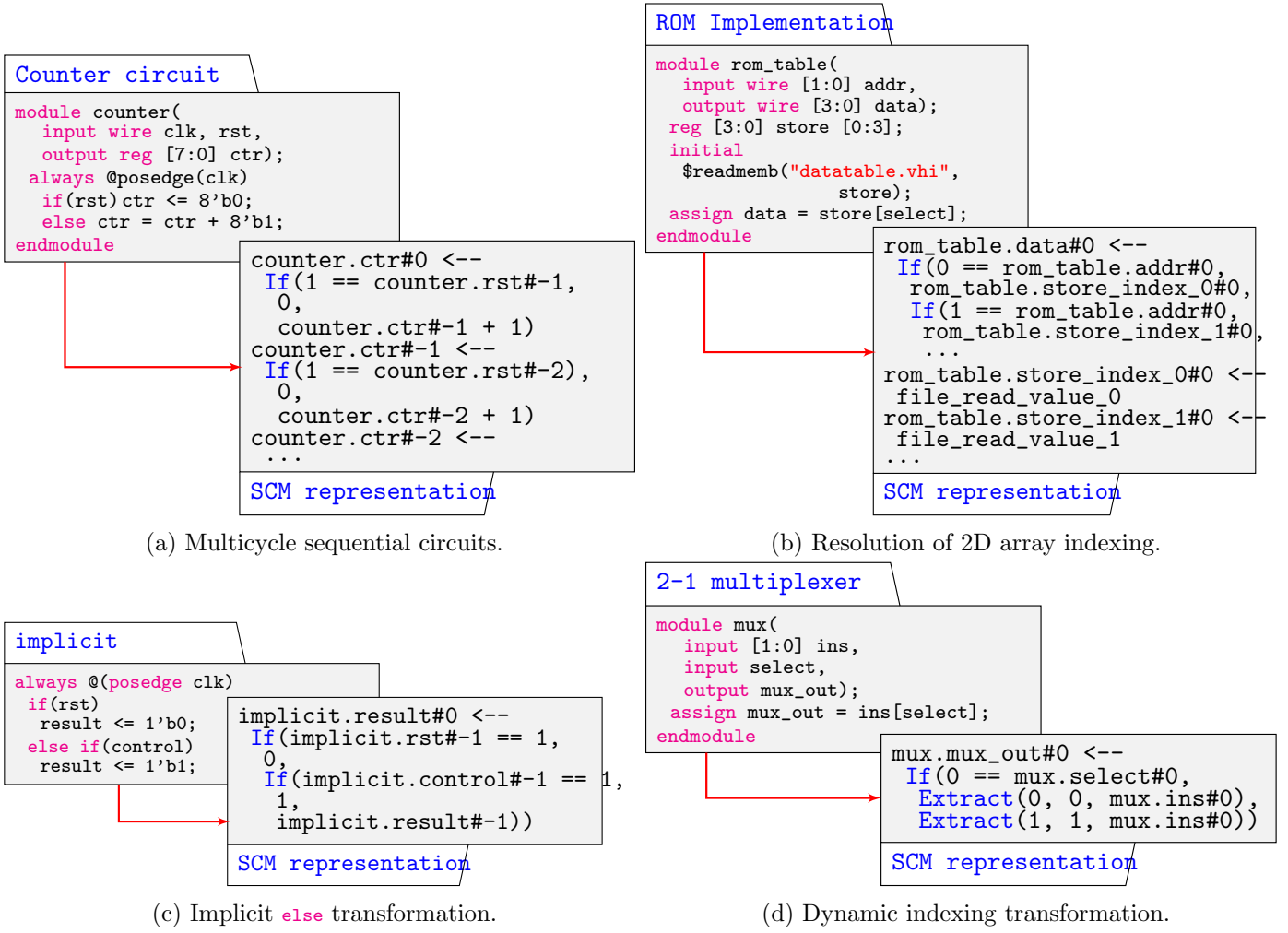


Figure 3.3: Domain Specific Language

Microscope runs this incremental solving process until all code patterns have been inferred. Any returned code patterns can be utilized to generate large-scale test patterns for the design to further explore potential vulnerabilities, as well as to fix hardware bugs.

3.4.2 Domain-Specific Language for HW-SCM

In the process of converting Verilog source code into HW-SCM SMT representation, several steps need to be followed. Firstly, all variables and parameters present in the source code are identified and treated as bit-vectors. Next, Verilog operators and bit-vector manipulations are mapped to their corresponding counterparts in QF-BV, ensuring the proper translation of operations. Lastly, control structures such as if-else statements and case statements are transformed into SMT conditional expressions, allowing for logical reasoning and analysis within the Microscope framework.

Modeling Sequential Logic

Each signal is assigned a timestamp indicating the clock cycle during which its operation occurs. In HW-SCM, the same hardware signal, when marked with different timestamp values, is treated as distinct symbols. This is demonstrated with timestamps in Figure 3.3a. `ctr` register is updated every clock cycle, depending on the state of the `rst`. The proposed DSL applies timestamps by adding the `#-n` suffix to signals, where `n` corresponds to a specific clock cycle.

Handling Dynamic Indexing Operations

Hardware description languages generally support *bit-selects* and *part-selects*. A bit-select operation occurs when an index from an array of `wire` or `reg` is selected, while a part-select operation happens when multiple consecutive indices are chosen. The proposed DSL introduces an equivalent control structure wherein an `If` condition coupled with statically indexed `Extract` constructs are used to obtain the necessary indices from the array. This construct is extended for all possible indexing values. An example is demonstrated through a simple 2-1 line multiplexer that uses a bit-select operation to set the output, is shown in Figure 3.3d.

Handling Implicit Control Structures

When encountering a `case` statement or an `if-else` statement that does not specify default behavior in a sequential block, the DSL assumes it to retain the value from the previous clock cycle. As depicted in Figure 3.3c, an implicit `else` statement is added to uphold consistency in modeling the behavior of the circuit within its HW-SCM.

Handling 2D Array Selection and Initialization Constructs

The proposed DSL flattened the array into individual elements and used a series of nested `If` statements to represent the indexing operation. Initialization constructs, such as `initial` procedural blocks and externally imported data, are parsed to basic assignment operations in the proposed DSL. The example is showcased in Figure 3.3b, where a ROM table is inferred and initialized with data from an external file. The resulting conversion breaks down the `store` array into individual elements and assigned to the output based on the value of `addr`. Moreover, individual entries are initialized by reading and parsing

the contents of the file specified by the `$readmemb()` function.

3.4.3 Assertion Development

Microscope utilizes both the HW-SCM (in DSL) and user-specified security assertions (expressed as DSL assertions) to perform causality inference. Microscope users construct assertions using the CWE database for guidance on classes of vulnerabilities that may apply to their design. This is standard practice in design verification as illustrated in [10;13;15;58](#). Before constructing the HW-SCM, users are required to specify the time window size, denoted as n , which also reduces scalability issues. Based on the specification, the Microscope will generate an HW-SCM model consisting of $n + 1$ layers for verification purposes. This implies that the given assertion will be evaluated within a time span of $n + 1$ consecutive clock cycles. Each signal is replicated $n + 1$ times with an extended numerical index. For instance, if the original signal is named `signal` the HW-SCM signals will be named `signal#0` (current clock cycle), `signal#-1` (previous clock cycle) up to `signal#-n` (n clock cycles ago) representing the signal value from historical state. Leveraging these annotations, we are able to convert SystemVerilog Assertions (SVA) or Property Specification Language (PSL) into our Domain-Specific Language (DSL) assertions, thereby effectively covering both combinational and sequential assertions.

The SMT solver returns either a *SAT* or *UNSAT* result based on the given assertions and HW-SCM. **Note:** the instruction input signal is also replicated $n + 1$ times for recording historical inputs, which are *exogenous variables* in the HW-SCM. Therefore, if the solver returns *SAT*, it means that the given DSL assertion and HW-SCM model allow for a valid stimulus to the system. In other words, there exists a value assignment to the software instructions (causality reason) that can trigger the root of hardware security threats (causality result). Following this, Microscope will employ incremental solving to identify all malicious instruction sequences that could activate the hardware vulnerability.

3.5 Experimental Results

3.5.1 Experiment Settings

A series of System-on-Chip (SoC) level evaluations with two distinct Instruction Set Architecture (ISA) SoCs - RISC-V and OpenRISC are performed in this work. Specifically, we selected the DarkRISCV^{[62](#)}, RISC-V mini^{[63](#)}, and OpenRISC 1200^{[64](#)} processors as the test bench. The vulnerabilities we used

for testing are derived from the work⁶⁵ and the commit history from OR1200. The results of these experiments are summarized in Table 3.2. Our testing environment consisted of a machine running Ubuntu 20.04, equipped with an i9-12900K processor and 32GB of memory. Z3⁶⁶ is applied as the SMT solver in this experiment.

3.5.2 Threat Model and Heuristic Assertions Development

Our framework serves as a valuable tool for verification engineers in finding causality between RT-level vulnerabilities within RT-level designs and software-level instructions. It offers a static method that validates the presence of these vulnerabilities by inferring their input patterns. These input patterns can consist of either compiled or assembled instructions for a processor in the SoC. The experiment demonstrates the Microscope by encompassing three CWE vulnerabilities and five types of design flaws extracted from the OR1200 commit history. In Table 3.1, we provide a comprehensive overview of these vulnerabilities, including the associated suspicious code patterns within the design that expose them, accompanied by concise descriptions for clarity.

Specifically, CWE-1262 represents an improper access control vulnerability related to a register interface, CWE-1234 relates to hardware internal or debug modes that allow for overriding locks, CWE-1245 pertains to the presence of improper finite state machines in hardware logic. Additionally, Bugzilla #51 and Bugzilla #76 highlight flaws in the ALU design, Bugzilla #90 demonstrates incorrect exception handling, and Bugzilla #88 and Bugzilla #97 exemplify incorrect implementations of instructions. The developed vulnerability assertions used in Microscope are listed in the last column of Table 3.2 and explained in the following paragraphs.

Bugzilla #51, #76

Two design flaws were identified in the ALU module of OR1200 when performing unsigned comparisons. The problem originates from the incorrect configuration of the `a_lt_b` flag, leading to erroneous computation outcomes. To identify the trigger pattern, the assertion specifies the erroneous behavior where the operand `a` is greater than `b` while the `a_lt_b` flag is still set. Microscope traces back the input signal `icpu_dat_i#i` (32 bit instruction) to identify the root cause.

Bugzilla #90

In the OR1200 processor, when handling a range exception, the exception program counter register (`epcr`) is reset to the jump instruction that was executed prior to the exception. The specific program counter value to be used for the reset is stored in either `dl_pc`, `id_pc`, or `ex_pc`, depending on the delay slot where the exception-causing instruction is located. We track the trigger pattern where the `epcr` is incorrectly set during an exception occurring in the second delay slot,

Bugzilla #88

When the value of `alu_op` is set to 13 (EXTW), the ALU output is incorrectly updated due to the assignment of the wrong operand. We track the trigger pattern that leads to these incorrect updates of the ALU output.

Bugzilla #97

OR1200 will not throw an exception when `l.ror` is not implemented. For a specified ISA, we track whether one missing instruction can raise an exception.

CWE-1234

We implement CWE-1234 by altering the control signal of the AES256 peripheral register. The malicious code snippet can be found in the last column of Table 3.1. The peripheral register value can be updated when the signal `io_dmem_wen` is set or debug mode is turned on. The trigger condition is either `io_dmem_wen == 1 OR debug == 1`. We build the HW-SCM based on the constraint `(io_dmem_wen#0 || debug#0)==1`. and restrict the time window to 3 in order to backtrack the input pattern that can trigger this assertion.

CWE-1262

Zero Register is considered a read-only register and cannot be written to in a processor. However, the vulnerability CWE-1262(Mail#00007(OR1200)) applies write permission to the Zero Register, altering its intended behavior. Referring to the Table 3.1, we build the constraint `reg_wb_addr#0 == 0 && reg_rf_wen#0 == 1` to backtrack the input pattern

CWE-1245

An undefined state `opcode=7'0000011` is intentionally inserted in the processor decoder as shown in Table 3.1. Microscope utilizes the constraint `req_valid#0 == 1'b1` to track the opcode pattern that enable the signal.

Benchmark	Vulnerability	Language	Cell Number	Layers	Time	Inference	DSL Property
DarkRISC-V	CWE-1262	Verilog	8,890	3	1.73 s	✓	<code>reg_wb_addr#0 == 0</code>
RISCV-mini	CWE-1262	CHISEL	17,108	3	8.16 s	✓	<code>reg_rf_wen#0 == 1</code>
DarkRISC-V	CWE-1234	Verilog	8,828	3	2.94 s	✓	<code>(io_dmem_wen#0 debug#0)==1</code>
RISCV-mini	CWE-1234	CHISEL	17,000	3	24.38 s	✓	
DarkRISC-V	CWE-1245	Verilog	8,971	3	1.45 s	✓	<code>req_valid#0== 1</code>
RISCV-mini	CWE-1245	CHISEL	16,927	3	11.36 s	✓	
OR1200	Bugzilla #51	Verilog	20,668	6	23.91 s	✓	<code>a_lt_b#0==1, comp_op#0[3]==0, a#0>b#0</code>
OR1200	Bugzilla #76	Verilog	20,714	6	24.03 s	✓	
OR1200	Bugzilla #88	Verilog	20,901	6	23.80 s	✓	<code>result#0==extended#0, alu_op#0==13</code>
OR1200	Bugzilla #90	Verilog	20,743	6	18.44 s	✓	<code>except_trig#-1[13:3] == 1,</code> <code>ex_dslot#-1 == 0,</code> <code>delayed1_ex_dslot#-1 == 1,</code> <code>epcr#0 == id_pc#0</code>
OR1200	Bugzilla #97	Verilog	20,945	6	18.42 s	✓	<code>id_insn[31:26]#-1== 46, ex_freeze#-1==0,</code> <code>id_freeze#-1==0, ex_flushpipe#-1==0,</code> <code>except_illegal#0==0</code>

Table 3.2: Experiment Result

3.5.3 Results and Analysis

CWE and Bugzilla vulnerabilities described in Table 3.1 are utilized to design security assertions in SoC-level benchmarks. We use Microscope to infer instruction patterns in three SoC platforms – RISCV-mini, DarkRISC-V, and OR1200. Microscope support various hardware languages, including Verilog and Chisel. The scalability of the benchmarks is evaluated based on the number of CMOS gates (NAND/NOR) in the column of **Cell Number**. **Layers** represents the number of clock cycles considered in the verification to resolve the assertion. **Time** of Microscope measures the total running time, including both HW-SCM building and Z3 solving.

Causality Inference in OR1200

Taking Bugzilla #51 and Bugzilla #76 as examples, one of the inference results are shown in Listing 3.1. It reveals four instructions from SoC instruction memory input `icpu_dat_i` that can trigger the vulnerability assertion. These instructions can incorrectly set the `a_lt_b` flag.

Listing 3.1 The Result Input Pattern about Bugzilla 51 from HW-SCM.

```
icpu_dat_i:
1.sfgtu  rA,rB  //11100100010...
//or
1.sfgeu  rA,rB  //11100100011...
//or
1.sflt_u rA,rB  //11100100100...
//or
1.sfleu  rA,rB  //11100100101...
```

Causality Inference in DarkRISC-V and RISC-V-mini

For CWE-1234, our findings indicate that any opcode pattern can trigger this bug, meaning there are no restrictions on the input instruction. This aligns with our expectation as the signal `debug` can override the `io_dmem_wen` signal, allowing the user to access the register interface regardless of the input instruction. To confirm our findings, we constructed another HW-SCM model that only relied on the property `io_dmem_wen#0 == 1`. Typically, the signal `io_dmem_wen` is used to monitor write access to storage devices in RISC-V cores. Our HW-SCM further revealed that only an opcode equal to 0100011 can set the signal `io_dmem_wen`, which is the STORE opcode as expected.

Approach	(Avg)Time	Replayable	Traces generated
Coppelia ¹⁰	252 s	yes	≥ 1
JasperGold ¹³	0.10 s	no	1
Microscope	21.72 s	yes	≥ 1

Table 3.3: Comparison with existing work on or1200 testbench

3.6 Related Work

A comparison between Microscope and its related works is elaborated to highlight the advantages and novelty of the proposed method in this section.

3.6.1 Comparison with Existing Works

We present a comparison of our work with the commercial tool, JasperGold FPV¹³, and the publicly available Coppelia¹⁰, as depicted in Table 3.3. “*Average Time*” is calculated based on six Bugzilla test cases. “*Replayable*” refers to whether the generated traces can be restarted from the hardware reset state, which decides if consecutive instructions can be obtained. “*Traces Generated*” denotes the number of triggered traces produced given a certain assertion. The detailed definitions of “*Replayable*” and “*Traces Generated*” can also be found at¹⁰. Since Coppelia doesn’t release detailed configuration for every test case, we use the best average time cost claims from the original paper. The JasperGold time cost is calculated by using JasperGold FPV run cover property counterexample generation time. With optimizations applied, Coppelia requires minutes to generate the exploit, whereas our method accomplishes in less than a minute. The improvement can be attributed to the fact that Microscope eliminates the need to convert Verilog designs into C++ representations, i.e., the constructed HW-SCM is directly built on the original design with precise timing information. Additionally, Coppelia cannot directly apply sequential assertions for generating software exploits. If the user wants to specify expected behavior over multiple clock cycles, they would have to manually insert extra flip-flops into the design and log the signal value from the previous clock cycles. Microscope can directly annotate signals from different layers of the HW-SCM to describe the sequential assertion since our model adds different timestamps for each signal.

In the domain of hardware-software boundary causal inference, there currently exists no dedicated commercial tool. However, some applications offer partial support for this functionality, albeit with a significant manual workload. Cadence JasperGold FPV, a cornerstone of the JasperGold Apps framework, can generate a counterexample for a user-defined assertion. Notably, JasperGold can utilize the cover property to infer input patterns and responds more swiftly than other tools. Nevertheless, a limitation is that only one counterexample is provided in each verification pass. To address all potential software-exploited bugs, a verification engineer must repeatedly execute the checking procedure until no counter traces are reported. This iterative process is both tedious and time-consuming. Furthermore, when the same baseline constraint is input into both JasperGold FPV and Microscope, there are instances where the exploits generated by JasperGold are not replayable. Specifically, the generated counterexample trace might commence from an intermediate state rather than the initial reset state.

This is particularly evident for vulnerabilities activated by state transitions, i.e., those requiring a specific continuous input sequence to trigger the payload.

3.6.2 Relationship between Information Flow Tracking and Microscope

IFT-based approaches have been well developed to detect bugs in purely hardware RTL design⁶⁷. When applied to the analysis of a hardware system, IFT aims to protect confidentiality and integrity by detecting sneaky paths of sensitive information leakage and modification. Causality inference is the process where causes are inferred from data. In the security area, it determines whether a security event is causally dependent on a preceding trigger event. The relationship between IFT and causality inference in terms of analyzing cyber attacks is first presented in⁶⁸ and elaborated in⁶⁹.

While IFT helps identify where a security breach may have occurred by monitoring data’s movement, causality inference can provide the ”how” and ”why”—how the breach happened and why it occurred in that particular way. We, therefore, consider IFT as a fundamental infrastructure upon which we base our causality analysis. Specifically, IFT helps generate data-flow graph in Microscope. In this procedure, IFT identifies which signals are related to each other. This can potentially allow the SMT solver to more effectively partition the problem or apply heuristics, which could improve efficiency. However, it’s important to note that even without using IFT as the infrastructure, the HW-SCM can still be obtained by directly parsing from RT-level (i.e., Verilog) codes. The trade-off, however, is that the time cost of SMT solving will be increased.

3.7 Conclusion and Future Work

This paper introduces the HW-SCM to apply the causality inference to RT-level hardware and software security co-verification. The proposed Microscope framework is developed to heuristically identify bug structures, and then automatically infer the potential malicious input instruction sequences that can trigger these bugs. Microscope is thoroughly validated using SoC-level platforms. In the future, we plan to collect open-source SoC platforms and insert CWE bugs so that more experiments can be carried out.

Threat	Vulnerability Description	Verilog code Example
CWE-1262 ⁵⁹ , Mail#00007(OR1200) ¹⁰	Improper Access Control for Register Interface. The value of the register is not protected, and security-related hardware data could be tampered with through the register interface. Specifically, the write operation targeting the register interface is not adequately validated.	<pre> if (write_en) regfiles[sensitive_index] <= write_data; </pre>
CWE-1234 ⁶⁰	Hardware Internal or Debug Modes Allow Override of Locks. A trusted lock bit, when enabled, can block the write operation to a set of registers or address regions. However, debug mode can bypass the lock mode and modify the locked device configuration, i.e., an external debug signal can override the original lock signal.	<pre> input debug; /* ... */ if (en debug) peripheral_register <= data; </pre>
CWE-1245 ⁶¹	Improper Finite State Machines (FSMs) in Hardware Logic. A specific input signal or signal sequence may cause the finite state machine (FSM) in the hardware logic to enter an undefined state, potentially resulting in a denial of service attack or privilege escalation.	<pre> case (opcode) 7'b0000011: /* incomplete load */ 7'b1101111: /* ... */ 7'b0010111: /* ... */ default: /* ... */ endcase </pre>
Bugzilla #51, Bugzilla #76 ¹⁰	Comparison wrong for unsigned inequality with different MSB. ALU module yield incorrect result for unsigned comparison .	<pre> assign a_lt_b = comp_op[3] ? ((a[width-1] & !b[width-1]) (!a[width-1] & !b[width-1] & result_sum[width-1]) (a[width-1] & b[width-1] & result_sum[width-1])):result_sum[width-1]; </pre>
Bugzilla #90 ¹⁰	EPCR on range exception is incorrect. Exception program counter register doesn't reset to the address of the jump instruction before the instruction that caused the exception.	<pre> else if (except_trig[13:3] == 11'b1) begin except_type <= 'OR1200_EXCEPT_RANGE; epcr <= ex_dslot ? wb_pc : delayed1_ex_dslot ? id_pc : delayed2_ex_dslot ? id_pc : id_pc; dsx <= ex_dslot; end </pre>
Bugzilla #88 ¹⁰	l.exlw instructions behave incorrectly No need to explicitly apply an extend operation when using the l.exlw instruction.	<pre> case (alu_op) 5'b0_1101: result = extended; </pre>
Bugzilla #97 ¹⁰	Ignore an exception that it should handle. When encountering an unsupported instruction, the control unit should recognize this condition and handle it appropriately, typically by generating an exception or interrupt.	<pre> case (id_insn[31:26]) 6'b101110: except_illegal <= 1'b0; </pre>

Table 3.1: Vulnerabilities description and code example

Chapter 4

WING-RTL: Attestation through Automatic FSM Watermark Generation and Integration on RTL

4.1 Introduction

Integrated circuit counterfeiting is a problem that plagues the electronics industry, costing companies and IP holders billions of dollars. Exacerbating the problem is that the component's end users are often unaware that they have unknowingly purchased counterfeits. For example, FTDI has faced a counterfeit problem where their USB VID and PID and branding were utilized in functional clones, undercutting their sales. However, the way FTDI generated a large backlash and affected only end users, not the counterfeiters²².

Device attestation is a solution that exhibits promise in identifying counterfeit integrated circuits and circuit boards. Under attestation, an on-device prover gathers information on some aspect of a device, and a trusted verifier performs a check on that aspect to make a decision on the authenticity of what is being examined. In the case of integrated circuits, a hardware attestation approach is applicable to determining whether a device is a counterfeit or not. The aspect to be examined on the device must be hard to replicate for an adversary. These can range from watermarks which add extra hardware features to the design²³, to fingerprinting using thermal or electromagnetic (EM) side

channels²⁴. Other solutions specialize in partial reconfiguration of FPGA fabric²⁵, with a portion of the FPGA containing the prover’s engine, which checks the rest of the hardware being configured.

However, previous approaches suffer from scalability or deployability issues. For example, it becomes difficult to measure temperature and EM emissions from integrated circuits outside a controlled environment with specialized equipment. Further complicating the issue is that consumer-level devices must conform to certain levels of EM emissions to be compliant with existing regulations. Approaches that target reconfigurable logic on FPGAs can not be used in discrete ASIC devices.

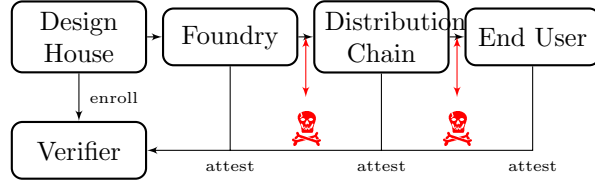


Figure 4.1: WING-RTL framework: designed ICs are watermarked and enrolled into a verification server. At any point during the supply chain, anybody can follow the attestation protocol to verify the authenticity of the IC. Adversaries may attempt to inject functional counterfeits into the supply chain or extract ICs from it for reverse engineering.

In this paper, we present WING-RTL, a cost-efficient and scalable lightweight attestation framework for IC identification. Designed to be accessible to end users, WING-RTL enables attestation at any point in the supply chain, as illustrated in Figure 4.1. The framework embeds a watermark into a finite state machine at design time and employs an on-chip watermark controller to securely service attestation requests. We demonstrate that the watermark can be used within a secure, low-cost attestation protocol that is resilient to attack and enables reliable counterfeit detection. WING-RTL incurs negligible overhead, particularly for system-on-chip designs where existing infrastructure can be reused.

4.2 Background

4.2.1 Watermarking Schemes

The finite state machine (FSM) based watermarking schemes modify/insert FSM transitions to embed watermarks. To extract the inserted watermarks, special excitation sequences are applied, and the extracted watermarks can be used for design verification. We refer to FSMs using a mathematical model. We use the common definition of a state machine as the 6-tuple $(\Sigma, \Lambda, \mathcal{S}, s_0, T, G)$; where Σ is the input alphabet, Λ is the output alphabet, $\mathcal{S}$ is a finite set of states, with $s_0 \in \mathcal{S}$ being the initial

state, T is defined as the transition function so that $T : \Sigma \times \mathbb{S} \rightarrow \mathbb{S}$, and G is the output function which is defined as $G : \mathbb{S} \rightarrow \mathbb{A}$ for Moore machines, and $G : \Sigma \times \mathbb{S} \rightarrow \mathbb{A}$ for Mealy machines.

4.2.2 Attestation

In security, attestation refers to the act of being able to formally verify a property of a device. The property to be verified may be its functionality, or that it completed an action successfully. Attestation uses a two-party system, where a trusted verifier $\mathcal{V}$ requests a measure from an on-device prover $\mathcal{P}$ in order to validate the property being measured. $\mathcal{P}$ securely collects a series of measures from the device in order to generate the measure. Any response generated by $\mathcal{P}$ must be non-replayable and unforgeable lest an adversary bypass the system and fool $\mathcal{V}$ into an incorrect diagnosis. For the purposes of this work, the property we wish to verify is whether an integrated circuit is a counterfeit or not.

4.3 Wing-RTL Design

We add a watermark to an existing finite state machine as well as connect it to a watermark controller for attestation purposes. These two newly introduced components serve as the backbone to provide an inexpensive, provably secure, and accessible integrated circuit attestation scheme, which can be used by IC distributors, their end users, and even consumers to detect functional counterfeits.

4.3.1 Adversarial Model

We wish to verify the authenticity of integrated circuits in a cost-effective manner that is accessible to anybody involved in the supply chain, even end users. We assume that the integrated circuit designer is a trusted entity who wishes to provide an authentication service to those who utilize the fabricated chip. The designer embeds a watermark on a finite state machine, which is connected to an on-die watermark controller. The latter can be accessed through a JTAG probe or some similar device. Verification of the IC's authenticity is performed by communicating with a designer-provided server.

Adversaries wishing to bypass the authentication mechanism must be able to fully replicate the functionality of both the watermark controller and watermark. We aim at the identification of functional counterfeits, counterfeit ICs which aim to replicate the original's functionality. To provide a full counterfeit, adversaries must obtain the finite state machine's watermark, for example, by continuously probing the IC by extracting syndromes. An adversary may mask themselves as legitimate users to

obtain information from the FSM by performing consecutive attestation requests or injecting random challenges to the device. We do not consider side channel attacks as means of reverse engineering, as there are solutions⁷⁰ which are orthogonal to our goals, nor we consider denial of service attacks against the attestation infrastructure.

4.3.2 Primitives for Operation

Before discussing the operation of the protocol, we will first enumerate a list of primitives that are necessary for the operation of the prover. Three main components are necessary for the operation of the protocol: a one-time programmable (OTP) memory, a watermarked finite state machine (FSM), and a watermark controller. The integrated circuit contains an OTP memory, which is used to store the serial number of the device. Our attestation scheme demands serial numbers to be unique. The OTP can also have further uses, such as storing on-board ROM code for a boot sequence if the target is a microcontroller.

Integral to our attestation protocol is a watermarked FSM. The watermarked states of the FSM implement an output function W , which is known only to the SoC designer. Under no circumstances, the outputs generated by W are to leave the SoC, disallowing the possibility of reverse engineering the FSM by constant probing. The outputs generated by W are forwarded to a watermark controller, which drives the FSM during the attestation process. The controller collects the output from the FSM, performing extra operations with it. The controller is accessible from an external interface, such as JTAG, and serves to provide inputs to and capture outputs from the FSM, as well as process the latter during an attestation request.

4.3.3 Protocol Operation

Offline Enrollment Phase.

Enrollment is performed by the SoC designer after fabrication is completed. For each integrated circuit, the SoC designer creates a new, unique serial number, which is recorded in the OTP memory on the SoC. The designer records the serial number in a database and ties it to an emulated model of the watermark. It is imperative that the serial number database alongside the emulated watermark models remain secure. These are later used as part of the verifier’s attestation process. Furthermore, in order to avoid false positives in the attestation protocol, the serial numbers should remain unique.

Online Verification Phase.

The online verification phase is the bulk of the attestation protocol. In our scheme, anybody in the supply chain, including an end user which has access to the necessary probe hardware, may attest the device. For verification to occur, the party performing the check must have a way to communicate with the verifier server. We summarize the operation of the attestation protocol in Figure 4.2.

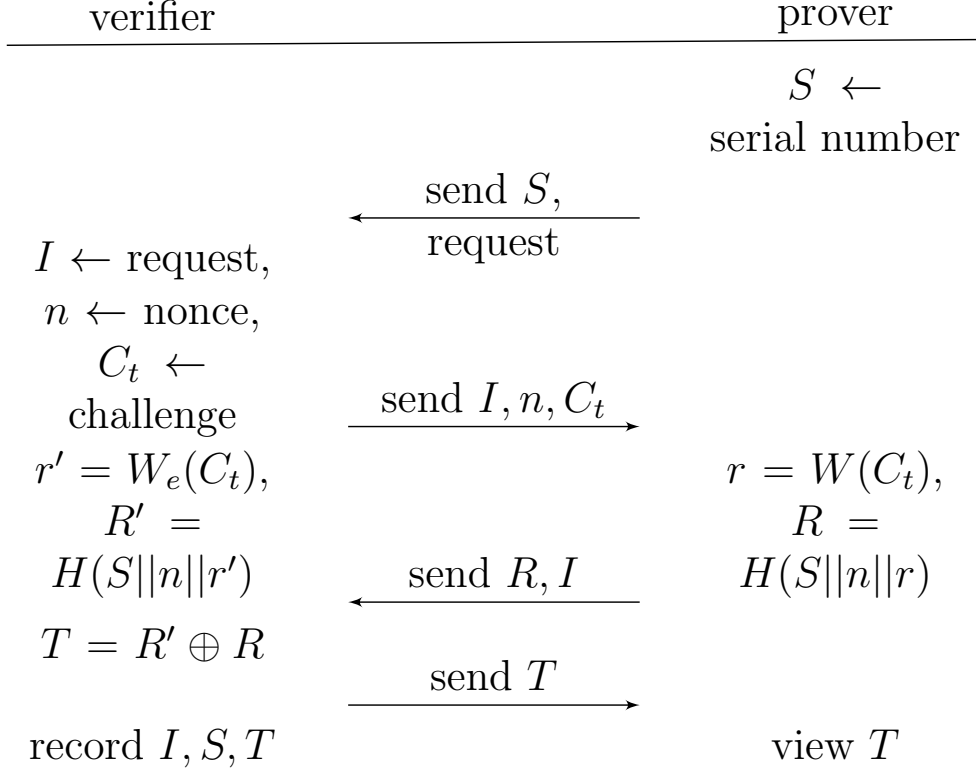


Figure 4.2: Overview of the live phase of the attestation protocol. The prover in our scheme consists of the watermarked FSM as well as the watermark controller within the integrated circuit.

The prover consists of two parts. First, an on-die watermark controller which communicates with a watermarked finite state machine (FSM) implements an output function W . The controller can feed a set of inputs to the FSM, which we call a challenge, C_t , and collect a response, r , which is the output of the FSM. The second part of the prover is a bridge that sends commands to the watermark controller and communicates with the verifier. This can be as simple as an internet-connected device with a JTAG probe.

The user's Internet-connected device on the prover initiates the verification phase. Communication with the verifier is assumed to be over an authenticated and secure channel, e.g., encapsulated

by standard TLS. To start the protocol, the user employs the probe to obtain the serial number of the integrated circuit S , then sends it to the verifier as the payload of an attestation request. The serial number is inherently unique and not considered a critical secret asset. The verifier generates a unique identification request I , a nonce n , and a bitstring which will serve as the challenge C_t . The identification request is used by the verifier so that multiple attestation requests can be performed in parallel. The verifier sends I , n , and C_t to the user. Using the probe, the user communicates with the watermark controller, sending both n and C_t . The watermark controller communicates with the watermarked finite state machine, using C_t as inputs and collecting the output of the state machine $r = W(C_t)$ as an output. The watermark controller concatenates S , n , and r and computes a cryptographically secure digest R which serves as a response to the verifier. The user collects R using the probe and sends it alongside the identification request I to the verifier. The verifier uses an emulated model of W to compute an expected response R' in the same way the watermark controller would. The verifier compares the response obtained from the user R and the expected response R' . A match indicates successfully identifying a genuine IC. The verifier logs the result of the comparison alongside a serial number, the time of the attestation request, as well as the request identification number. The result of the request is sent to the user to verify authenticity.

The verifier can employ Internet Protocol Geolocation⁷¹ to identify the area where the verification request took place. Location information is recorded by the verifier alongside the rest of the information. Storing location information allows for the reconstruction of the distribution chain, aiding the identification of possible counterfeits by examining the path an IC has taken in the global market. As new efforts to diversify IC manufacturing take hold, such as with the newly introduced *CHIPS and Science Act* in the United States and the *European Chips Act*, counterfeits introduced in the supply chain can be more readily detected. If an IC is detected to have an unusual place in the manufacturing and distribution chain, it can be identified as a potential counterfeit.

4.4 Wing-RTL Implementation

4.4.1 Addition of the Watermark

As part of the prover in our scheme we add a watermark to a finite state machine (FSM). We perform this task automatically through RTL analysis. Detection methodology follows the algorithm proposed

in⁷². The detected FSM is automatically augmented by embedding a watermark. The watermarked FSM is automatically connected to a watermark controller. The watermark scheme is shown in Figure 4.3. Watermark states, w_k , are added to the FSM implementing a Mealy machine. This scheme allows us to avoid signals being erratically driven by the FSM during attestation.

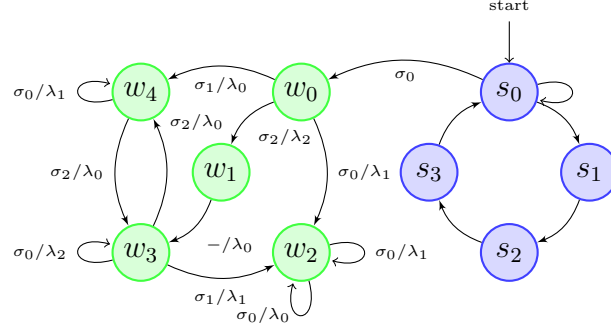


Figure 4.3: Finite State Machine watermarking scheme. As to not interfere with the regular operation of the FSM we use a separate set of watermarking states.

We add a new input to the FSM and one new output. We also add new clock and reset signals to the module, as well as a select line as inputs to the watermarked module. The select line allows us to control the clock and reset sources for the FSM. These signals are propagated by our tool to the watermark controller, which drives the newly added inputs as well as reads the output during the attestation process.

4.4.2 Enrollment

During the enrollment phase, each integrated circuit in the series is given a unique serial number. This serial number is stored internally in the OTP and in a designer-controlled database. A simulation model of the watermark scheme is stored alongside the serial numbers. The database also stores the date and time in which the enrollment of the watermarked integrated circuit was added.

4.4.3 Testing Platform

Testing is initiated by a party that wishes to verify the authenticity of the integrated circuit. For testing purposes, we utilize the example SoC in Figure 4.4. We used our algorithm to insert a watermark in the peripheral module. This connects to the watermark controller, which has bus access as to use the SHA-256 core to perform the necessary checksum computations. The watermark controller is connected to the JTAG scan chain, allowing for interfacing with a standard JTAG probe.

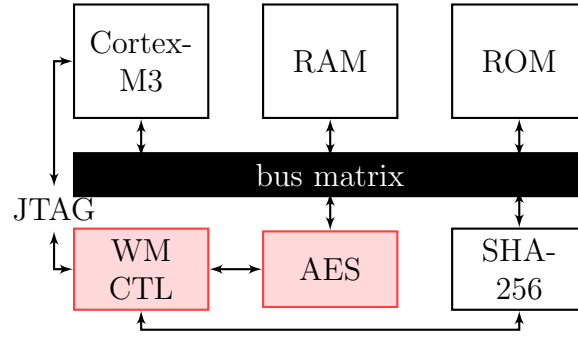


Figure 4.4: Simplified view of the SoC used as test platform. We add the watermark to the finite state machine of the AES module.

The watermark controller exposes five internal registers to the scan chain. One of these registers serves as the `status` register. The other four serve as means to send data and commands to the watermark controller, and to read from it. We call these registers `challenge`, `nonce`, `control`, and `response` registers, respectively. We use the `challenge` and `nonce` registers to clock in both the watermark challenge and nonce sent by the verifier. When writing the challenge, the first two bytes indicate the number of inputs that will be given to the finite state machine on the challenge request. We use the `control` register to initiate the operation of sending inputs to the watermarked FSM. Outputs from the FSM are stored internally in the watermark controller. Once a block of outputs is collected, the watermark controller sends it to the SHA-256 core and initiates the hash operation. The `status` register reflects the in-progress operation. Any temporary hash results are kept internal to the watermark controller. When all hash operations complete, the watermark controller saves the result into the `response` register. This can be read by the JTAG probe and sent to the verifier.

We implemented a mock-up of the verifier in Python. It is capable of recording attestation requests, generating random challenges and nonces, and verifying The responses from the prover. We use a separate Python script to initiate the attestation request and to interface with the design in a cycle accurate HDL simulator using Cocotb⁷³. This Python script sends the simulated model the challenge and nonce generated by the mock verifier, as well as hold the attestation identification request to send as a response.

The simulated probe uses the JTAG pins on the design to set up the prover operation. When the prover operation is started, the probe monitors the `control` register for the done flag. This represents that the watermark controller has finished forwarding the challenge to the finite state machine, collected its response, hashed the result in the SoC’s SHA-256 engine, and has obtained the digest. By sending

the finish request command, the digest is saved into the **response** register. Our watermark controller enforces that the done flag is set before saving a value into the **response** register: the finish request command is ignored if the flag is not set, and the attestation request is aborted.

Ensuring Secure Erasure of Watermark Response

The watermark controller in our design uses the SHA-256 accelerator present in the SoC. This is done to reduce the total hardware overhead and to avoid unnecessary redundant hardware. However, reusing the SHA-256 engine comes at the cost of the watermark controller having to send data to it. An adversary who is probing the integrated circuit in order to reverse engineer the watermarked FSM sends a challenge to the watermark controller. The watermark controller starts the process of hashing the output of the FSM with the memory mapped SHA-256 engine. Data written by the watermark controller to the memory mapped registers may be exposed to the SoC bus and become readable by software. We must ensure that the data sent to the hash engine can not be read by software. There are two events when this could occur.

At the end of a hash procedure, portions or all of the data from the output function of the watermarked FSM remain in the memory mapped registers of the SHA-256 core. Adversary controlled software on the device proceeds to read the registers, leaking information. A naive solution to this problem is to have the watermark controller clear the registers of the SHA-256 core after attestation has completed. This, however, leaves another avenue for attack. The adversary interrupts the attestation procedure, for example by soft resetting the integrated circuit. Any data written as part of the hash procedure to the memory mapped registers on SHA-256 can not be cleared by the watermark controller at the end of the attestation process. Adversary controlled software can leak the contents of the memory mapped registers.

Our design of the SHA-256 IP core contains a dual set of input registers which are multiplexed by the watermark controller. One set of these registers are exposed to the SoC interconnect and reachable by software. The other set of registers are directly connected to the watermark controller. While attestation is in progress, the watermark controller exposes its own set of registers to the SHA-256 core. Because these registers are not memory mapped their content is inaccessible to the rest of the SoC. We also route partial digests over a private channel to the watermark controller so that no intermediate computational steps are exposed to the SoC bus.

4.5 Experimental Results

Benchmark	Baseline		Watermarked		Overhead LUT/FF
	LUT	FF	LUT	FF	
aes128	1128	672	1131	678	3/6
apb2spi	29	76	57	81	28/5
crcahb	398	200	453	206	55/6
ima_adpc_enc	221	78	210	83	-11/5
pid_ctrl	755	391	808	395	53/4
prep3_binary	17	16	91	30	74/14
prep3_onehot	23	15	117	26	94/11
prep4_binary	31	4	58	9	27/5
prep4_onehot	56	16	158	26	102/10
spi	82	67	118	73	36/6
uart	85	64	131	70	46/6
xtea	395	180	423	184	28/4
average	-	-	-	-	44.6/6.8

Table 4.1: Watermark overhead in select state machines when targeting an Artix 7 XCA100TCGS324 FPGA.

Work	Watermarking Scheme			Verification Scheme		Resiliency	
	Type	Stage	Target	Authentication	Method	NI-RE	I-RE
Cui et al. ⁷⁴	FSM	HLS	Circuit	Scan Chain	Golden Signature	✗	✗
Shayan et al. ⁷⁵	Trojan	Post-Synthesis	IP	Scan Chain	Golden Signature	✗	✗
Das et al. ⁷⁶	Side Channel	Pre-Synthesis	Processor	Simulation	Golden Signature	✓	✗
SLPSK et al. ⁷⁷	Sensitive Nets	Post-Synthesis	IP/Circuit	Primary Input	Ch-Rp Pair	✓	✗
Bagul et al. ⁷⁸	Obfuscation	Pre-Synthesis	IP	Authorship	Golden Signature	✓	✗
Vliegen et al. ²⁵	Configuration	Post-Synthesis	FPGA	Attestation	Emulation	N/A	N/A
Kong et al. ⁷⁹	PUF	Post-Synthesis	IP	Attestation	Emulation	✓	N/A
WING-RTL (this work)	FSM	Pre-Synthesis	SoC	Attestation	Protocol	✓	✓

Table 4.2: Table comparing different state-of-the-art IP protection schemes based on watermarking scheme, verification scheme, and resiliency analysis against IC counterfeit. WING-RTL watermarks IC at the pre-synthesis stage and leverages attestation to verify the provenance of the IC against Non-invasive Reverse engineering (NI-RE) attacks and Invasive Reverse Engineering (I-RE) attacks.

4.5.1 Overhead of Watermarking Scheme

We test the watermarking scheme for area consumption in different Finite State Machines. The FSMs were obtained from OpenCores and were chosen as a representative set of components in System-on-Chips. Benchmarks include cryptographic cores, memory-mapped serial interfaces, and closed loop controllers. For testing purposes, we set the watermark function to have a 4 bit input and a 4 bit output. The watermark function encodes a total of 10 watermarking states for a total of $|\Sigma \times \mathbb{S}| = 160$ entries in the transition function, and $|\mathbb{A}| = 16$ symbols in the output alphabet. We synthesized the watermarked FSMs with Xilinx Vivado 2020.2 targeting a Xilinx Artix 7 FPGA ⁸⁰.

The overheads introduced in the state machines are described in Table 4.1. Our watermarking technique introduces an average of 6.8 flip-flop increments; LUT usage varies across designs due to

optimization. For instance, the `ima_adpc_enc` design exhibits an overall reduced consumption of resources, which is due to the resource sharing optimization in Vivado, which allows multiple operations to share the same arithmetic resource. With this optimization disabled, we exhibit an 8.9% increase in LUT usage. The watermark controller takes 132 LUTs and 314 flip-flops on the FPGA. We then integrated our watermarked IP cores and the watermark controller into an SoC based on the ARM Cortex-M3 microcontroller available as part of the DesignStart FPGA program⁸¹. We clock the design at the standard 50 MHz frequency, exhibiting no negative slack. When integrating the watermarked AES-128 core and watermark controller we see a total overhead of only 1.5%.

4.5.2 Scheme Resiliency

We performed a resiliency evaluation of the watermarking scheme and attestation protocol with respect to an attacker with access to a device, with the goal of building a counterfeit of the device by reverse engineering the integrated circuit while providing the means of bypassing or faking the results of the attestation protocol. A naive way is for the adversary to collect multiple nonce-challenge pairs and their response from a device, and store them in the counterfeit. The counterfeit device presents the stored responses to attestation requests. This quickly becomes unassailable, as the number of nonce-challenge pairs is effectively infinite.

Non-invasive Reverse Engineering.

We start with the adversary performing piecewise derivation of transition and output Maps attacks on the device. The adversary creates their own nonce and challenge pair (n, C_t) and sends it to the device. We note that the protocol's prover returns $R = H(S||n||r)$, where $H()$ is a cryptographically secure hash function, S is the device's serial number, n is a nonce entered in the user or adversary, and r is the output of the watermarking function $r = W(C_t)$. We wish to reverse engineer the watermarking function by entering challenges to the device and collecting the digest from the hash function. A portion of the input to $H()$ is already known: S and n . This reduces the search space for the plain-text to r .

The adversary sends a C_t to the device so that the watermark advances only one state. The adversary then ends the request. This causes r to have the smallest size possible. Recall that the output function of the watermark is defined as $G : \mathbb{Z} \times \mathbb{W} \rightarrow \mathbb{A}$, that of a Mealy machine. By sending only one input symbol, the watermark sends $(\sigma_i, w_0) \rightarrow \lambda_k$, so that $r = \lambda_k$. The watermark moves to

state $(\sigma_i, w_0) \rightarrow w_j$, where w_j is a state unknown to the adversary. However, the watermark controller computes $R = H(S||n||r) = H(S||n||\lambda_k)$. For small output alphabets such as those used during testing, the hash is trivial to brute force. Multiple probes of this type allow the adversary to fully map the output function at w_0 .

The adversary is then free to probe state w_j on the watermark by sending two inputs σ_i, σ_n to the FSM. The first produces a known output and transitions the FSM into w_j . The second produces an unknown output $(\sigma_n, w_j) \rightarrow \lambda_l$. The watermark controller computes $R = H(S||n||\lambda_l||\lambda_k)$, where λ_k is the known output of (σ_i, w_0) . This becomes trivial to brute force. At this point, the adversary has fully reverse engineered w_j , where the value of w_j is unknown. By sending different inputs as σ_i different states can be explored. The adversary maps these predicted values, then repeats the process with three inputs keeping two constant and the last one changing as a way to explore the output function of the new unknown state. This methodology is followed until there is enough information to collapse equal states in their output map while following a predicted transition map. Our watermark controller requires a minimum size for the collected syndrome. We set this value to one block size of the hash function. This renders brute forcing infeasible.

Invasive Reverse Engineering.

The adversary may attempt to decap and photograph the watermarked FSM. Obfuscation techniques can serve as a first line of defense against this type of attack, however our current implementation does not utilize them as these are an orthogonal line of defense. If the adversary obtains a netlist from the decapped integrated circuit they can attempt to reverse engineer the watermarking FSM. However, these tools require prior knowledge of the design, such as which register is the state register. If the adversary still manages to obtain this information, then the unique device's serial number used in the protocol must be cloned. Since the adversary can not enroll new devices into the attestation server, counterfeit ICs must use duplicate serial numbers and can be detected through location information. If we note that the previous location where attestation took place is in an unexpected location it implies that the IC in question was found in two incongruent places. This indicates a deviation in the reconstructed supply chain, signaling the possibility of a counterfeit. The supply chain can then be queried for further information.

4.6 Related Works and Comparison

4.6.1 Watermarking Scheme

Authors in ⁷⁴ extended the transition function of FSMs using HLS to generate a unique watermark responding to specific input sequences. Shayan et al. ⁷⁵ proposed a Hardware Trojan-inspired watermarking technique that inserts trigger and payload logic in a gate-level netlist. Both these approaches connect the watermark directly into a scan chain.

Unfortunately, this directly exposes the syndrome of the watermark and state transitions to the adversary, allowing for the reconstruction of the watermarking logic.

Side channel-based watermarking ⁷⁶ leverages the inherent low-level circuit features to generate a unique simulation trace as the signature of an IP. These methods are difficult to implement in practice, as specialized equipment is necessary to capture and filter traces.

SLPSK et al. ⁷⁷ inserted watermarking logic in sensitive nets. Using a stored challenge, authors can identify IP ownership. Since the challenge and response are readily available, this method can not be used for counterfeit detection.

Authors in ⁷⁸ present a dynamic watermarking technique using hardware obfuscation based on logic encryption to prevent reverse engineering by using a two-stage authentication to produce an owner's signature to prevent infringement of intellectual property rights (IPR). Unfortunately, these approaches require the non-scalable storage of challenge-response pairs (Ch-Rp Pairs).

4.6.2 Hardware Attestation

Authors in ⁷⁹ propose a PUF-based prover, where the unclonable function is utilized to generate a response unique to each integrated circuit. This is used to link software-based attestation mechanisms to hardware. Computations performed by the prover on the device are mixed with the PUF's output which serves as a trust anchor for the prover.

Unfortunately, the manufacture and characterization of a PUF for identification purposes has considerable overhead and cost during production. Although the authors

claim low hardware resource utilization in their implementation, no baseline data is given for comparison.

Authors in ²⁵ propose SACHa to perform attestation in configurable hardware. By using partial

reconfiguration on FPGAs, the authors partition the logic of a bitstream into a small static partition which contains $\mathcal{P}$, with the rest used for the desired functionality.

This portion of the FPGA also contains a register which stores a $\mathcal{V}$ -updated nonce. Attestation is initiated by $\mathcal{V}$ by sending the desired configuration to the FPGA alongside a nonce. The static portion of the FPGA configures the dynamic partition with a received bitstream and sets the nonce. It proceeds to compute a MAC over the dynamic portion sending the result to $\mathcal{V}$. While configuration and computation of the MAC on the FPGA takes place, $\mathcal{V}$ mirrors the actions of $\mathcal{P}$ computing an expected response. Once finished, $\mathcal{P}$ sends the computed MAC to $\mathcal{V}$, comparing the result with the expected value and completing the protocol.

4.6.3 Comparison with Previous Work

We show a summary of watermarking and IC attestation schemes in Table 4.2. We compare the resiliency of the watermarking approach against invasive and non-invasive reverse engineering techniques, and well as their usability in counterfeit identification. WING-RTL utilizes a dedicated remote attestation protocol with a protected on-chip watermark as part of the prover infrastructure. The use of remote attestation disallows an adversary from tampering with the verifier infrastructure. Although WING-RTL allows for interfacing with the watermark controller through JTAG, we do not expose FSM transition and watermark information. WING-RTL ensures that any information obtained from the watermark is not available to an adversary wishing to create functional counterfeits, providing resiliency against reverse engineering attempts.

4.7 Conclusion

In this paper, we presented WING-RTL, a scalable and accessible attestation framework for counterfeit detection in integrated circuits. Designed to support end-user verification, WING-RTL requires no modifications to existing SoC infrastructure and incurs minimal area overhead. We analyze its security under a strong adversarial model and demonstrate the resilience of the attestation protocol. Future work will focus on further refining the mechanism while preserving scalability and reusing existing on-chip infrastructure.

Chapter 5

BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero-Trust Settings

5.1 Introduction

The integration of third-party intellectual property (IP) cores into system-on-chip (SoC) designs is now a standard practice in semiconductor development. By 2021, about 1.15 trillion devices had been shipped with chips containing various IP cores, highlighting the market’s scale². In this market, IP vendors such as Arm, Intel, Cadence, and Synopsys provide the IP core design together with supplementary documentation and test benches to the IP user. Their revenue primarily comes from licensing these high-performance hardware designs and charging a fee for each chip manufactured that incorporates the IP.

Formal verification enables IP users to exhaustively verify design properties before purchase. SAT-based hardware verification stands out due to recent breakthroughs in SAT solving^{3–7;10}. However, the nature of formal verification requires the exposure of the IP core to the IP user so that the latter can undertake the verification, which raises significant risks of IP theft^{26–29}. The IP vendor is at risk of IP theft and commercial loss without strict protections when they provide temporary access to their products and disclose detailed product information prior to a transaction closing. Privacy leakage is,

however, bidirectional. An untrusted IP vendor can exploit the specifications to infer the IP user’s design needs and purchasing intentions. Such knowledge not only exposes system details of value to competitors³⁰, but also enables the vendor to manipulate pricing, posing additional commercial risks.

Existing IP core protection methods focus on preventing illegal hardware copying through techniques such as obfuscation and watermarking^{31–34}. This line of work is orthogonal to ours, which aims to preserve confidentiality throughout the IP verification phase. While legitimate efforts to shield hardware IP, such as the IEEE 1735 standard³⁵, can help, they have been proven insufficient to prevent theft³⁶, leading to financial losses across the industry. Recent demands for zero-trust approaches highlight the need for a secure framework that enables IP exchange without reliance on trusted third parties³⁵.

There remains a strong need for IP core traceability, which allows users to identify the origin of an IP and avoid redundant verification. The risks of untraceable hardware gained renewed attention in 2024 during the Israel–Hezbollah conflict, where explosive-laden pagers were falsely labeled as Gold Apollo products³⁷, underscoring the security implications of missing traceability in critical systems.

In sum, an ideal IP core distribution platform should achieve the following three goals, all without a trusted party:

Verifiability. The platform should guarantee that an acquired IP core behaves as advertised. To this end, the IP user must be able to perform IP verification before entering an agreement with the vendor.

Confidentiality. The IP vendor’s proprietary designs and the IP user’s sensitive data should both remain private until a legal framework is established, enabling integration.

Traceability. The platform should offer verifiable provenance and integrity checks for IPs, ensuring that distributed IPs are authentic while remaining agnostic to post-acquisition use.

Achieving verifiability, confidentiality, and traceability simultaneously in hardware IP distribution is challenging due to inherent tensions among these goals. Verifiability often requires exposing design details to the user, compromising confidentiality. Likewise, embedding traceability markers can alter control or data paths, potentially breaking logic equivalence checking. To mitigate such tensions, a natural direction is to leverage cryptographic techniques for verification. **ppSAT**³⁸ was originally designed to protect the privacy of Boolean satisfiability queries, and its techniques, though applicable in principle, have not been applied to hardware and do not provide an end-to-end zero-trust workflow for real-world IP trading. Its limited scalability further restricts its practicality for hardware IP verification. To address the above limitations, we propose BlindMarket.

5.2 Preliminaries

5.2.1 SAT-based Hardware Verification

Many hardware verification tasks can be reduced to SAT problems. The hardware design can be modeled as a transition system, and its behavior over a bounded number of time steps can be encoded as a Boolean formula ϕ_D by unrolling the transition relation across multiple cycles.

The property to be verified in a hardware design is expressed as a Boolean formula φ . In the open-source hardware community, temporal properties are of much interest. In this paper, we focus on the property using the operator of Overlap Implication (OI), Non-Overlap Implication (NOI), and Concatenation (Concat)⁸². Together, these operators provide a composable basis for expressing temporal relationships over bounded execution traces. Specifically, OI ($A \mid \rightarrow B$) asserts that whenever A holds, B must hold at the same clock cycle, whereas NOI ($A \mid \Rightarrow B$) requires B to hold in the next clock tick if A holds in the current clock tick. The Concat operator ($A \## NB$) checks that A must be true on the current clock tick and B on the N^{th} clock tick. Here, A and B are boolean expressions defined as a sequence⁸³. One typically constructs the conjunction $\phi_D \wedge \neg\varphi$ in conjunctive normal form (CNF). If this formula is satisfiable, it reveals a counterexample that violates the intended property; otherwise, the property holds within the given bound.

5.2.2 Cryptographic Tools

Privacy-Preserving SAT Solving Secure Two-Party Computation (2PC) is a cryptographic protocol that enables two parties with private inputs to jointly compute a function $f(a, b)$ without revealing anything beyond the final output.

Privacy-preserving SAT solving (ppSAT)³⁸ leverages 2PC to allow two parties to check the satisfiability of the conjunction of their CNF formulas without revealing the formulas themselves. In ppSAT, each party encodes its CNF formula as a pair of matrices. Let $\phi = \bigwedge_{j=1}^m C_j$ denote a CNF formula with m clauses over n variables. Each clause C_j is represented by a pair of binary vectors $(P[j], N[j])$

of length n , where the encoding is defined as follows:

$$(P[j][i], N[j][i]) = \begin{cases} (1, 0) & \text{if } x_i \text{ appears in } C_j, \\ (0, 1) & \text{if } \neg x_i \text{ appears in } C_j, \\ (0, 0) & \text{otherwise.} \end{cases}$$

The complete formula ϕ is encoded as a pair of binary matrices $(P, N) \in 0, 1^{n \times m}$, where each column j corresponds to the clause representation $(P[j], N[j])$.

We consider **ppSAT** as consisting of two phases: the **sharing** (**ppSAT.share**) phase and the **solving** (**ppSAT.solving**) phase. Each party first encodes its private Boolean formula as a pair of matrices, denoted by (P_a, N_a) and (P_b, N_b) , respectively. In the sharing phase, the two parties secret-share the concatenated matrices using Yao's sharing⁸⁴. A value v is said to be Yao-shared, denoted by $\llbracket v \rrbracket$, if it is the output of a GC-based 2PC computation and has not been revealed to either party. In the subsequent solving phase, **ppSAT** performs SAT solving by updating the shared matrix representation within the 2PC framework. Each update is referred to as a giant step, which consists of three procedures: unit propagation, pure literal elimination, and variable decision branching. In particular, variable decision branching relies on an external heuristic to determine which variable assignment to try when no variable can be directly decided, and the solving process must proceed by branching.

Oblivious transfer. Oblivious transfer (OT) is a fundamental cryptographic primitive that enables a sender to transfer one of many pieces of information to a receiver, while preserving the privacy of both parties. In a 1-out-of- N OT protocol, the sender holds a vector of messages $(m_1, m_2, \dots, m_N)$, and the receiver holds an index $\alpha \in \{1, \dots, N\}$. At the end of the protocol, the receiver learns m_α and nothing about $\{m_j\}_{j \neq \alpha}$ while the sender learns nothing about the receiver's choice α . We denote this invocation as

$$(m_i, \perp) \leftarrow \text{OT}(i, \{m_1, m_2, \dots, m_N\})$$

Ledger System. A ledger is a record-keeping system that tracks the state and ownership of assets. A cryptographic ledger extends this concept with cryptographic primitives to provide verifiable storage and consensus. BlindMarket requires such a ledger for transparency but is agnostic to the underlying

consensus mechanism (centralised, decentralised, permissionless, etc.), as long as it supports reliable storage of cryptographically validated records and the simple queries defined below.

5.3 Related Work

We compare our work against prior efforts from three perspectives: verifiability, confidentiality, and traceability.

Table 5.1: Existing methods cannot provide privacy protection for formal-based IP verification without a third party.

Approach	Parties	Verification	Stage
TIPP ³¹⁻³³	-	-	Post-fabrication
IEEE 1735 ³⁵	3	Simulation/Synthesis	Pre-silicon
Garbled EDA ⁸⁵	3	Simulation	Pre-silicon (Netlist)
MP ℓ oC ⁸⁶	2	Simulation	Pre-silicon (Netlist)
Pythia ⁸⁷	2	Simulation	Pre-silicon (Netlist)
This Work	2	Formal Logic	Pre-silicon(RTL)

Verifiability & Confidentiality. As summarized in Table 5.1, existing approaches either lack verification support³¹⁻³⁴, rely on simulation-based verification^{86;87}, or require a trusted third party^{35;85}. Whereas our work enables privacy-preserving formal verification for a wide range of IP designs in a two-party setting.

Various IP protection techniques, including watermarking, active metering, and logic locking³¹⁻³⁴, as summarized in Table 5.1, aim to deter IP misuse. However, these approaches primarily protect IP after trading or fabrication and provide no guarantees of verifiability. The IEEE 1735 standard³⁵ provides an encryption and management framework involving a trusted EDA tool vendor, yet attacks^{36;88} have exposed inherent weaknesses in its deployment within current commercial EDA tools. Garbled EDA⁸⁵ enables an IP user to obtain simulation results without revealing sensitive information such as the process design key, IP core, or EDA binaries, but it relies on a trusted EDA tool provider. MP ℓ oC⁸⁶ combines secure multiparty computation (MPC) and logic locking to support privacy-preserving IP simulation, while Pythia⁸⁷ allows the IP vendor to generate a zero-knowledge proof demonstrating the correctness of the computation to the IP user. As reflected in Table 5.1, these methods are limited to simulation-based verification at the netlist level and are primarily evaluated on the ISCAS’85 benchmarks, leaving open questions regarding scalability and applicability to formal logic verification.

Traceability. Blockchain can enhance hardware security by tracking and authenticating ICs across supply chains. Islam et al.⁸⁹ used smart contracts to log IC identity, ownership, and PUF-derived data, while Chaudhary et al.⁹⁰ reduced on-chain storage by moving CRPs off-chain with on-chain references. ICToken⁹¹ extends tracking from fabrication to end users via NFT tokenization⁹² and logic locking. However, existing approaches primarily focus on post-fabrication traceability and provide limited coverage in earlier design stages. Our work fills this gap by integrating soft IP traceability into a blockchain to enable secure and transparent lifecycle management.

5.4 Threat Model

BlindMarket operates in a zero-trust setting without relying on any trusted broker or verification provider. The system consists of two phases: (1) secure verification and (2) IP distribution, and its security relies solely on cryptographic protocols and on-chain records. During the secure verification phase, we adopt the semi-honest adversary model, where both parties follow the protocol but may attempt to infer additional information from the transcript. Under this model, the IP user learns only the final verification result and public metadata, while the IP vendor learns neither the user’s selected IP index nor the property specification. A malicious party may abort the protocol or provide malformed inputs, which may affect availability or correctness. Preventing such behaviors would require malicious-secure computation with input-consistency checks, which is beyond the scope of this work. In the *IP distribution phase*, a dishonest IP user may actively deviate from the protocol to obtain license authorization without approval from the IP vendor fraudulently. Such adversaries can collude with compromised authorized users to forge a license for protected IP content.

5.5 BlindMarket System Design

5.5.1 BlindMarket Workflow

BlindMarket involves two parties: the IP user and the IP vendor. In practice, an IP vendor typically maintains a portfolio of IP designs available for licensing. We model this portfolio as a set of abstract specifications, each describing the functionality of a design without disclosing implementation details or internal structures. Based on these high-level descriptions, an IP user selects a target design to verify while keeping their selection and property of interest hidden from the vendor. One of the core

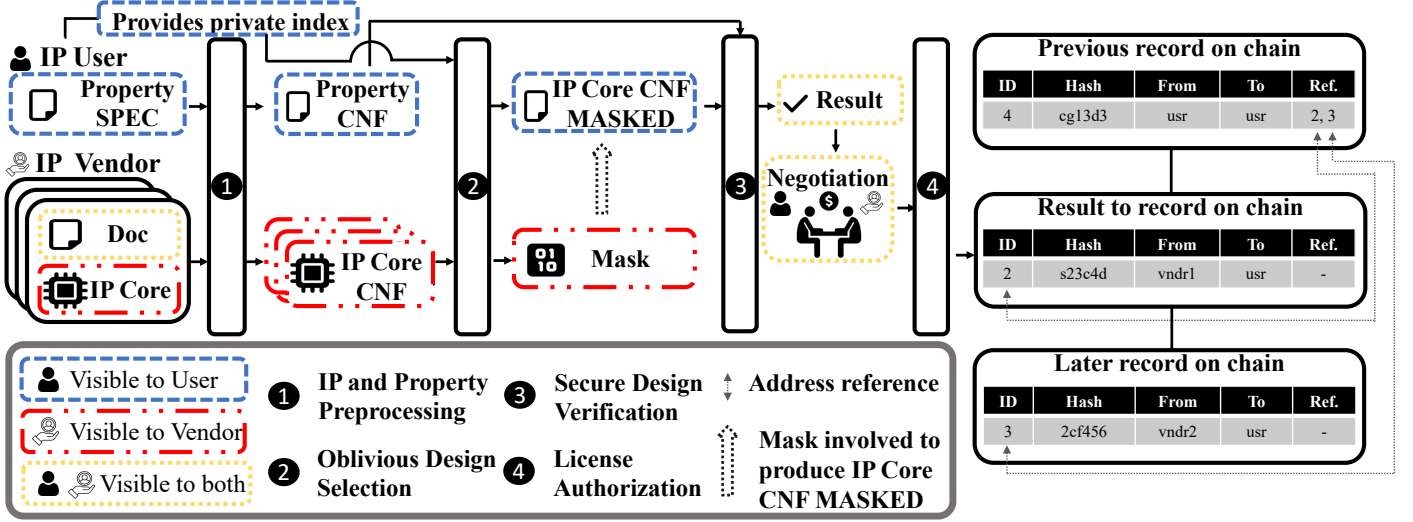


Figure 5.1: End-to-end workflow of secure IP verification and licensing in BlindMarket, organized into four phases. (1) In the preprocessing phase, the vendor encodes each IP core into CNF and masks it using randomness, while the user encodes the verification property. (2) During oblivious design selection, the user retrieves the masked CNF of the selected design via 1-out-of- N OT, without revealing the selection to the vendor. (3) In the secure design verification phase, the vendor and user engage in a 2PC protocol to reconstruct the selected design and jointly verify it against the property using **hw-ppSAT**. (4) Finally, if verification succeeds, the vendor issues a certificate bound to the authorized user and selected IP, publishing it to the ledger for traceability.

challenges is to allow the user to verify the correctness of a selected design without revealing the design itself or the user’s verification intent.

In addition to confidentiality and verifiability, BlindMarket enforces traceability after the IP core is delivered. Specifically, the vendor issues a usage certificate bound to the authorized user. This certificate enables the user to incorporate the IP into their design. BlindMarket ensures that even in the presence of collusion, an authorized user cannot transfer or replicate this certificate for the unauthorized user.

To achieve these goals, BlindMarket consists of four core phases: (1) IP and property preprocessing, where both the user and the vendor convert IP designs and property to appropriate formats; (2) oblivious design selection, where the user selects a design without learning its content or revealing their selection; (3) secure design verification, allowing the user to check correctness without accessing the full design; and (4) license authorization, where the vendor issues a certificate that binds the authorized user to the selected IP. Figure 5.1 summarizes the process.

Customized IP and property translations. For each commercial IP product, the vendor discloses a high-level functional specification and a set of *observable signals* to the IP user. The set of observable

signals typically includes the primary I/O and a subset of internal signals. To support secure verification, the vendor locally converts the design into a Boolean formula and provides *semantic map* of the observable signals, which are detailed in Section 5.5.2.

The IP vendor then reveals the *semantic map* to the IP user so that they can model the property as a Boolean formula using variables with the same semantics as the vendor does. The value of IP usually lies in the implementation of logical operations and interconnection, which is not shared in BlindMarket. To balance efficiency and confidentiality, BlindMarket provides the optimization approach for design pruning when the user is willing to reveal assertion signals. We detail the optimization in Section 5.6, and our evaluation results (see Section 5.7.1).

Oblivious design selection. The IP user selects a design of interest from the vendor’s portfolio, while keeping the selection hidden to protect purchase intentions. This phase enables private design selection for subsequent verification without revealing the chosen design to either party.

After preprocessing, the Boolean formula of all designs in the vendor portfolio are obfuscated with the same randomness. The obfuscated formula is then transmitted via an oblivious transfer (OT) protocol. The way we obfuscate the formula is information-theoretically secure, and therefore ensures confidentiality of the design. Meanwhile, the security of OT guarantees the privacy of the IP users’ choice and thus interests. The technical details of the protocol are presented in Section 5.5.3.

Secure design verification. The secure design verification protocol is initialized after the user obtains the obfuscated design, as detailed in Section 5.5.4. In this phase, the user provides the obfuscated Boolean formula of the design and the Boolean formula of their property. At the same time, the vendor inputs the randomness he uses to mask all formulas. The protocol first demasks the design formula using input from two parties in 2PC. The demasked design and the property are then checked for satisfiability using a subsequent privacy-preserving SAT-solving protocol. The output of the solver is revealed to users, indicating that the design satisfies the expected property.

License authorization. The vendor registers each design in a ledger system using a unique identifier. To support a zero-trust framework, we implement the ledger using a blockchain-based infrastructure. Upon the completion of a sale, the vendor authenticates the transaction by appending a new entry to the ledger. During integration, the IP vendor uploads an additional record that references the previously recorded purchase. The structure of the record and operations are described in detail in Section 5.5.6.

5.5.2 Customized IP and Property Translations

Existing translation tools built on hardware synthesis frameworks typically bind the design to a specific cell library and flatten it into gate-level netlists. Such flattening often removes high-level semantic structures, making it difficult to express verification properties that are naturally defined at the behavioral level. Moreover, complex operators (e.g., multiplexers) are decomposed into primitive logic gates, which introduces a large number of auxiliary variables and clauses during CNF generation. Since BlindMarket encodes the CNF as a pair of binary matrices $(P, N) \in \{0, 1\}^{n \times m}$, larger values of n and m directly increase the computational and communication overhead of the subsequent secure protocols. Therefore, we customize a word-level translation for IP designs and properties that maintains a small number of variables, and both users and vendors independently encode the design and properties into Boolean formulas based on a publicly agreed-upon bound.

Formula Generation of Design

On the IP vendor side, the design is encoded as a Boolean formula through the following steps. Our static analysis frontend (based on^{9;41}) takes the synthesizable RTL as input and emits a word-level intermediate representation, which is then translated into a quantifier-free bit-vector SMT formula (QF-BV)⁹³, where each variable is modelled as a fixed-size bit vector. For instance, in Example 1, the signal **counter**, defined starting at line 3, is translated into its corresponding SMT bit-vector representation beginning at line 10. The resulting SMT formula is then converted into CNF via bit-blasting⁹⁴ and Tseytin transformation⁹⁵. Bit-blasting translates bit-vector operations into Boolean logic, while the Tseytin transformation encodes Boolean formulas in CNF using auxiliary variables without exponential growth.

Meanwhile, BlindMarket frontend allows the vendor to automatically construct a *semantic map* for the *observable signals*, which correspond to a small subset of design variables. The semantic map assigns each observable signal a unique literal, as shown in lines 15–16, where each bit of the variables **counter** and **reset** is assigned a unique literal. This mapping enables the IP user to formulate properties in Section 5.5.2. The signals used in the control expressions defined as control signals are also tracked in the semantic map (such as **reset** in Example 1 line 2), which remains private and is weighted by its depth in its statement’s abstract syntax tree (AST)⁹⁶. We leverage this metric to accelerate the SAT

solver, as detailed in Section 5.5.5. Example 1 demonstrates how a simple counter updates `counter` based on previous value of control signal `reset` and `enable` symbolically. The SMT constraints in lines 10–13 specify an equivalent logical formula that encodes its behaviour, where we append $\#i$ to each signal to denote the timestamp. Specifically, `counter#2` denotes the symbolic value of `counter` at the second clock cycle and is updated by a nested ternary expression referencing `counter#1` from the first clock cycle. Furthermore, the vendor provides a semantic map that associates the 3-bit signal `counter#2` with the literal $\{\ell_4, \ell_3, \ell_2\}$ and the 1-bit signal `reset#1` with ℓ_1 , while also recording the depths of `reset#1` and `enable#1` as 1 and 2, respectively.

```

1  always @(posedge clk) begin
2      if (reset)
3          counter <= 3'b0; // Reset the counter to 0
4      else if (enable)
5          counter <= counter + 3'b10; // Increment the counter by 2
6      else
7          counter <= counter + 3'b1; // Increment the counter by 1
8      end
9  /* SMT Formula */
10 counter#2 <--> (ite (= |reset#1| #b1)
11 #x00(ite (= |enable#1| #b1)
12     (bvadd |counter#1| #x02)
13     (bvadd |counter#1| #x01)))
14 /* Semantic map */
15 {reset#1 : 1, counter#2[0] : 2,
16 counter#2[1] : 3, counter#2[2] : 4}
17 /*Ctrl Depth*/
18 {reset#1 : 1, enable#1 : 2}

```

EXAMPLE 1: Incrementing counter with synchronous reset

Boolean Encodings of Properties

The IP user then applies the *semantic map* to encode its property as a Boolean formula. BlindMarket focuses on properties composed of three common primitives, namely NOI, OI, and Concat, each of which can be translated into basic Boolean logic. NOI is represented by $\neg a_i \vee b_{i+1}$, indicating that if signal `a` holds at time step i , then `b` must hold at the next time step $i + 1$. In contrast, OI is encoded as $\neg a_i \vee b_i$, requiring that `b` hold in the same cycle as `a`. Concat expresses a sequential pattern and is encoded as $a_i \wedge b_{i+n}$, meaning that signal `a` must hold at time i , and signal `b` must hold exactly n cycles later.

In Example 1, the user obtains the semantic map from the IP vendor, and a property such as `reset#1` $\rightarrow$ (`counter#2` = 0), can be used to check whether `counter` is reset to zero whenever `reset` is asserted. Under the semantic map, this requirement is translated into the Boolean formula

$$\neg\ell_1 \vee (\neg\ell_2 \wedge \neg\ell_3 \wedge \neg\ell_4).$$

5.5.3 Oblivious Design Selection

Oblivious selection is performed prior to secure verification to prevent the IP vendor from knowing the IP design of the user’s interests. In standard verification workflows, the user must disclose the target they wish to verify to initiate the process. Such disclosure can be problematic, particularly when the vendor may exploit knowledge of user preferences to manipulate pricing or restrict offerings.

BlindMarket leverages oblivious transfer (OT) to address this challenge. OT enables an IP user (the receiver) to obtain one design from a set offered by the vendor (the sender) without revealing which design is selected. However, directly applying OT will compromise confidentiality by exposing the full IP design to the user. To preserve design privacy while enabling verification, the OT procedure must be integrated with the privacy-preserving framework. This integration requires the SAT solver to operate in a white-box manner, as detailed in Section 5.5.4.

Initially, the CNF of each design ϕ_k in the design portfolio Φ is encoded as a pair of binary matrices (P_k, N_k) on the vendor side. Then each pair (P_k, N_k) is masked using a shared pair of random binary matrices (R_P, R_N) . We require (R_P, R_N) to be refreshed for every verification session. Otherwise, colluding users could XOR masked instances obtained from different runs to cancel the masks and infer structural relationships between protected designs. Specifically, the masked representation is computed as $(\widetilde{P}_k, \widetilde{N}_k) = (P_k \oplus R_P, N_k \oplus R_N)$. The collection of masked matrices $(\widetilde{P}_k, \widetilde{N}_k)$ is used as the vendor’s input to the oblivious transfer (OT) protocol.

On the IP user side, the user selects an index i corresponding to the desired design based on the public metadata provided in the vendor’s design portfolio. This index serves as the receiver’s input to a 1-out-of- N OT protocol. During the OT execution, the two parties jointly run the protocol such that, at the conclusion, the user obtains the pair of masked matrices $(\widetilde{P}_i, \widetilde{N}_i)$ corresponding to the selected design. The IP vendor retains the random masking matrices (R_P, R_N) .

5.5.4 Secure Design Verification

Secure design verification aims to determine whether a proprietary design satisfies a user-specified property, without revealing the design to the user or disclosing the property to the vendor. However, applying ppSAT directly is insufficient in our setting as it requires both parties to agree on a common

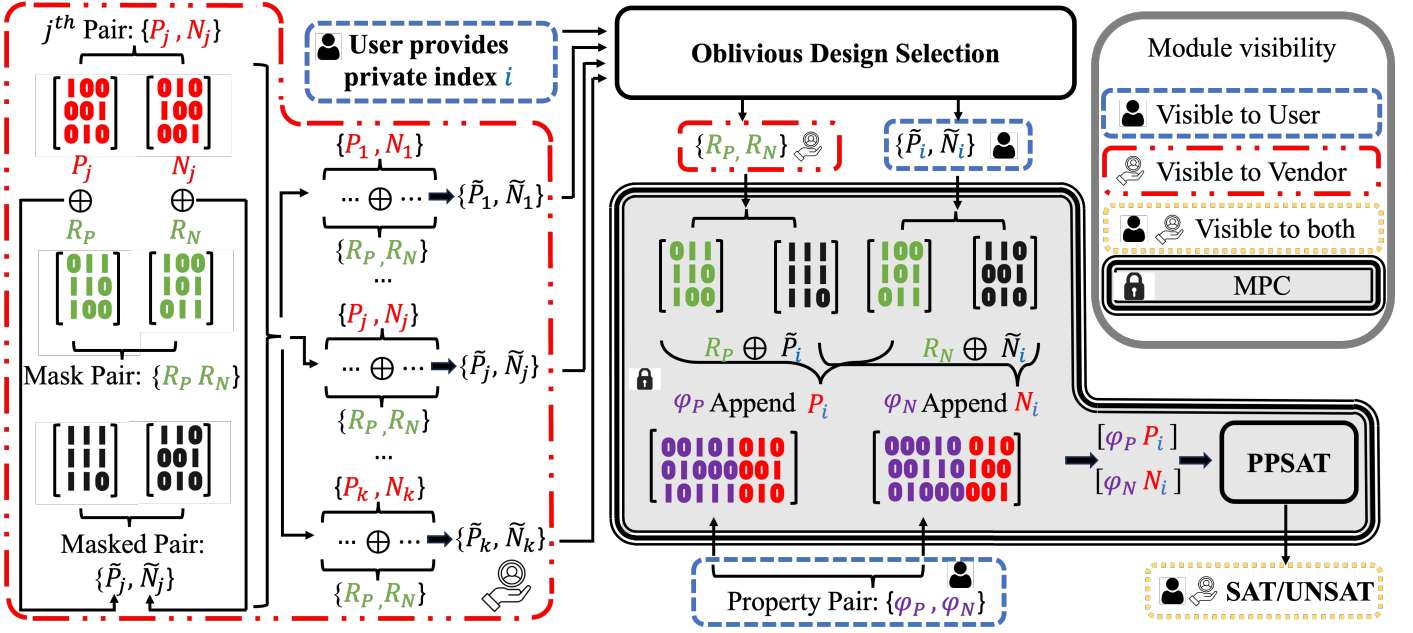


Figure 5.2: Secure hardware design verification workflow in BlindMarket. The vendor masks each CNF-encoded design using random matrices, and the user selects a target design via oblivious transfer (OT), receiving only the masked version. The vendor and user then run a 2PC protocol to reconstruct the selected design and combine it with the user’s property to form $\psi = \phi_i \wedge \neg\varphi$. The satisfiability of ψ is evaluated using `hw-ppSAT` under 2PC, revealing only the final result to both parties while preserving mutual confidentiality.

verification target in the setup. This synchronization step reveals the IP user’s selection to the vendor.

To resolve this issue, after the oblivious design selection protocol, the user obtains a pair of obfuscated matrices $(\tilde{P}_i, \tilde{N}_i)$ corresponding to the selection index i . Following this, BlindMarket initiates a secure 2PC protocol in which the user inputs the masked design matrices and the property φ , while the vendor inputs the retained random masking shares (R_P, R_N) . The design matrices are then reconstructed by demasking in 2PC:

$$(P_i, N_i) = (\tilde{P}_i \oplus R_P, \tilde{N}_i \oplus R_N). \quad (5.1)$$

This demasked pair of matrices encodes the selected design ϕ_i . The verification task is then formulated as the satisfiability of the Boolean formula $\psi = \phi_i \wedge \neg\varphi$, which is secret-shared between the two parties within the 2PC. The `ppSAT.Solve` protocol is then invoked to evaluate the satisfiability of ψ and the satisfiability result is revealed to both parties at the end. Throughout this process, the IP design remains hidden from the user, and the verification property remains unknown to the vendor, ensuring mutual confidentiality.

5.5.5 hw-ppSAT

Numerous techniques have been proposed in the formal method community to improve solver efficiency through conflict analysis^{97;98}, structural analysis^{99;100}, etc. However, lifting these techniques into a privacy-preserving setting remains an open challenge³⁸. Although `ppSAT.Solve` adapts DPLL as its base routine with various branching heuristics for acceleration, as a general-purpose SAT solver, it doesn't exploit structural information derived from hardware designs. Prior work has shown that incorporating structural information into variable ordering can substantially improve solving efficiency in SMT solving¹⁰¹. To enable `ppSAT.Solve` to benefit from hardware semantics under encryption, BlindMarket augments the `ppSAT.Solve` with a hardware control-flow guided heuristic, referred to as `hw-ppSAT`.

Control-flow Guided Heuristic

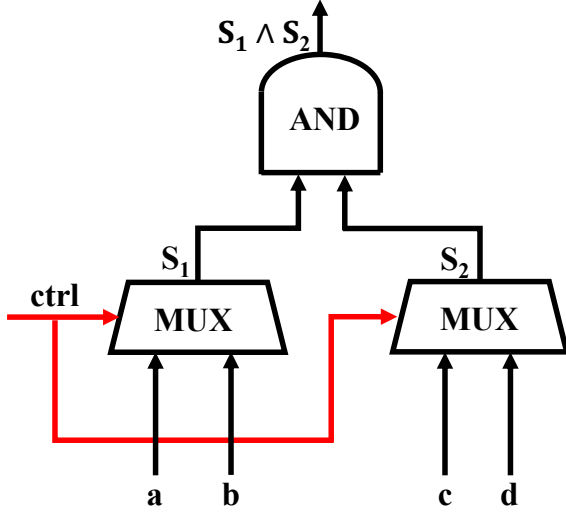
Formally, let $\mathcal{V}$ denote the set of all variables in the Boolean formula generated from a hardware IP design. Our heuristic defines a partial order relation $\prec$ over $\mathcal{V}$. For any two variables $x, y \in \mathcal{V}$, if $y \prec x$, then x should always be justified before y . That is, when both x and y are undecided, and the solver must make a branching decision, it will always assign a value to x before considering y . The order captures the three principles as follows:

Principle 1: Variables encoding control signals are prioritized for branching over all other signals. In hardware designs, a control signal may govern multiple data paths. Once it is assigned, many downstream data values become either constants or irrelevant to the evaluation, allowing the solver to prune a large portion of the search space.

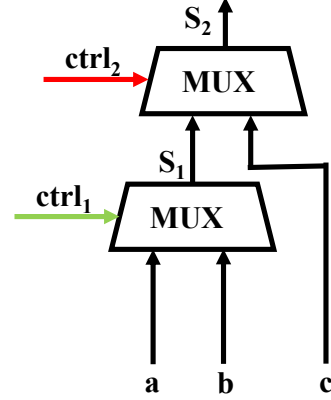
Example. Figure 5.3a depicts the gate structure for $S_1 \wedge S_2$, where the values of S_1 and S_2 depend on the signal `ctrl`. When `ctrl` is true, S_1 and S_2 propagate the values of `a` and `c`, respectively; otherwise, `b` and `d` are passed. Without prioritizing the control signal `ctrl`, verifying $S_1 \wedge S_2$ requires evaluating four possible cases: $a \wedge c$, $a \wedge d$, $b \wedge c$, and $b \wedge d$. Two of these lead to conflicts. However, when `ctrl` is fixed, the conflicting cases are eliminated, leaving only two scenarios to verify.

Formally, let $\mathcal{V}_C \subseteq \mathcal{V}$ be the set of control signals, and $\mathcal{V}_D = \mathcal{V} \setminus \mathcal{V}_C$ be the set of non-control signals. $\forall c \in \mathcal{V}_C, \forall d \in \mathcal{V}_D : d \prec c$.

Principle 2: Within a single time frame, outer control signals dominate inner control signals. In a nested



(a) Heuristic 1: `ctrl` is prioritized



(b) Heuristic 2: `ctrl2` dominates `ctrl1`

Figure 5.3: Illustration of two signal prioritization principles. In (a), the control signal `ctrl` is prioritized during branching, since assigning `ctrl` to `false` renders the values of `a` and `c` irrelevant to the satisfiability of the encoded design. In (b), for two control signals `ctrl1` and `ctrl2`, `ctrl2` is prioritized, because once `ctrl2` is assigned `false`, `ctrl1` no longer influences the evaluation.

structure, an outer control signal governs the scope that encloses the logic of inner signals. Evaluating an inner control before its corresponding outer signal leads to unnecessary computation, because if the outer control disables that block, the inner assignment becomes irrelevant and its effect is masked by the outer logic.

Example. If `ctrl2` is false in Figure 5.3b, the value of `S2` is determined by `c`, while `ctrl1` does not affect the result due to the blocked data flow from `a` and `b`. Therefore, the interpretation of `ctrl1` is dominated by `ctrl2`.

Principle 3: Across unrolled time frames, control signals at later time stages dominate those at earlier time stages. When a design is unrolled, signals at earlier time stages are structurally nested within those of later time stages, making them analogous to inner controls. Following the same prioritisation rationale as Principle 2 and classic tuning practices¹⁰², the solver should evaluate later-stage control signals before earlier ones. Formally, for signals $c_i \in \mathcal{V}_C$ at time i and $c_j \in \mathcal{V}_C$ at time j , if $j > i$, then $c_i \prec c_j$, meaning that c_j should be interpreted first.

Figure 5.4 illustrates the unrolled gate structure of the `counter` signal from Example 1. In this case, the symbolic value of `count3` is updated from `count1` through the unrolled circuit. To verify a property involving `count3`, the solver should justify `reset2` and `enable2` before `reset1` and `enable1`.

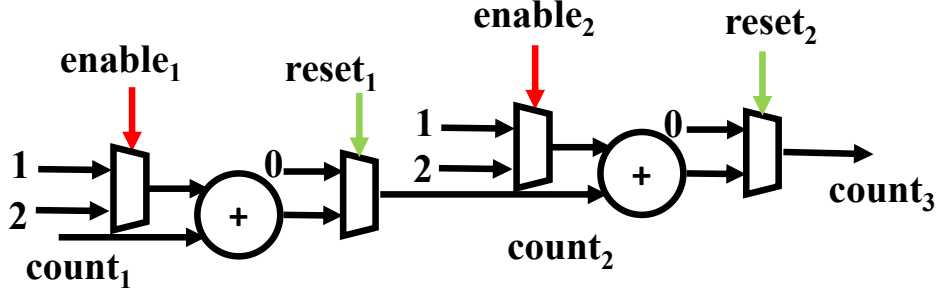


Figure 5.4: When the circuit is unrolled, later-stage control signals dominate earlier-stage ones. Thus **enable₂** dominates **reset₁**. Based on Principle 2, **reset₁** dominates **enable₁**. Consequently, the priority chain becomes **enable₂** $\succ$ **reset₁** $\succ$ **enable₁**.

In general, determining the partial order set of control signals from a CNF is non-trivial because structural information disappears once the design is encoded into Boolean clauses. BlindMarket allows the IP vendor to determine the partial-order set locally during preprocessing (see Section 5.5.2), without leaking any sensitive design information to the IP user.

Specifically, control signals are extracted through AST traversal. The prioritisation of control signals is derived from dataflow analysis using Depth First Search (DFS). For signals $c_1, c_2 \in \mathcal{V}_C$ within a single time frame, $c_1 \prec c_2$ if $\text{Dep}(c_1) > \text{Dep}(c_2)$, where $\text{Dep} : \mathcal{V}_C \rightarrow \mathbb{N}$ maps each signal to its depth obtained by the DFS from dataflow. Both extraction and prioritisation can be performed based on RTL static analysis tools such as^{9;41}.

hw-ppSAT Implementation

Unlike `ppSAT.Solve`, `hw-ppSAT` requires the IP user to provide a partial-order set $\prec_C$ as an additional input, which is generated on the IP vendor side.

To prevent information leakage, the heuristic set associated with each IP design is retrieved through the same Oblivious Design Selection protocol described in Section 5.5.3. Concretely, the IP vendor generates a heuristic set $\prec_C^k$ for each design ϕ_k during preprocessing and includes it in the message vector used in the OT protocol. The IP user then retrieves the corresponding heuristic set using a 1-out-of- N OT via

$$(\prec_C^i, \perp) \leftarrow \text{OT}(i, \{\prec_C^1, \prec_C^2, \dots, \prec_C^N\}).$$

By the security guarantee of OT, the user learns only $\prec_C^i$ and obtains no information about $\{\prec_C^j\}_{j \neq i}$, while the vendor learns nothing about the user's selection i . Furthermore, since $\prec_C$ only specifies a

partial ordering of variables for branching decisions, it does not reveal structural information about the underlying circuit.

Algorithm 2 Decision Step in hw-ppSAT

Input: $L = \{\ell_1, \dots, \ell_{2n}\}$, $C = \{C_1, \dots, C_m\}$, $\prec_{\mathbb{C}}$

Output: $d \in [1..n] \times \{0, 1\}$

```

1:  $d \leftarrow (\perp, \perp)$ ,  $d' \leftarrow (\perp, \perp)$  ▷  $d$ : decision;  $d'$ : fallback decision
2: for  $i \in \prec_{\mathbb{C}}$  do
3:    $s_P \leftarrow \perp$ ,  $s_N \leftarrow \perp$  ▷ Check literal polarity in clauses
4:   for  $j \leftarrow 1$  to  $m$  do
5:      $s_P \leftarrow s_P \vee C_j.\text{contain}(\ell_i)$ 
6:      $s_N \leftarrow s_N \vee C_j.\text{contain}(\neg \ell_i)$ 
7:   end for
8:    $d \leftarrow s_N ? (i, 0) : d$ 
9:    $d \leftarrow s_P ? (i, 1) : d$ 
10: end for
11: for  $i \in L \setminus \prec_{\mathbb{C}}$  do ▷ Fallback to standard DLIS heuristic for non-control signals
12:    $d' \leftarrow \text{DLIS}(C, i)$ 
13: end for
14:  $d \leftarrow (d \neq (\perp, \perp)) ? d : d'$  ▷ Use heuristic decision; otherwise, fallback

```

We formally define the hw-ppSAT decision step in Algorithm 2, which implements a hybrid method that combines DLIS (the best in ppSAT.Solve) with a control-flow-guided heuristic. In this approach, hw-ppSAT selects a control literal available from $\prec_{\mathbb{C}}$ for the branching heuristic. If all control literals have been evaluated and the formula remains unsolved, the solver defaults to the DLIS strategy for the remaining non-control literal.

Table 5.2: Soft IP metadata in each record.

Attribute	Description	Size
ID	Unique IP identifier	32B
Hash	Hash of source code	32B
From	Vendor's address	20B
To	User's address	20B
Reference	Integrated IDs (up to 5)	32B $\times$ 5
Total		264B

5.5.6 License Authorization

To ensure traceability and transparency throughout the lifecycle of a soft IP, its record is published to the distributed ledger. This record contains the IP's ownership information, transaction details, and

references to its integrated components. The format of the records is detailed in Table 5.2. BlindMarket offers three functions for license authorization services. Specifically, **authzSoftIP** allows an IP vendor to grant usage authorization by publishing a signed record that binds the buyer’s and vendor’s address. **integrateSoftIP** records IP integration by publishing an updated **Reference** record whose component field encodes IP provenance. However, these ledger-based mechanisms do not physically prevent redistribution outside the marketplace. Redistribution risks can be mitigated by integrating complementary protection techniques such as hardware locking³¹, active metering³², and watermarking or fingerprinting methods^{33;34}, where the buyer’s ownership information can serve as a unique identifier embedded into the distributed IP instance.

5.6 Design Pruning

Encoding a design as a pair of binary matrices $(P, N) \in \{0, 1\}^{n \times m}$ for property verification can sometimes incur unnecessary cryptographic overhead, as not all variables and clauses derived from the design are relevant to the property. To reduce the matrices’ size, design pruning is applied before oblivious IP selection. As described in Section 5.5.2, each design statement is translated into an SMT representation. Given a specific property, the IP vendor can perform a Cone-of-Influence reduction¹⁰³ through backtracking the transitive fan-in from the property signal and retaining only the property-relevant parts of the design to minimize m and n . SAT-based model checking has proved that COI reduction is particularly effective for control-dominated designs where the property depends on a small subset of signals, but provides limited reduction for datapath-intensive circuits where dependencies rapidly propagate through the design^{104;105}. In Section 5.7.1, our experimental results further confirm this observation.

Moreover, pruning is performed offline on the vendor side based on the user-provided property signal. Although the semantics of the property remain private to the user, the signal itself must be revealed to the vendor for COI analysis. To mitigate this disclosure, users may first submit pruning requests for multiple candidate designs as decoys to hide the actual verification target and then proceed with verification for the selected design.

5.7 Experiment

Testbed. We implement OT and 2PC components (oblivious design selection and secure verification) of the framework using the semi-honest 2PC library of the EMP-toolkit¹⁰⁶. We build our static analysis frontend on top of PyVerilog⁴¹ and apply Z3⁶⁶ to generate Boolean formula for both the designs and the properties. The experiment is conducted on an AWS `z1d.3xlarge` with 12 vCPUs and 96 GB memory.

Benchmarks. Table 5.3 lists all benchmark descriptions and detailed assertions. We verify bounded *assert properties* and *cover properties* on ANSI-C¹⁰⁷, OpenCores¹⁰⁸ and Trusthub¹⁰⁹. ANSI includes the corresponding assertions within the code, and Trusthub, where chip-level designs are annotated with known vulnerabilities that can be expressed as *cover property*. For example, we verify constraints on output signals and state transitions to ensure correct sequential behaviour in the finite state machines `b01` and `b02`. The `Rrobin` design serves as a round-robin arbiter, where mutual exclusion is enforced by asserting that two acknowledge outputs should never be high simultaneously. The `adbg` module, a JTAG-compliant TAP controller, includes a password check mechanism, and we verify that no invalid input can bypass this check to set `passchk = 1`. In the Trusthub benchmark, `RS232_T700` verifies whether an input sequence causes the transmitter to enter a Denial-of-Service (DoS) state. In `memctrl_T100`, we check whether writing `0x77` in the CSR register results in the flash sleep bit (`fs`) being erroneously set to 1. Lastly, `wb_conmax_T300` checks whether any crafted input pattern allows a master to alter its destination slave unexpectedly.

5.7.1 Performance Evaluation

We highlight two types of performance improvements in this evaluation: (1) time reductions due to design pruning, and (2) improvements from `hw-ppSAT`. For 13 IPs and their corresponding 26 (pruned) design pairs, 7 out of 13 IPs benefit directly from design pruning, achieving noticeable reductions in both variable and clause counts. In addition, 17 out of 26 instances (13 design pairs) demonstrate improved verification time with our control-flow-guided heuristics, as shown in Figure 5.5. Compared to the total solving time, the overhead introduced by the IP selection and demasking phases is negligible—typically accounting for less than a few percent of the overall runtime. This indicates that the main performance bottleneck lies in the SAT solving stage, where our heuristics yield the most substantial speedup.

Table 5.3: Design, Line of Code, Abstract Description and Assertion. We expect UNSAT for the negation of the assertion property conjoined with the design, and SAT for the counterexample derived from the conjunction of the cover property and the design.

Benchmark[Source]	LoC	Description	SVA Assertion
Rrobin ¹⁰⁷	59	round robin arbiter	assert (ack0==0 ack1==0)
b01 ¹⁰⁷	111	8-state Mealy FSM	assert (OVERFLW==1 -> ##2 (OVERFLW==0))
b02.1 ¹⁰⁷	81	7-state Moore FSM	assert (stato==D -> ##2 (stato==B))
b02.2 ¹⁰⁷	81	7-state Moore FSM	assert (U==1 -> ##1 U==0)
b02.3 ¹⁰⁷	81	7-state Moore FSM	assert (U==1 -> stato==B)
b03 ¹⁰⁷	119	4-request arbiter	assert (GRANT_0==0 GRANT_0==8 GRANT_0==4 GRANT_0==2 GRANT_0==1)
traffic_light ¹⁰⁷	52	3-state FSM with counter	cover (reset##5 (Light_Sign == YELLOW_LIGHT))
lock_case ¹⁰⁸	25	4-state Moore FSM	cover (unlock == 1)
Prep3 ¹⁰⁸	122	8-state Mealy FSM	assert (next_out == 8'h20 -> ##1 (next_out == 8'h11 next_out == 8'h30))
adbg ¹⁰⁸	531	JTAG controller	cover (next_passchk == 1)
memctrl_T100 ¹⁰⁹	1786	memory controller	cover (csr_r[2]==0 && fs ==1)
RS232_T700 ¹⁰⁹	231	uart transmitter	cover (state_DataSend==7)
wb_conmax_T300 ¹⁰⁹	7626	general round robin arbiter	cover (slv_sel != wb_addr_i[31:28])

Although the overall verification time can still be long for large and complex IPs, this cost is one-time and occurs only during the initial verification phase before the IP transaction.

Evaluation of Design Pruning We first evaluate the impact of pruning with ppSAT. Table 5.4 also reports the corresponding pruning-induced improvements for hw-ppSAT. Since pruning is solver-agnostic and fundamentally reduces the size of the input matrices processed by the solver, its impact is similar for both solvers, and we therefore omit a separate discussion.

As shown by the highlighted entries in Table 5.4, ppSAT achieves significantly faster performance on the pruned designs in 8 out of 13 benchmark pairs. This improvement is driven by the reduced formula size, which lowers both the overhead of oblivious design selection (OT time) and the 2PC workload (Solving time).

In particular, benchmarks such as b01 and Prep3 contain extraneous state machine logic that is not exercised by the property, allowing pruning to reduce the SAT solving time from 51853.60s to 2667.76s in b01, and from 20561.45s to 15176.60s in Prep3. The large IPs such as memctrl_T100 and wb_conmax_T300 with multiple control/data paths achieve orders-of-magnitude speedups. For memctrl_T100, both cases reach a timeout, while the timeout threshold is reduced from 14 million seconds to 24 thousand seconds. In the case of RS232_T700, although the design initially contains 1487 variables, only a subset related to the denial of service vulnerability is relevant, allowing a drastic formula reduction to just 225 variables and a run-time drop from 163885s to 31603.80s. In contrast,

Table 5.4: We summarize the end-to-end execution time of BlindMarket for 13 IP designs in both their original and pruned forms. **Var/Cls** denotes the number of variables and clauses in the verification formula ψ . **Design Size** represents the size of the encoded design transferred during OT, measured in 128-bit blocks. **OT(1/32)** reports the execution time of 1-out-of-32 OT. **Demask** measures the time required to reconstruct the selected design within the 2PC (Equation 5.1). For the SAT solving stage, **DLIS** records the number of giant steps taken under the standard **ppSAT** DLIS heuristic, and **ppSAT** reports the corresponding solving time. Similarly, **Ctrl** denotes the number of giant steps taken under **hw-ppSAT**, while **hw-ppSAT** reports the solving time using the control-flow-guided heuristic. The results show that combining design pruning with **hw-ppSAT** achieves faster verification times on 12 of 13 benchmarks. In the table, **yellow cells** represent time improvements from design pruning, and **green cells** represent time improvements from heuristic guidance.

Design	Bound	Var	Cls	Design Size(blocks)	OT (1/32)(ms)	Demask(s)	DLIS	ppSAT(s)	Ctrl	hw-ppSAT(s)
Prep3	3	876	2617	35266	146.37	80.06	911	20561.45	911	20561.45
Prep3_prune	3	771	2392	32116	142.23	63.56	818	15176.60	818	15176.60
RS232.T700	5	1487	5200	124512	513.56	290.69	20000*	1638850	2000	163885
RS232.T700_prune	5	225	708	2784	12.33	5.99	20000*	31603.80	222	350.80
Rrobin	6	154	387	1512	4.97	2.03	11561	6704.32	521	302.13
Rrobin_prune	6	154	387	1512	4.25	2.04	11561	6675.69	501	289.29
adbg	4	2186	13844	498168	2115.35	1148.19	1100	356573.80	1100	356573.80
adbg_prune	4	2103	13756	467500	1877.73	1094.98	1102	346138.20	1102	346138.20
b01	4	256	994	3952	16.81	9.52	20000*	51853.60	1978	5128.32
b01_prune	4	211	762	3024	12.08	6.28	1686	2667.76	1539	2435.16
b02_1	3	163	565	2132	9.28	3.33	108	97.80	108	97.80
b02_1_prune	3	144	478	1784	9.41	2.47	97	65.73	97	65.73
b02_2	2	118	395	720	3.80	1.64	298	138.30	204	94.67
b02_2_prune	2	118	395	720	3.17	1.62	298	140.27	204	96.02
b02_3	3	160	570	2132	8.77	3.24	1153	1080.95	178	166.88
b02_3_prune	3	160	570	2132	8.37	3.39	1153	1055.94	178	163.02
b03	7	1289	5026	109670	436.25	223.30	240	15413.88	240	15413.88
b03_prune	7	1289	5026	109670	430.14	226.00	240	15234.12	240	15234.12
lock_case	4	176	450	1728	9.43	2.87	20000*	15844.80	196	155.28
lock_case_prune	4	176	450	1728	7.36	2.92	20000*	15661.34	196	153.48
memctrl.T100	1	5183	12943	1059194	4469.61	2450.10	20000*	14628460	4365	3192661.40
memctrl.T100_prune1	1	268	458	2592	10.66	4.44	20000*	24927	236	294.14
traffic_light	5	590	2028	20190	91.33	41.60	2376	28052.48	1017	12007.31
traffic_light_prune	5	550	1948	19390	77.22	36.82	2142	22330.56	925	9643.22
wb_conmax_*	1	2781	5705	246972	1045.03	582.29	20000*	3264500	20000*	3264500
wb_conmax_*.prune	1	296	421	1974	8.24	3.63	2745	3391.09	384	474.38

* timeout when reaching 20000 giant steps;

smaller designs such as **b02_2**, **b03**, **Rrobin**, and **lock_case** exhibit little to no performance change, as they are single-purpose IPs with compact state machines that leave no room for structural reduction.

Evaluation of hw-ppSAT The integration of control-flow-guided heuristics in **hw-ppSAT** delivers a clear performance advantage across a majority of benchmarks. As seen in Table 5.4, 17 out of 26 instances exhibit faster solving times when switching from the default decision heuristic (**ppSAT**) to the control-aware version (**hw-ppSAT**). This speedup is further illustrated in Figure 5.5. This improvement is primarily driven by the reduced number of giant steps. For instance, in **b01_prune**, **hw-ppSAT** reduces runtime from 2667.76s to 2435.16s by cutting 147 steps, while in **RS232.T700_prune**, the step count drops from 20000(timeout) to 222, resulting in a total runtime reduction from 31603.80s to 350.80s. Similarly, the **traffic_light_prune** benchmark improves from 22330.56s to 9643.22s, and

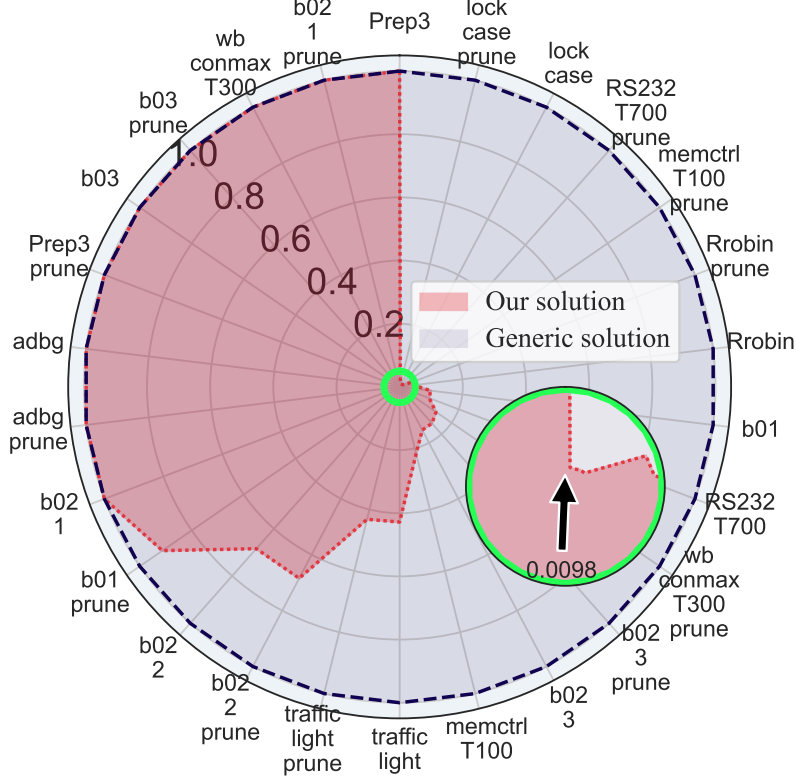


Figure 5.5: Illustrates the performance ratio between (generic)ppSAT and (our) hw-ppSAT across various (pruned) benchmarks. Each segment represents a specific design, with the outer boundary (ratio = 1.0) indicating the baseline solving time of ppSAT. The shaded region corresponds to hw-ppSAT; a smaller area indicates a greater speedup. As shown, most designs experience a substantial reduction in solving time, with normalized ratios significantly below 1.0. In the best-case scenario, our method achieves a remarkable speedup with a ratio as low as 0.0098.

memctrl_T100_prune from 24927s to 294.14s.

In 9 out of 26 instances, no runtime improvement is observed. Comparing the Ctrl and DLIS configurations shows that the proposed heuristic neither improves nor degrades performance in these cases. This behavior is inherent to SAT solving: heuristics are only invoked when the solver must choose the next undecided literal. For these instances, the solver can resolve the formula through propagation without making heuristic-driven decisions, and thus the heuristic is never applied during the solving process.

5.7.2 Communication Overhead

Figure 5.6a illustrates the communication overhead introduced by the 1-out-of-2^{depth} OT protocol. The x-axis represents the design size measured as the product of the number of variables and clauses

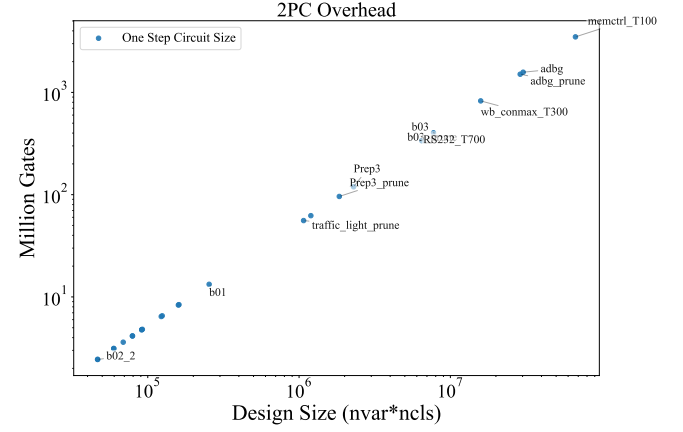
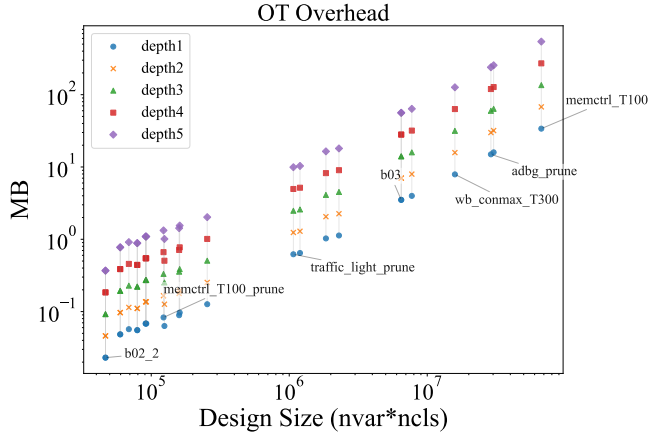
$(n_{var} \times n_{cls})$, while the y-axis reports the total OT cost in MB. Each point corresponds to a verification instance. As shown in Figure 5.6a, the OT communication overhead increases approximately linearly with the design size. As discussed in Section 5.2.2, each clause in the CNF-encoded design is represented as a pair of binary vectors of length n . Consequently, the OT overhead scales with $n_{var} \times n_{cls}$.

Furthermore, the OT overhead grows approximately linearly with 2^{depth} . In a 1-out-of- 2^{depth} OT protocol, increasing $depth$ enlarges the portfolio size $N = 2^{depth}$. As a result, more masked candidate messages must be communicated during the transfer. For the smallest benchmark, `b02`, with $depth = 1$, the OT communication overhead is approximately 0.022 MB, whereas for the largest benchmark, `mem_ctrl`, with $depth = 5$, the OT overhead reaches approximately 517 MB.

Figure 5.6b shows the cryptographic overhead of 2PC. This overhead primarily arises from `hw-ppSAT`, which adapts the DPLL algorithm as its core routine and compiles each *giant step* into the garbled circuit, where the communication overhead scales linearly with the number of AND gates (about 32 bytes per gate). Thus, the total communication cost is the per-step circuit cost multiplied by the number of *giant steps*. Similar to the OT protocol, the circuit size grows approximately linearly with the design size. Larger designs produce larger CNF encodings and require more evaluation during verification. For the largest design, `mem_ctrl`, the circuit size can reach around 3 billion gates per step, making secure verification impractical without optimization. After applying design pruning, we reduce the communication overhead to about 6 million gates per step. Furthermore, using the control-flow-guided heuristic, the number of giant steps is reduced from more than 20,000 to 236, making secure verification feasible.

5.7.3 Comparison with Non-private Baseline

We implement `hw-ppSAT` and `ppSAT` in Python and include the non-private baseline experiment in Figure 5.7. To ensure consistency with Table 5.4, the timeout is set to 20,000 giant steps, identical to that used in Table 5.4. Figure 5.7 shows that reaching 20,000 giant steps corresponds to approximately 100 seconds for the largest timed-out design. The results also indicate that solver runtime is largely dominated by the number of executed giant steps, as the baseline implementations follow the same DPLL core routine. The most time-consuming benchmark, `memctrl_T100`, finishes verification in 37.9 seconds, corresponding to an approximately 8.4×10^4 slowdown compared with the non-private baseline. In contrast, `b02_1_prune` finishes within 0.02 seconds, incurring an overhead of about 3×10^3 . As



(a) Communication overhead of the 1-out-of- 2^{depth} OT protocol versus design size ($n_{var} \times n_{cls}$). The overhead scales approximately linearly with design size and increases with portfolio size 2^{depth} .

(b) 2PC communication overhead per giant step measured by the garbled circuit size. The circuit size grows approximately linearly with the design size ($n_{var} \times n_{cls}$).

Figure 5.6: Communication overhead analysis of the proposed protocol. (a) shows the cost of the 1-out-of- 2^{depth} OT protocol as a function of design size, while (b) reports the 2PC overhead per giant step measured by garbled circuit size. Both results demonstrate near-linear scalability with respect to the design size ($n_{var} \times n_{cls}$).

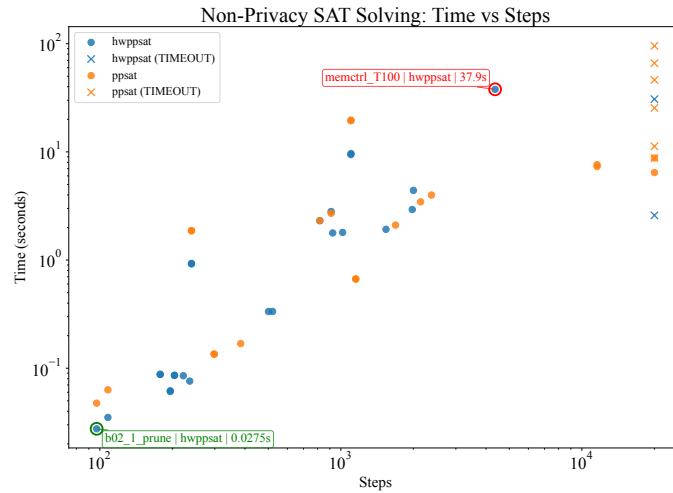


Figure 5.7: Runtime of the non-private baseline implementation of **hw-ppSAT** without 2PC. The results indicate that solver runtime is largely dominated by the number of executed steps.

expected, the non-private verification implementation is significantly faster than the privacy-preserving one due to the additional cryptographic operations. Moreover, the slowdown is not uniform across benchmarks and primarily depends on the design size, as larger designs incur substantially higher overhead in GC-based 2PC computation.

5.8 Conclusion & Discussion

This chapter introduces BlindMarket, the first end-to-end framework for addressing privacy concerns in hardware formal verification and ownership traceability after IP trading. We propose a zero-trust model that eliminates the need for a trusted third party while enabling formal logic verification, a capability not addressed by prior industrial or academic work. By leveraging 2PC and OT, we extend the state-of-the-art privacy-preserving SAT solver **ppSAT** to the hardware domain (**hw-ppSAT**) and apply optimizations that make IP-level verification practical. A practical challenge of BlindMarket lies in its interactive protocol, where stages such as oblivious design selection and secure SAT solving require synchronized message exchanges between the IP vendor and the IP user, introducing additional latency compared with offline verification. Despite this overhead, the framework remains compatible with existing post-licensing IP protection solutions and can be integrated into current hardware supply-chain ecosystems.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This dissertation begins with the problem of reasoning about security-relevant behavior in RTL and ends with a broader claim about what trustworthy hardware assurance must become in modern semiconductor ecosystems. Its main contribution is not merely the presentation of four separate techniques, but the demonstration that hardware trust must be preserved as the design moves across increasingly wider contexts: from a single RTL description into an integrated SoC, from hardware behavior into software-triggerable execution, from design intent into fabricated devices, and from proprietary implementation into commercial exchange among mutually untrusted parties. In this sense, the progression of the dissertation is also a progression in what formal verification is asked to accomplish. In the first stage, verification is used to detect hidden or malicious interactions that only arise after IP integration, as shown by If-Tracker’s emphasis on condition-aware information-flow reasoning rather than simple path existence. In the second stage, verification is used to explain how vulnerabilities become exploitable, as Microscope connects temporal hardware behavior with software-visible causes and turns formal analysis into a tool for causal understanding, debugging, and test generation. In the third stage, assurance is extended to authenticate the physical realization of a design, as Wing-RTL shows that even a correctly verified hardware design cannot function as a trustworthy root if the deployed device itself cannot be distinguished from a counterfeit. In the fourth stage, formal methods are further asked to enable secure cooperation under zero trust, as BlindMarket demonstrates that verification can serve not only as a correctness-checking mechanism but also as a privacy-preserving protocol that jointly supports

verifiability, confidentiality, and traceability in IP exchange. The broader implication is that formal methods remain highly valuable for hardware security when they preserve design semantics instead of abstracting them away too early, and when they are combined with domain-specific mechanisms such as static RTL analysis, causal modeling, attestation logic, and cryptographic protocols. What emerges from this dissertation, therefore, is a layered assurance methodology rather than a collection of isolated results. It suggests that future hardware trust will not be guaranteed by a single proof at a single stage of the lifecycle, but by verification flows that remain meaningful across design, integration, deployment, licensing, and long-term provenance. By establishing this view, the dissertation advances both a practical research agenda and a conceptual framework for understanding hardware trust as a cross-layer, cross-party, and lifecycle-wide problem.

6.2 Future Work

Although this dissertation advances formal approaches for hardware trust, several important directions remain open. A first step is to connect the proposed methods into a unified verification pipeline that spans RTL security analysis, hardware–software validation, supply-chain authentication, and protected IP exchange. Such an end-to-end framework would allow evidence generated at one stage to guide decisions at later stages and would move hardware assurance from isolated checks toward lifecycle-aware trust management.

A second direction is scalability. Future research should extend these methods to larger and more heterogeneous systems, including multi-IP SoCs, chiplet-based designs, accelerator-rich platforms, and longer firmware-driven execution traces. This will require stronger abstractions, better reduction techniques, and tighter integration with industrial verification flows so that the proposed methods remain practical for real-world hardware.

Future work is also needed in automation and stronger security guarantees. Security properties and assertions could be derived more automatically from architecture specifications, bug databases, and software behavior, reducing the manual effort required from verification engineers. At the same time, confidential verification protocols can be strengthened against more powerful adversaries, and authentication mechanisms can be linked more closely with provenance tracking so that trust can be maintained from IP selection to post-silicon deployment.

Overall, the next stage of this research is to move from effective point solutions for individual

trust boundaries toward a unified, scalable, and deployable infrastructure for hardware trust across the semiconductor lifecycle.

Bibliography

- [1] Sandip Ray, Eric Peeters, Mark M Tehranipoor, and Swarup Bhunia. System-on-chip platform security assurance: Architecture and validation. Proceedings of the IEEE, 106(1):21–37, 2017.
- [2] Semiconductor Industry Association et al. Global semiconductor sales, units shipped reach all-time highs in 2021 as industry ramps up production amid shortage. Haettu, 16:2022, 2022.
- [3] Nicole Fern, Ismail San, and Kwang-Ting Tim Cheng. Detecting hardware trojans in unspecified functionality through solving satisfiability problems. In 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2017.
- [4] Katharina Ceesay-Seitz, Flavien Solt, and Kaveh Razavi. μ cfi: Formal verification of microarchitectural control-flow integrity. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pages 213–227, 2024.
- [5] Xiaolong Guo, Raj Gautam Dutta, Jiaji He, and Yier Jin. Pch framework for ip runtime security verification. In 2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), pages 79–84. IEEE, 2017.
- [6] Xingyu Meng, Shamik Kundu, Arun K Kanuparthi, and Kanad Basu. Rtl-contest: Concolic testing on rtl for detecting security vulnerabilities. IEEE TCAD, 2021.
- [7] Qizhi Zhang, Liang Liu, Yidong Yuan, Zhe Zhang, Jiaji He, Ya Gao, Yao Li, Xiaolong Guo, and Yiqiang Zhao. A gate-level information leakage detection framework of sequential circuit using z3. Electronics, 11(24):4216, 2022.
- [8] Himanshu Jain, Daniel Kroening, Natasha Sharygina, and Edmund M Clarke. Word-level predicate-abstraction and refinement techniques for verifying rtl verilog. IEEE TCAD, 27(2): 366–379, 2008.
- [9] Clifford Wolf. Yosys open synthesis suite, 2016.

- [10] Rui Zhang, Calvin Deutschbein, Peng Huang, and Cynthia Sturton. End-to-end automated exploit generation for validating the security of processor designs. In 2018 51st Annual IEEE/ACM MICRO, pages 815–827. IEEE, 2018.
- [11] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In OSDI, volume 8, pages 209–224, 2008.
- [12] Wilson Snyder. Verilator: Open simulation-growing up. DVClub Bristol, 2013.
- [13] Cadence. JasperGold Platform and Formal Property Verification App User Guide, June 2021.
- [14] Cadence. Jaspergold spv app. URL https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/security-path-verification-app.html.
- [15] Synopsys. VC Formal Verification User Guide, December 2019. Version P-2019.06-SP2.
- [16] Pramod Subramanyan and Divya Arora. Formal verification of taint-propagation security properties in a commercial soc design. In 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1–2. IEEE, 2014.
- [17] Nusrat Farzana, Fahim Rahman, Mark Tehranipoor, and Farimah Farahmandi. Soc security verification using property checking. In 2019 IEEE International Test Conference, pages 1–10. IEEE, 2019.
- [18] Adib Nahiyan, Mehdi Sadi, Rahul Vittal, Gustavo Contreras, Domenic Forte, and Mark Tehranipoor. Hardware trojan detection through information flow security verification. In 2017 IEEE International Test Conference (ITC), pages 1–10. IEEE, 2017.
- [19] Dongfang Li, Wei Shen, and Zhihao Wang. A noval method of security verification for jtag protection function. In 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), pages 487–492. IEEE, 2019.

- [20] Nicole Fern, Ismail San, Cetin Kaya Koç, and Kwang-Ting Tim Cheng. Hiding hardware trojan communication channels in partially specified soc bus functionality. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36(9):1435–1444, 2016.
- [21] Madelyn Glymour, Judea Pearl, and Nicholas P Jewell. Causal inference in statistics: A primer. John Wiley & Sons, 2016.
- [22] Sean Gallagher. FTDI on counterfeit chip bricking: “Our intentions were honorable”. 10 2014. <https://arstechnica.com/information-technology/2014/10/ftdis-anti-counterfeit-in-g-efforts-sit-between-a-rock-and-a-hard-place/>.
- [23] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. FPGA Fingerprinting Techniques for Protecting Intellectual Property. In Proceedings of the IEEE Custom Integrated Circuits Conference, pages 299–302, 1998.
- [24] Tom Kean, David McLaren, and Carol Marsh. Verifying the Authenticity of Chip Designs with the DesignTag System. In IEEE International Workshop on Hardware-Oriented Security and Trust, pages 59–64, 2008.
- [25] Jo Vliegen, Md Masoom Rabbani, Mauro Conti, and Nele Mentens. SACHa: Self-attestation of Configurable Hardware. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 746–751. IEEE, 2019.
- [26] Reuters Staff. Former russian asml employee set for court hearing in alleged ip theft case. Reuters, 2024. URL <https://www.reuters.com/technology/former-russian-asml-employee-set-court-hearing-alleged-ip-theft-case-2024-12-09/>. Accessed: 2025-04-02.
- [27] Rishi Iyengar. Tesla sues former employee for allegedly stealing autopilot source code for chinese rival. CNN Business, 2019. URL <https://www.cnn.com/2019/03/22/tech/tesla-xiaopeng-motors-lawsuit/index.html>. Accessed: 2025-04-02.
- [28] Debora Halbert. Intellectual property theft and national security: Agendas and assumptions. The Information Society, 32(4):256–268, 2016.
- [29] Prabhat Mishra, Swarup Bhunia, and Mark Tehranipoor. Hardware IP security and trust. Springer, 2017.

- [30] Siladitya Ray. Samsung Bans ChatGPT Among Employees After Sensitive Code Leak, May 2 2023. Forbes.
- [31] Andrew B Kahng and et al. Constraint-based Watermarking Techniques for Design IP Protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20(10):1236–1252, 2001.
- [32] Yousra Alkabani and Farinaz Koushanfar. Active Hardware Metering for Intellectual Property Protection and Security. In USENIX Security Symposium (USENIX Security '07), volume 20, pages 1–20, 2007.
- [33] Hadi Mardani Kamali, Kimia Zamiri Azar, Farimah Farahmandi, and Mark Tehranipoor. Advances in Logic Locking: Past, Present, and Prospects. IACR Cryptol. ePrint Arch., 2022.
- [34] Rajat Subhra Chakraborty and Swarup Bhunia. HARPOON: An obfuscation-based SoC design methodology for hardware protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28(10):1493–1502, 2009.
- [35] IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP). IEEE Std 1735-2014 (Incorporates IEEE Std 1735-2014/Cor 1-2015), pages 1–90, 2015. doi: 10.1109/IEEESTD.2015.7274481.
- [36] Julian Speith, Florian Schweins, Maik Ender, Marc Fyrbiak, Alexander May, and Christof Paar. How Not to Protect Your IP—An Industry-Wide Break of IEEE 1735 Implementations. In 2022 IEEE Symposium on Security and Privacy (Oakland '22), pages 1656–1671. IEEE, 2022.
- [37] Maya Gebeily, James Pearson, and David Gauthier-Villars. How Israel’s bulky pager fooled Hezbollah. <https://www.reuters.com/graphics/ISRAEL-PALESTINIANS/HEZBOLLAH-PAGERS/mopawkkwjpa/>. Accessed April 3rd, 2025.
- [38] Ning Luo, Samuel Judson, Timos Antonopoulos, Ruzica Piskac, and Xiao Wang. ppSAT: Towards Two-Party Private SAT Solving. In 31st USENIX Security Symposium (USENIX Security 22), pages 2983–3000, 2022.
- [39] Cad for assurance. Axisoc, 2021. URL <https://cadforassurance.org/soc-platform/soc-benign-benchmark/system-on-chip-benchmarks/>.

- [40] Jiaji He, Xiaolong Guo, Travis Meade, Raj Gautam Dutta, Yiqiang Zhao, and Yier Jin. Soc interconnection protection through formal verification. Integration, 64:143–151, 2019.
- [41] Shinya Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In International Symposium on Applied Reconfigurable Computing, pages 451–460. Springer, 2015.
- [42] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. Hardfails: Insights into software-exploitable hardware bugs. In 28th USENIX Security 19, pages 213–230, 2019.
- [43] Arun Kanuparthi, Hareesh Khattri, Parbati Kumar Manna, and Narasimha Kumar V Mangipudi. Cwe-1193: Power-on of untrusted execution core before enabling fabric access control. 2019.
- [44] Nicole Fern. Cwe-1276: Hardware child block incorrectly connected to parent system. 2020.
- [45] Mitre Corporation. Cve-2017-0352: Incorrect access control in device leads to privilege escalation. 2017.
- [46] Xun Li, Mohit Tiwari, Jason K Oberg, Vineeth Kashyap, Frederic T Chong, Timothy Sherwood, and Ben Hardekopf. Caisson: a hardware description language for secure information flow. In ACM SIGPLAN Notices, volume 46, pages 109–120. ACM, 2011.
- [47] Xun Li, Vineeth Kashyap, Jason K Oberg, Mohit Tiwari, Vasanth Ram Rajarathinam, Ryan Kastner, Timothy Sherwood, Ben Hardekopf, and Frederic T Chong. Sapper: A language for hardware-level security policy enforcement. In ACM SIGARCH Computer Architecture News, volume 42, pages 97–112. ACM, 2014.
- [48] Danfeng Zhang, Yao Wang, G Edward Suh, and Andrew C Myers. A hardware design language for timing-sensitive information-flow security. ACM SIGPLAN Notices, 50(4):503–516, 2015.
- [49] Andrei Sabelfeld and Andrew C Myers. Language-based information-flow security. IEEE Journal on selected areas in communications, 21(1):5–19, 2003.
- [50] Xiaolong Guo, Raj Gautam Dutta, Jiaji He, Mark M Tehranipoor, and Yier Jin. Qif-verilog: Quantitative information-flow based hardware description languages for pre-silicon security as-

- essment. In 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pages 91–100. IEEE, 2019.
- [51] Questa-SecureCheck. Questa Formal Verification Apps, 2021. URL <https://eda.sw.siemens.com/en-US/ic/questa/formal-verification/secure-check/>.
- [52] Radix-S. Simulation Based Security Verification, 2021. URL <https://tortugalogic.com/radix-s/>.
- [53] Xiaolong Guo, Raj Gautam Dutta, Prabhat Mishra, and Yier Jin. Automatic code converter enhanced pch framework for soc trust verification. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(12):3390–3400, 2017.
- [54] Kan Xiao, Adib Nahiyan, and Mark Tehranipoor. Security rule checking in ic design. Computer, 49(8):54–61, 2016.
- [55] EM Benhani, Lilian Bossuet, and Alain Aubert. The security of arm trustzone in a fpga-based soc. IEEE Transactions on Computers, 68(8):1238–1248, 2019.
- [56] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtvushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In 28th USENIX Security Symposium (USENIX Security 19), pages 249–266, 2019.
- [57] Common weakness enumeration, 2022. <https://cwe.mitre.org/>.
- [58] Yunfeng Tao. An introduction to assertion-based verification. In 2009 IEEE 8th International Conference on ASIC, pages 1318–1323. IEEE, 2009.
- [59] Nicole Fern. Cwe-1262: Improper access control for register interface. 2020.
- [60] Arun Kanuparthi, Hareesh Khattri, Parbati Kumar Manna, and Narasimha Kumar V Mangipudi. Cwe-1234: Hardware internal or debug modes allow override of locks. 2020.
- [61] Arun Kanuparthi, Hareesh Khattri, Parbati Kumar Manna, and Narasimha Kumar V Mangipudi. Cwe-1245: Improper finite state machines (fsms) in hardware logic. 2020.

- [62] <https://github.com/darklife/darkriscv>.
- [63] <https://github.com/ucb-bar/riscv-mini>.
- [64] <https://github.com/openrisc/or1200>.
- [65] Baleegh Ahmad, Wei-Kai Liu, Luca Collini, Hammond Pearce, Jason M. Fung, Jonathan Valamehr, Mohammad Bidmeshki, Piotr Sapiecha, Steve Brown, Krishnendu Chakrabarty, Ramesh Karri, and Benjamin Tan. Don't cweat it: Toward cwe analysis techniques in early stages of hardware design. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392174. doi: 10.1145/3508352.3549369. URL <https://doi.org/10.1145/3508352.3549369>.
- [66] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 337–340. Springer, 2008.
- [67] Wei Hu, Armaiti Ardeshiricham, and Ryan Kastner. Hardware information flow tracking. ACM Computing Surveys (CSUR), 54(4):1–39, 2021.
- [68] Yonghwi Kwon, Dohyeong Kim, William Nick Sumner, Kyungtae Kim, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. Ldx: Causality inference by lightweight dual execution. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, pages 503–515, 2016.
- [69] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela F Ciocarlie, et al. Mci: Modeling-based causality inference in audit logging for attack investigation. In NDSS, volume 2, page 4, 2018.
- [70] Haocheng Ma, Jiaji He, Yanjiang Liu, Leibo Liu, Yiqiang Zhao, and Yier Jin. Security-Driven Placement and Routing Tools for Electromagnetic Side-Channel Protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(6):1077–1089, 2020.
- [71] Gloria Ciavarrini, Maria S Greco, and Alessio Vecchio. Geolocation of Internet Hosts: Accuracy Limits through Cramér–Rao Lower Bound. Computer Networks, 135:70–80, 2018.

- [72] Orlando Arias, Zhaoxiang Liu, Xiaolong Guo, Yier Jin, and Shuo Wang. RTSec: Automated RTL Code Augmentation for Hardware Security Enhancement. In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 596–599. IEEE, 2022.
- [73] Benjamin John Rosser. Cocotb: a Python-based digital logic verification framework, 2018.
- [74] A. Cui, C. Chang, S. Tahar, and A. T. Abdel-Hamid. A Robust FSM Watermarking Scheme for IP Protection of Sequential Circuit Design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30(5):678–690, 2011.
- [75] Mohammed Shayan, Kanad Basu, and Ramesh Karri. Hardware Trojans Inspired IP Watermarks. IEEE Design & Test, 36(6):72–79, 2019.
- [76] Upoma Das, M Sazadur Rahman, N Nalla Anandakumar, Kimia Zamiri Azar, Fahim Rahman, Mark Tehranipoor, and Farimah Farahmandi. PSC-Watermark: Power Side Channel based IP Watermarking Using Clock Gates. In 2023 IEEE European Test Symposium (ETS), pages 1–6. IEEE, 2023.
- [77] Patanjali SLPSK, Abhishek Anil Nair, Chester Rebeiro, and Swarup Bhunia. SIGNED: A Challenge-Response Scheme for Electronic Hardware Watermarking. IEEE Transactions on Computers, 2022.
- [78] Priyanka Bagul, Vandana Inamdar, et al. Hardware Obfuscation based Watermarking Technique for IPR Ownership Identification. International Journal of Reconfigurable Computing, 2023.
- [79] Joonho Kong, Farinaz Koushanfar, Praveen K Pendyala, Ahmad-Reza Sadeghi, and Christian Wachsmann. PUFatt: Embedded Platform Attestation based on Novel Processor-based PUFs. In 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2014.
- [80] Xilinx Inc. XA Artix-7 FPGAs Data Sheet: Overview, November 2017. DS197 (v1.3).
- [81] Arm Ltd. Arm Cortex-M on FPGA, Accessed 2021. <https://www.arm.com/resources/free-arm-cortex-m-on-fpga>.
- [82] Amelia Dobis. Formal verification of hardware using mlir. Master’s thesis, ETH Zurich, 2024.

- [83] Srikanth Vijayaraghavan and Meyyappan Ramanathan. A Practical Guide for SystemVerilog Assertions. Springer Science Business Media, 2005.
- [84] Andrew Chi-Chih Yao. How to generate and exchange secrets. In Annual Symposium on Foundations of Computer Science. IEEE, 1986.
- [85] Mohammad Hashemi, Steffi Roy, Fatemeh Ganji, and Domenic Forte. Garbled EDA: Privacy Preserving Electronic Design Automation. In 2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD '22), pages 1–9, 2022. To appear.
- [86] Dimitris Mouris, Charles Gouert, and Nektarios Georgios Tsoutsos. MPloC: Privacy-Preserving IP Verification using Logic Locking and Secure Multiparty Computation. Cryptology ePrint Archive, 2023.
- [87] Dimitris Mouris and Nektarios Georgios Tsoutsos. Pythia: Intellectual Property Verification in Zero-Knowledge. In 2020 57th ACM/IEEE Design Automation Conference (DAC '20), pages 1–6, 2020. doi: 10.1109/DAC18072.2020.9218639.
- [88] Animesh Chhotaray, Adib Nahiyan, Thomas Shrimpton, Domenic Forte, and Mark Tehranipoor. Standardizing Bad Cryptographic Practice: A Teardown of the IEEE Standard for Protecting Electronic-Design Intellectual Property. In ACM SIGSAC CCS '17, pages 1533–1546, 2017.
- [89] Md Nazmul Islam and Sandip Kundu. Enabling ic traceability via blockchain pegged to embedded puf. ACM Transactions on Design Automation of Electronic Systems (TODAES), 24(3):1–23, 2019.
- [90] Chandan Kumar Chaudhary, Urbi Chatterjee, and Debdeep Mukhopadhyay. Auto-pufchain: an automated interaction tool for pufs and blockchain in electronic supply chain. In 2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), pages 1–4. IEEE, 2021.
- [91] Shashank Balla, Yiming Zhao, and Farinaz Koushanfar. Ictoken: An nft for hardware ip protection. arXiv preprint arXiv:2412.06726, 2024.
- [92] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. arXiv preprint arXiv:2105.07447, 2021.

- [93] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK), volume 13, page 14, 2010.
- [94] Clark W Barrett, David L Dill, and Jeremy R Levitt. A decision procedure for bit-vector arithmetic. In Proceedings of the 35th Annual Design Automation Conference, pages 522–527, 1998.
- [95] Grigori S Tseitin. On the complexity of derivation in propositional calculus. Automation of reasoning: 2: Classical papers on computational logic 1967–1970, pages 466–483, 1983.
- [96] V Aho Alfred, S Lam Monica, and D Ullman Jeffrey. Compilers principles, techniques & tools. pearson Education, 2007.
- [97] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. Handbook of satisfiability, pages 131–153, 2009.
- [98] Marijn JH Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In Haifa Verification Conference, pages 50–65. Springer, 2011.
- [99] Chi-An Wu, Ting-Hao Lin, Chih-Chun Lee, and Chung-Yang Huang. Qutesat: a robust circuit-based sat solver for complex circuit structure. In 2007 Design, Automation & Test in Europe Conference & Exhibition, pages 1–6. IEEE, 2007.
- [100] He-Teng Zhang, Jie-Hong R Jiang, and Alan Mishchenko. A circuit-based sat solver for logic synthesis. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–6. IEEE, 2021.
- [101] Jianhui Chen and Fei He. Control flow-guided smt solving for program verification. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pages 351–361, 2018.
- [102] Ofer Shtrichman. Tuning sat checkers for bounded model checking. In Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12, pages 480–494. Springer, 2000.

- [103] Sergey Berezin, Sérgio Campos, and Edmund M Clarke. Compositional reasoning in model checking. In International Symposium on Compositionality, pages 81–102. Springer, 1997.
- [104] Ken L McMillan. Applying sat methods in unbounded symbolic model checking. In International Conference on Computer Aided Verification, pages 250–264. Springer, 2002.
- [105] Edmund M Clarke. Model checking. In International conference on foundations of software technology and theoretical computer science, pages 54–56. Springer, 1997.
- [106] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. Emp-toolkit: Efficient multiparty computation toolkit, 2016.
- [107] Rajdeep Mukherjee, Michael Tautschnig, and Daniel Kroening. v2c – A Verilog to C Translator Tool . In Tools and Algorithms for the Construction and Analysis of Systems (TACAS '16), pages 580–586. Springer, 2016. ISBN 978-3-662-49673-2.
- [108] Various Contributors. OpenCores.org, 2021.
- [109] <https://www.trust-hub.org/>.