Reliable Hardware Trojan Formal Verification

Czea Sie Chuah[1], Christian Appold[2*] and Tim Leinmüller[2]

[1] Technical University of Munich, Germany
[2]DENSO AUTOMOTIVE Deutschland GmbH, Germany

Abstract

Due to cost and time reasons, hardware development is currently often done using IP of external vendors and outsourcing of fabrication to third parties. The globalization of the hardware supply chain, which is inherent in the RISC-V ecosystem, increases the risk of malicious insertions of Hardware Trojans (HTs) into the design. HTs are malicious modifications of hardware to change functionality or to leak secret data. Especially in safety- and security-critical areas like autonomous driving, HTs can cause severe consequences and even endanger human lifes. We research how model checking based formal verification can be used in a systematic way to detect each HT inserted during hardware design reliably. We are the first to use signal connection properties on top of design functionality properties for more reliable HT detection. To reduce verification time and strongly increase HT detection coverage, we developed a tool to generate a high number of properties fully automatically. Additionally, we will publish a large set of specification and microarchitecture intent derived properties important for HT detection in processors and show how these properties have to be combined with connection properties for HT detection. Our work results in a guideline for reliable HT detection in processors using formal verification.

Introduction

Over the past decades, hardware security vulnerabilities have raised concerns, especially in critical applications like autonomous driving and factory automation. Hardware Trojans (HTs) are malicious modifications of hardware that cause undesired behavior. Outsourcing hardware design and manufacturing have increased the risk of HTs, particularly in processors. They are vulnerable and widely used in various devices. With the rise of processors with open RISC-V ISA, the risk of HTs is even higher, as ISA accessibility makes it easier to find and exploit security vulnerabilities.

Formal verification techniques for HT detection have been researched in recent years [1, 2]. However, most of these methods require significant manual effort and lack automation [3, 4]. This limits their scalability and effectiveness resulting in low detection coverage. In [5] security assertions are automatically generated and example HTs detected, but they don't target reliably detecting each inserted HT. Our approach offers greater automation and saves time. HT detection coverage is improved by introducing the use of signal connection properties. We will present key design intent properties to safeguard RISC-V processors from HTs and provide guidelines for combining these properties with connection properties. To demonstrate the completeness of the property set for HT detection, we developed a generic HT model and write the HT detection properties that each generic HT is detected. The property set can be created during hardware design and used by processor vendors to implement reliable HT formal verification. IP vendors can deliver the properties with IP, enhancing security of the IP.

*Corresponding author: c.appold@eu.denso.com

Methodology

In this work we present an approach enabling detection of each inserted HT for targeted processor constituents which we protect with our properties. For this, we use a combination of two types of properties:

1. **Design Intent Properties:** Derived from specification and microarchitecture intent. Used to verify implemented design functionality, for which being HT-free is essential. The properties must be written to verify corresponding functionality exhaustively for each signal value change and also when a signal should stay stable. As an example, for the Program Counter (PC) we use the following property for each possible condition to set it:
```
assert property (@(posedge clk)
writeWbPipelineReg && conditionPCSetting
|-> ##1 wbRegInstPC == desiredCondPC)
```
2. **Connection Properties:** These are properties proving pure signal connections with or without a storage element in between are free of inserted HTs. Our work is the first proposing signal connection verification for HT detection. In the following is an example property for a direct signal connection from the PC in instruction queue to memory without storage element in between:
```
assert property (@(posedge clk)
instQueuePC==memOutPC)
```

For HT formal verification, design intent properties alone are insufficient. Using only them for proving proper PC setting does not ensure the PC is as desired after the position where the PC signal value is proven. For example the PC could be compromised on its way to instruction memory. Also, the PC immediate in the instruction word coming from memory could be modified in the pipeline. Hence, to prove HT absence

without detection gaps, we recommend additional use of connection properties. Proper combination and connection of both property types enables reliable HT detection. The approach works for arbitrary hardware designs, but we focus on HT detection in processors.

We research a comprehensive property set for gap-free HT verification combining our two property types and are the first to suggest using formal verification also to verify the completeness of a property set for reliable HT detection. For this, we developed a generic HT model able to model an arbitrary HT modifying signal values. The HT model uses a new free input (*TROJ_input*) into the hardware design. During verification the formal verification tool assigns each possible value to this input in each hardware state. Our generic HT modifying an example signal *sig1* looks as follows:

```
assign sig1_mod = sig1 XOR TROJ_input
```

With this, the signal *sig1_mod* is modified in each state of the hardware with every possible arbitrary value during verification and used instead of the signal *sig1* in the design. In formal verification we use dedicated properties to ensure every possible triggered HT, modeled through our generic HT model at different positions in the processor, is detected through our property set. This allows us to prove each possible inserted HT is detected through the property set.

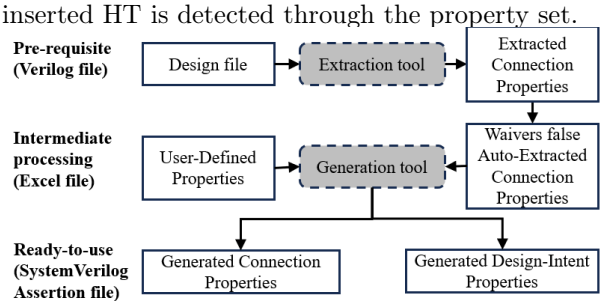

Figure 1: *HT detection property generation approach*

Property Generation

Reliable HT detection requires a large number of properties. We implemented an automatic property generation tool to speed up HT detection using our approach. The inputs needed for our tool are the hardware design, user-defined connection and design intent properties, and waivers for false auto-generated properties (see Figure 1). Hardware designs typically have a large number of signal connections. Our tool fully automatically generates connection properties for all signal connections between input and output ports of modules without intermediate logic or storage elements in between only from hardware design in RTL. This is done for all modules and submodules, which significantly increases coverage of design parts verified for HT insertion, enabling detection of all inserted HTs in those connections through formal verification with generated properties. Tool outputs are fully auto-generated connection properties not disabled by waivers and user-defined connection and design intent properties, all in SystemVerilog Assertions (SVA).

Results and Conclusion

In this work we develop an approach for reliable HT detection using formal verification. We implemented an automated property generation tool and are the first to incorporate connection property checks into HT verification. Our tool automatically generates a large number of connection properties, allowing the corresponding signal connections to be easily verified for HT insertion absence. We propose applying this automated connection checking approach as a general security rule for HT absence in hardware designs. Additionally, we have developed significant parts of the necessary user-defined properties for reliable HT detection in PC and privilege mode processor constituents, and will continue to expand the property set until it provides full protection against HTs.

The work is executed on a 32-bit 4-stage pipeline RISC-V processor. For our formal verification experiments, the tool Cadence Jasper Formal Verification Platform [6] is used. All our design intent properties could be proven in below 24 hours. From the RTL code of our processor, our tool generated roughly 1500 connection properties fully automated. Verification experiments show that the runtime of formal verification is below a minute for connection properties. Some of the properties to prove arbitrary HT detection capability by our properties are proven in below 24 hours. For the rest, we intend to apply advanced proof techniques to reduce them below 24 hours runtime. The experimental results confirm our approach is promising for scalable reliable HT detection. In future we aim to present a comprehensive property set for processors, to verify key design constituents for HT absence and ensure suitability of our property set for reliable HT detection using our generic HT modeling.

References

[1] M. Rathmair, F. Schupfer, and C. Krieg. "Applied formal methods for hardware Trojan detection". In: *Int. Symp. on Circuits and Systems (ISCAS)*. 2014.

[2] B. Mainak and M. S. Hsiao. "Trusted RTL: Trojan detection methodology in pre-silicon designs". In: *HOST*. 2010.

[3] R. Jeyavijayan, V. Vivekananda, and K. Ramesh. "Detecting malicious modifications of data in third-party intellectual property cores". In: *DAC*. 2015, pp. 1–6.

[4] X. Zhang and T. Mohammad. "Case study: Detecting hardware Trojans in third-party digital IP cores". In: *Int. Symp. on Hardware-Oriented Security and Trust*. 2011.

[5] M. R. H. Iman et al. "Processor Vulnerability Detection with the Aid of Assertions: RISC-V Case Study". In: *Nordic Circuits and Systems Conf. (NorCAS)*. 2024, pp. 1–7.

[6] *Jasper RTL Apps | Cadence Design Systems*. 2025. URL: https://www.cadence.com/en%5C_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-verification-platform.html.