Hardware property checker for run-time Hardware Trojan detection

Xuan Thuy Ngo, Jean-Luc Danger, Sylvain Guilley, Zakaria Najm, Olivier Emery

Hardware Property Checker for Run-Time Hardware Trojan Detection

Xuan Thuy Ngo*, Jean-Luc Danger*†, Sylvain Guilley*†, Zakaria Najm* and Olivier Emery†
* Institut MINES-TELECOM, TELECOM ParisTech, CNRS LTCI (UMR 5141), 75 634 PARIS Cedex 13, FRANCE.
† Secure-IC S.A.S., ZAC des Champs Blancs, 15, rue Claude Chappe – Bât. B, 35 510 Cesson-Sévigné, FRANCE.

***Abstract*—Nowadays, Hardware Trojans (HTs) become a real threat because of IC design and fabrication outsourcing trend. In the state of the art, many efforts were devoted to counter this threat, especially at netlist level. However, some clever HTs are actually a combination between a hardware and a software vulnerability, which, together, allow an exploitation. In this paper, we intend to detect such advanced HT, by resorting to a run-time detection. This method consists in identifying some high-level and critical behavioral invariants, and by checking them during the circuit operation. The assertion and Property Specification Language (PSL) is used to describe the properties to be checked. Then, a Hardware Property Checker (HPC) is created and integrated in the IC in order to verify these properties in run-time. We discuss how to define the critical properties for HPC. We also explain how this method is complementary with others, especially how the Hardware Checker can itself be protected against a tampering attempt. A case of study on LEON processor was performed to demonstrate the feasibility of this detection technique.**

***Index Terms*—Hardware Trojans (HT), Run-time detection, Hardware Property Checker (HPC), Assertions, Property Specification Language (PSL).**

I. Introduction

Nowadays, more and more semiconductor companies outsource their IC designs and fabrication steps because of their high cost and complexity. Nevertheless, this trend opens the door for a dangerous attack named Hardware Trojan (HT) insertion. A HT is a malicious module inserted in the original Integrated Circuit (IC) during at design or fabrication stage. A properly inserted HT can effectuate various dangerous tasks like Denial of Service, leakage of sensible data via circuit outputs, etc [9]. A HT can be globally seen as a composition of two parts: a trigger, which *reads* the target circuit state, and a payload, which *writes* on the target circuit state (to run its malicious function). Once inserted, we cannot remove a HT. Therefore HT has become a hot topic in the hardware security field. Because of its dangerous nature, many works have investigated detection methods.

In the state of the art (Fig. 1), there are two main protection approaches against HT: **detection** and **prevention**.

In the prevention technique, the goal is to modify the original circuit to either assist another protection technique or obfuscate the original logic part. The obfuscation technique ensures that attacker dont have enough information to insert an efficient HT [12], [4].

Xuan Thuy Ngo is the corresponding author. This project has been funded by the French Government, under grant FUI #14 **HOMERE** 959 (**H**ardware tr**O**jans : **M**enaces et robust**E**sse des ci**R**cuits int**E**grés).

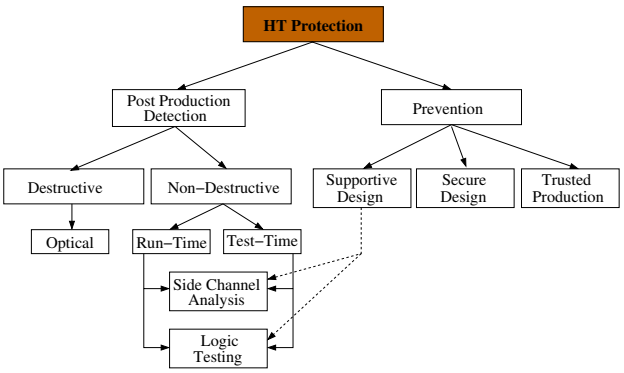

Fig. 1. Hardware trojan protection techniques

Detection methods encompass optical, test-time or run-time approaches. In deep sub-micron CMOS processes, the optical method uses destructive reverse engineering techniques to find out the HT [13]. The test-time detection is based on several techniques like hardware software co-design [6], reconfigurable logic [1], logic testing [3] and side-channel analysis [2].

Regarding the run-time method, Huffmire et al. specify legal memory access policies for FPGA-based embedded systems [8]. The policies are synthesized into a reconfigurable hardware module that decides the legality of every memory access request generated from a datapath module. This work was further developed into a method of generating hardware-based security checkers to detect processor malicious inclusions at runtime [5]. Security-relevant invariants of a processor's architectural specification are described on corresponding circuits of the processor's design using Property Specification Language assertions. Security checkers are then automatically generated as synthesizable hardware to verify linear temporal logic properties of expected behaviors.

In this paper, we also focus on the run-time checker approach, which is recently used for faults detection, to detect the HT during the IC operation. The goal is to create a synthesizable assertions module Hardware Property Checker (HPC) which verifies the permitted and prohibited behaviors of IC at run-time. In the previous works [5], authors concentrated rather on PSL to HDL conversion tool rather than HT detection. In this article, we show how to define the completed

properties list which will be checked by the HPC for HT detection. Moreover we propose three original approaches to protect HPC itself against HT.

The rest of this paper is structured as follows. Sec. II gives the summary of assertions, Property Specification Language (PSL) and how the HPC can be inserted in the design flow. Sec. III discusses about how to define the sensitive properties that will be checked by HPC for HT detection. Sec. V proposes some approaches for HPC protection against HT attack. Finally we conclude in Sec. VI.

II. Hardware Property Checker

A. Assertion and Property Specification Language

Assertions have been used by chip designers for many years to aid in the evaluation of functional correctness, and so-called assertion-based verification is common in the processor design industry today.

Hardware assertions differ from software assertions in an important way, because hardware languages are primarily based on constructs that execute in parallel, whereas most software languages describe execution that is fundamentally sequential. In general, a software assertion is checked when the program execution arrives at the assertion point, but a hardware assertion is continuously evaluated, in parallel with the rest of the system's execution.

Before the advent of hardware verification, assertions had long been used to verify the correctness of software programs, and the idea has been around for even longer. The assertion concept was introduced by Goldstine and von Neumann in 1947, according to Jones. PSL evolved from a language called Sugar, developed by IBM. Sugar, used for model-checking, was so-named because it featured a great deal of "syntactic sugar," so that temporal logic formulas could be written in a way that is more easily understood. The initial standardization effort was led by Accellera, and PSL was standardized by IEEE in 2005; the most recent version was approved in 2010 under number `1850-2010`. PSL is a language for the formal specification of hardware. It is used to describe properties that are required to hold in the design under verification. PSL provides a means to write specifications that are both easy to read and mathematically precise. It is intended to be used for functional specification on the one hand and as input to functional verification tools on the other. Thus, a PSL specification is an executable specification of a hardware design.

B. Hardware Property Checker

As discussed in the previous section, the assertions or PSL allows to check the permitted and prohibited circuit properties. And in many HT insertion examples [10], [9], attackers try to modify the behaviour or violate the property of the target circuit[1]. Therefore, assertions are an adapted approach for HT detection. The main idea is to find out the critical properties of the target circuit which can detect potentially the HT payload.

[1]Exceptions are *kill-switches*, which merely destroy the targetted chip.

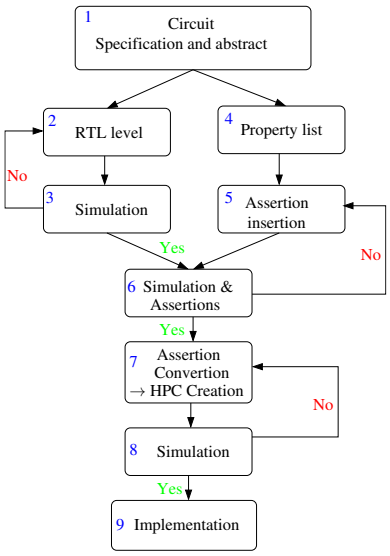

Fig. 2. Design Flow with Hardware Property Checker

Then, a hardware module, that we called Hardware Property Checker (HPC), is synthesized which checks dynamically these IC properties at run-time. The Figure 2 presents the modified design flow for HPC integration. It is composed of the following steps:

1) Define the specification of the target circuit.
2) Transfer to RTL level.
3) Simulate the RTL of target circuit.
4) Define the critical properties list based on target circuit specification.
5) Create the corresponding assertions for these properties using PSL.
6) Simulate these assertions.
7) Convert these assertions to a synthesizable HDL module and create HPC.
8) Simulate the HPC.
9) Implement the target circuit and HPC.

This technique is based on one main feature: the properties conversion into the synthesizable module HPC. In the state of the art, there are several works on the run-time checker for acceptance of properties for faults detection at run-time. Example are the construction of "Testers" [11] for dynamic verification or "SynPSL" [7] which translates logic formulas into equivalent synthesizable HDL entities. Recently, in [5], Bilzor et al. propose a tool, named "`psl2hdl`" which translates directly PSL syntax into equivalent HDL entities. But they focus on the `psl2hdl` tool development and give just a small example for HT detection. In this paper, we apply these tools to generate the HPC for HT detection purpose. We also explain how to define the complete Hardware Property Checker for HT detection and propose some protection for HPC against HT attack.

III. Properties Definition for HPC

The goal of HPC is to verify critical properties. Any HT which violates these properties will be detected. However, how we can define these properties? These properties can be found thanks to two approaches: Using **circuit specification** or studing the **existing HTs** in the state of the art.

A. Properties Definition with IC Specification

Any IC comes with a technical specification and documentation which defines its purposes and constraints. So it is the main source of the properties definition needed for HT detection. The technical documentation must be studied carefully in order to identify every implicit/explicit properties. For example on a processor, these properties can be related to:

- **Supervisor / user mode**.
- **Memory access conditions**.
- **Instruction pipeline specification**.
- **On-chip buses rules**.
- etc.

From one circuit to another, there can be different properties. A complete study must be investigated for a given circuit in order to define all sensitive properties. These sensitive properties will also depend on the expected security level.

B. Properties Definition with HT list

In this approach, we list all existing HTs. We study all information about their trigger and payload parts. Then we find how the target circuit will be affected by these HTs. After that, we define the corresponding properties to detect these HTs. It can be formalized as:

- Studying the state of the art of HT trojan insertion. For example, many HT insertions can be found on the website www.trust-hub.org for different target circuits.
- Keeping the HTs which infect/modify the target circuit.
- Defining the corresponding properties using the target circuit documentation and HT architecture.

Note that test techniques can not verify these properites which will be active after HT activation. Therefore runtime checker is preferable.

IV. Example of HC on LEON Processor

In this section, we demonstrate an example of Hardware Checker construction. The target circuit is the LEON Processor. It is a 32-bit processor with SPARC V8 instruction set. It has a 7-stage pipeline with separated instruction and data caches. More information about this processor can be found on the link http://www.gaisler.com/index.php/products/processors/leon3. In this experiment, we decided to implement a HC to check some properties in the Integer Unit of LEON Processor. Using the approach discussed in III-A, the following properties will be checked:

- **Supervisor Mode**: The user/supervisor mode is changed to supervisor when there is a trap or when there is a reset.
- **WRPSR instruction**: This is a privileged instruction that can be only executed when the processor is in supervisor mode.
- **Illegal instructions**: Illegal instructions can not be executed in the processor.

These properties are found in the SPARC Architecture Manual V8 at this URL: http://www.gaisler.com/doc/sparcv8.pdf. The following PSL codes are inserted on the LEON processor to verify these properties.

```
// supervisor property check
   -- PSL property su_enable_1 is
      always (rst -> sregs.s);
   -- PSL property su_enable_2 is
      always (sregs.et ->sregs.s);
// WRSPR instruction check
   -- PSL property PSR_write is
      always ((op_sig = FMT2)
      and ((op3_sig = WRPSR)
      and (sregs.s ='0')) -> priv_inst);
// Illegal instructions check
   -- PSL property illegal_inst_test is
      always((op2_sig = "001") or
      (op2_sig = "101") -> inllegal_inst);
```

IP	Max Frequency (MHz)	Nb of LUT	Nb of Registers
Original LEON	154	6841	2553
LEON + HC	154	6849	2553

TABLE I
LEON and HC synthesis on Virtex 5LX30 Xilinx FPGA

Then the HC corresponding to these PSL codes are instantiated. This HC is synthesized so that to evaluate its overhead. The Tab. I presents the synthesized results of LEON with and without HC on Virtex 5LX30 Xilinx FPGA. The synthesis shows that the HC is implemented using 8 LUTs. It represents only 0.11% of LEON processor, which is very small comparing to the protected circuit. However, this HC might detect many HTHs trying to alter or modify the Integer Unit.

V. Hardware Property Checker Protection

The HPC allows to detect HTs which are inserted at RTL level before HPC integration. But it is vulnerable against an attacker who inserts a HT at netlist or layout level. This is logical because he has access to the original layout and also the HPC. And an intelligent attacker, who reverses successfully the HPC, can always insert an HT which bypass the properties on the Checker. Or worse, he can even insert a HT attacking directly the HPC. This is the main problem of any runtime detection technique: **How can we protect the checker itself against HT insertions?**. In this subsection, we give some suggestions for HPC protection.

1) Using the 3-D circuit: A "three dimensional integrated circuit" (3-D IC) is an integrated circuit manufactured by stacking silicon wafers and/or dies and interconnecting them vertically using through-silicon vias (TSVs). So that they behave as a single device to achieve performance improvements at reduced power and smaller footprint than conventional two dimensional processes. This integration technique can be applied for HPC protection using the following steps:

- Create the HPC for a target circuit.
- Separate the HPC from the target circuit.
- Create the 3-D circuit for the target circuit and HPC with the following constraints: the **target circuit and HPC are placed on different dies**.
- Manufacture separately the target circuit and the HPC dies.
- Assembly the target circuit and the HPC.

2) Reconfigurable Hardware Property Checker: One of solution for HPC protection is creating a reconfigurable HPC. It consist in inserting a small reconfigurable FPGA on the target circuit. Then this FPGA will be used to implement the HPC. It could be possible because of the HPC size. Generally, HPC size is much smaller than the target circuit. For example in [5], a HPC containing 10 sensitive properties has 0.03% of overhead for an OpenRisc processor. And a complete HPC may have an overhead between 1% to 5%. So the utilization of an FPGA for HPC, in this case, is much smaller than for the whole target circuit. With this approach, we can configure dynamically HPC. Another advantage is the ability to verify properties sequentially and also new properties can be checked even after HPC deployment. This solution is flexible, but does not allow to check the properties in parallel. The next solution permits parallel and static verifications.

3) Encoded Circuit Method for HPC Protection: The "Encoded Circuit", proposed in [4], is a prevention method against HT insertion. This technique encodes and masks all internal registers (including control and data registers) with a Linear Complementary Pair (LCP) of codes C and D. It ensures that any HT connected to strictly less than $d_{trigger}$ registers ($d_{trigger}$ is the dual distance of D code used as security parameter) will be ineffective. And a HT which modifies less than $d_{payload}$ bits ($d_{payload}$ is the minimal distance of C) will create a wrong codeword. The main drawback of this method is the overhead of $\times 6$ for a circuit checker of [5]. But it can be perfectly adapted for HPC protection with following reasons:

- According to [5], the runtime checker for each property is mainly composed of one or some flip-flops. Now the main idea of the encoded circuit is to encode and mask all registers. Therefore, we can use encoded circuit method to encode and mask all flip-flops of HPC.
- The encoded circuit has an important overhead of $\times 6$. But an complete HPC for OpenRisc has an extra logic between 1% to 5% [5]. Then the HPC protection with encoded circuit has an overhead between 6% to 30%. It is an acceptable price for the security.

Each proposition approach has its own advantages and drawbacks. Their applications will be dependent for the target circuit and the security level. In some cases, a combination of these approaches could be used to ensure the IC security.

VI. Conclusions

HT insertion has become a serious issue with the globalization of semiconductor industry. In this paper, we propose an technique to detect the HT at run-time. This method consists in identifying some high-level and critical behavioral invariants, and by checking them during the circuit operation. Such method is validated by the "assertion" approach which is frequently used to check IC operation at simulation phase. Then, a Hardware Property Checker is created and integrated in the IC to verify these properties at run-time. We discuss how to define the complete list for the sensitive properties against HT insertion using the IC specification and existing HTs. A case of study on LEON processor shows the feasibility of this technique. We also propose to detect HPC using three different approaches: 3-D circuit, configurable HPC and "encoded circuit" protection.

References

[1] Miron Abramovici and Paul Bradley. Integrated circuit security: new threats and solutions. In Frederick T. Sheldon, Greg Peterson, Axel W. Krings, Robert K. Abercrombie, and Ali Mili, editors, *CSIIRW*, page 55. ACM, 2009.

[2] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan Detection using IC Fingerprinting. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 296–310, Washington, DC, USA, 2007. IEEE Computer Society.

[3] M. Banga and M.S. Hsiao. ODETTE: A non-scan design-for-test methodology for Trojan detection in ICs. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 18–23, June 2011.

[4] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Zakaria Najm, and Xuan Thuy Ngo. Linear Complementary Dual Code Improvement to Strengthen Encoded Circuit Against Hardware Trojan Horses. In *2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015, McLean, VA, USA*, May 5-7 2015.

[5] Michael Bilzor, Ted Huffmire, Cynthia E. Irvine, and Timothy E. Levin. Evaluating security requirements in a general-purpose processor by combining assertion checkers with code coverage. In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pages 49–54, June 2012.

[6] Gedare Bloom, Bhagirath Narahari, and Rahul Simha. Os support for detecting trojan circuit attacks. In *Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, HST '09, pages 100–103, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Florian Eibensteiner, Rainer Findenig, and Markus Pfaff. Synpsl: Behavioral synthesis of psl assertions. In Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2009*, volume 5717 of *Lecture Notes in Computer Science*, pages 69–74. Springer Berlin Heidelberg, 2009.

[8] Ted Huffmire, Timothy E. Levin, Thuy D. Nguyen, Cynthia E. Irvine, Brett Brotherton, Gang Wang, Timothy Sherwood, and Ryan Kastner. Security primitives for reconfigurable hardware-based systems. *TRETS*, 3(2):10, 2010.

[9] Yier Jin, Nathan Kupp, and Yiorgos Makris. Experiences in Hardware Trojan Design and Implementation. In *Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, HOST '09, pages 50–57, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and implementing malicious hardware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 5:1–5:8, Berkeley, CA, USA, 2008. USENIX Association.

[11] Amir Pnueli and Aleksandr Zaks. PSL Model Checking and Run-time Verification via Testers. In *Proceedings of the 14th International Conference on Formal Methods*, FM'06, pages 573–586, Berlin, Heidelberg, 2006. Springer-Verlag.

[12] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *DATE*, pages 1069–1074. IEEE, 2008.

[13] Randy Torrance and Dick James. The State-of-the-Art in IC Reverse Engineering. In *CHES*, volume 5747 of *LNCS*, pages 363–381. Springer, September 6-9 2009. Lausanne, Switzerland.