POLITECNICO DI TORINO
Repository ISTITUZIONALE

On the detection of always-on hardware trojans supported by a pre-silicon verification methodology

Original
On the detection of always-on hardware trojans supported by a pre-silicon verification methodology / Ruospo, A.; Sanchez, E.. - ELETTRONICO. - (2019), pp. 25-30. (Intervento presentato al convegno 20th International Workshop on Microprocessor/SoC Test, Security and Verification, MTV 2019 tenutosi a usa nel 2019) [10.1109/MTV48867.2019.00013].

Availability:
This version is available at: 11583/2845978 since: 2020-09-17T12:19:54Z

Publisher:
Institute of Electrical and Electronics Engineers Inc.

Published
DOI:10.1109/MTV48867.2019.00013

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

28 May 2023

On the Detection of Always-on Hardware Trojans Supported by a Pre-Silicon Verification Methodology

Annachiara Ruospo, Ernesto Sanchez
Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy
{annachiara.ruospo, ernesto.sanchez}@polito.it

***Abstract*—Hardware-based vulnerabilities are becoming a serious threat in the Integrated Circuit (IC) industry. Current System-on-Chip (SoC) designs are comprised of many Intellectual Property (IP) blocks coming from third-party vendors. These can maliciously insert additional hardware, commonly known as Hardware Trojans, aiming at degrading performance, altering functionality or even leaking secret information. According to their activation mechanism, Hardware Trojans are classified as triggered or always-on. While the detection approaches for the first class are widely explored even during the early stages of the IC design flow, the detection of always-on type mainly relies on side channel analyses, carried out after fabrication. This work presents a methodology oriented to detect always-on Hardware Trojans during the pre-silicon design stage. The proposed approach is able to detect suspicious intrusions by exploiting a signature mechanism developed during the RTL verification phase. The activity of carefully selected signals is spied to record and keep the state of the core. Finally, the efficacy of the technique has been validated on an open-source IP core with three different always-on Trojans.**

***Index Terms*—Hardware Trojans, Pre-Silicon Verification, Security.**

I. Introduction

In the recent years, the security assessment of commercial computing platform has become a serious concern. With the growing complexity of modern System-on-Chips (SoCs) as well as the involvement of many entities is their development flow, it is hard to control the potential threat posed by hardware-based attacks. Hardware-Trojans (HTs) are malicious and intentional modifications of the device, whose aim is to alter the desired circuit behavior. Infected devices may experience changes in functionality or specification, may degrade their performance or even leak secret information [1]. Moreover, they can lower the device reliability by changing physical parameters.

In the research community, a great effort has been spent on classifying Hardware Trojans according to factors such as their insertion phase, the activation mechanisms, the location and the effects entailed by their intrusion. The authors in [2] [3] provide a broad description of HTs as well as a wide category of benchmarks. Depending on their activation mechanism, they are ranked as *triggered* or *always-on*. The former are activated under certain rare internal or external conditions, while the latter as soon as their host designs are powered-on. Both of them are silent and accurately hidden into the design to avoid all the pre and post-silicon verification, validation and testing mechanisms. However, their effect could seriously compromise the correct functionality of the target device and therefore it is of a paramount importance to develop accurate strategies for their detection.

Regarding the state-of-the-art Hardware Trojans detection techniques, much of research focuses on discovering the first class of Trojans: the triggered-type [4] [5] [6]. Most of these works try to trigger malicious logic by exploiting formal methods such as theorem proving and equivalence checking. Clearly, both the effort and the time required for implementing formal techniques grow as the complexity of the target device increases.
Concerning always-on Hardware Trojans, existing techniques mainly rely on side-channel analyses. A design hosting an always-on Trojan does not change its functionalities, producing apparently the correct outputs. Therefore, it is hard to detect them without observing side-channel parameters. Indeed, an infected design evidences a change of physical characteristics, in the form of power, delay or current. Techniques addressing this class of Trojans, [7] - [8] - [9], usually depend on a Trojan free golden reference model that is not always available. The second main drawback is that side-channel analyses are carried out after fabrication. A pre-silicon verification and validation methodology is suggested in [10], but from a formal perspective. They demonstrate the use of theorem proving methods for providing high-level protection of IP cores as well as the use of symbolic algebra in equivalence checking.

The main intent of this work is to provide a methodology to detect always-on Hardware Trojans in the early stages of the design development process of pipelined processor cores, i.e., the pre-silicon phase, without recurring to formal techniques. The methodology relies on a fingerprint mechanism implemented during the verification phase to secure either the program execution and the processor behaviour. Moreover, it provides a careful analysis on the most important processor signals involved in the process. Experimental results show that, the proposed approach is suited for detecting performance-degrading Trojans before the fabrication phase.

The rest of the paper is organized as follows: Section II provides a brief description of similar work. Section III first gives an overview of the typical SoC development flow and

then describes the considered threat model. The proposed approach is presented in Section IV. Section V details the case study and shows the experimental results. The overall methodology is completely delineated and validated in this section through three always-on Hardware Trojans. Finally, Section VI concludes the paper with final remarks and future works.

II. Related Work

In [11], a behaviour-based protection scheme is proposed to secure program execution by introducing a validation mechanism over the control flow of the instructions. The mechanism aims at extracting the normal program's behavior by means of a signature process in order to check at run-time the execution paths. The paper proposed an enhancement of the Branch Target Buffer (BTB) entries to secure the program execution. In [12], the authors describe an encryption key mechanism based on measurable properties, named ICmetrics, of a given hardware device. Such characteristics may take the form of internal signal distributions or values derived from the highly changing signals. However, in this paper, the security is faced in one direction, i.e., to incorporate features that are independent and highly multi-modal.
Finally, concerning the detection timing, a structured approach to perform security validation at each stage of the product lifecycle is proposed in [13] [14]. The security vulnerabilities related to a stage are identified and fixed before moving to the next one. This approach is based on formal techniques for verifying hardware security during all the development phase of the device, hereinafter named Security Development Lifecycle (SDL). However, it is well known that the main drawback when dealing with formal approaches is their increasingly complexity.

III. Threat Model

When analysing Hardware Trojans and developing new techniques for their detection, it is necessary to first analyse the attack model and study all the possible scenarios. A clear view on how and when potential attackers could implant malicious hardware in a SoC design is essential to develop a trustworthy and strong methodology. Before outlining our threat model, a brief overview of the SoC development flow is provided. In [1], the authors identify three main phases in the flow:

- *IPs Development:* During the design specification phase, to fulfill the System-on-Chip requirements, a list of IPs is established. In order to comply with the compelling time-to-market and lower research and development (R&D) costs, some of them are built in-house, others bought from third-party IP vendors (3PIPs). In this second case, they could be provided in three forms: *soft* if only the HDL description at RTL is provided; *firm* whether IPs are delivered in a gate-level implementation; *hard* in case IPs are delivered as Graphic Database System (GDSII) representation of a fully placed and routed design.
- *SoC Integration:* The first assignment of a SoC Integrator is to integrate all the soft IPs and to generate the RTL specification of the whole SoC. It carefully traces all the necessary steps coming from the whole RTL System-on-Chip design up to the final SoC layout in GDSII format. First of all, to find design bugs, the RTL is exhaustively verified and tested. Once this step is completed, the SoC integrator synthesizes the RTL description on a target technology library by exploiting external EDA tools. Alternatively, she/he can buy firm IPs and directly integrate them into the netlist. The next step is the integration of Design-For-Testability (DFT), Debug and Built-In Self-Test (BIST) structures; they are typically entrusted to third-party specialized vendors. Finally, the gate netlist is translated in a physical layout in a GDSII format, which is sent to the foundry for the IC fabrication.
- *Foundry:* The fabrication process is the more expensive step of the SoCs development flow, therefore, their fabrication is usually granted to external foundries.

As underlined in [1], the reason behind the outsourcing lies in the extremely high costs for maintaining such a long supply chain from design specification to packaging. Clearly, the adoption of third-party IPs, EDA tools [15] or foundries poses serious security concerns. According to the threat model, each of these three phases could be considered either *trusted* or *untrusted*. A brief summary is shown in Figure 1.

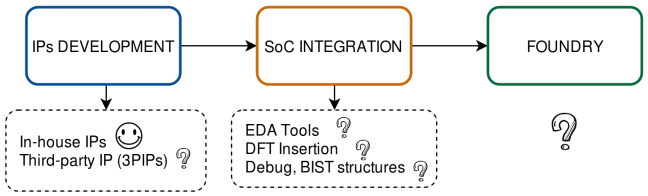

Fig. 1. Modern System-on-Chip supply chain.

A comprehensive analysis on a wide range of possible attack models for potential hardware Trojans is presented in Table I. In these seven possible models, one or more of the above-mentioned entities (IPs vendors, SoC Integrator, Foundry) are considered untrusted.

The proposed strategy applies to the G model of the Table I, where the attacker can be an untrusted SoC Integrator, untrusted EDA tool vendor, or untrusted foundy. However, aiming at detecting Trojans in the Pre-Silicon phase, the foundry as an untrusted entity is out of the scope of this work. For a better understanding, this attack model refers to companies who have designed their own proprietary IPs but need to rely on an untrusted design house, external skilled engineers and foundry to manufacture their final ICs [3].

In the considered threat model, the IP design team is responsible for defending against attacks while the attacker is hiding in the third-party entities that come into play during the Pre-Silicon phase, i.e., before the fabrication. A common practice is that IP designers turns to third-party skilled engineers for the Design For Testability (DFT) insertion. When the IP is provided for the DFT insertion to external entities, it is free from Hardware Trojans. However, when it returns from the

TABLE I
COMPREHENSIVE ATTACK MODELS [1]

Model	Description	3PIP Vendor	SoC Developer	Foundry
A	Untrusted 3PIP vendor	Untrusted	Trusted	Trusted
B	Untrusted foundry	Trusted	Trusted	Untrusted
C	Untrusted EDA tool or rogue employee	Trusted	Untrusted	Trusted
D	Commercial off-the-shelf component	Untrusted	Untrusted	Untrusted
E	Untrusted design house	Untrusted	Untrusted	Trusted
F	Fabless SoC design house	Untrusted	Trusted	Untrusted
G	**Untrusted SoC developer with trusted IPs**	**Trusted**	**Untrusted**	**Untrusted**

DFT process, it is necessary to establish whether someone has maliciously introduced additional hardware.

IV. PROPOSED APPROACH

The proposed technique focuses on the detection of *always-on* Hardware Trojans (HTs) maliciously inserted into the RTL design, i.e., those starting as soon as their host designs are powered on [3]. The methodology has been developed to defeat the integrity of an Intellectual Property (IP) based Central Processing Unit (CPU). It is assumed that, to escape detection during the pre and post-silicon validation phase, these always-on Hardware Trojans are silent and do not modify the output of the computations. Changes mostly come in the form of power, current or delay and, many times, the difference due to the presence of HTs could be negligible and therefore undetectable. The proposed methodology differentiates from the side-channel analyses for always-on Trojan detection since it applies in the pre-silicon phases of the supply chain.

The idea is to conceptually *take a picture* of the CPU before supplying it to the third-party vendors of the SoC development chain, i.e., the untrusted entities of the process. This *picture* holds essential information of the HT-free CPU performances. Basically, the activity of mindfully selected signals is registered during the execution of a set of *secure programs*. These signals are selected and picked up from the CPU Data-path and from the CPU Control Unit. Depending on the architecture of the processor, they are carefully selected among those carrying on more advantageous information. Section V provides a detailed analysis of this choice. A secure program is an ordinary verification program featuring a high code coverage. For sake of robustness, more than one program goes through this process. These programs have not other special requirements, except that they differ as much as possible, targeting different units.

Once the target programs are selected, the procedure is straightforward and consists on the following. A secure program is executed in simulation as a normal verification program. Meanwhile, the activity of the previously selected signals is recorded by means of a *secure monitor* lying in the Testbench. The goal of this module is to write at each clock cycles the signals' values in a text file.

At the end of the simulation, this file undergoes a *fingerprint process* to compute a message digest for that *secure program*. Clearly, these steps are repeated for each and every *secure program*. For a better understanding, the fingerprint process consists on a message digest algorithm which, starting from a file, produces a fixed-length signature for that file. If the file changes, then the message digest changes [16]. A digest message algorithm relies on Cryptographic Hash Functions to map bitstrings of arbitrary finite length into strings of fixed length [17]. These are widely adopted in applications such as digital fingerprinting of messages, message authentication, and key derivation to guarantee the integrity of the data. The spread of such technique is justified if considering the several advantages of a Cryptographic Hash Function:

- The process is deterministic, i.e, the same message is always translated in the same hash value.
- It is a *one-way* function: impossible to invert. Only a brute-force approach could succeed in reversing the message and deriving the unencrypted data.
- The hash value is fast and simple to obtain starting from any kind of input data.

Among all the possible cryptographic algorithms, SHA-256 has been selected. It is noteworthy that SHA-256 is widely used in security for the property of being collision resistant: indeed, if compared to other two existing cryptographic algorithms such as MD5 and SHA1, it has not been broken yet. Moreover, it is very fast to compute and the length of the message digest is acceptable for the target system.

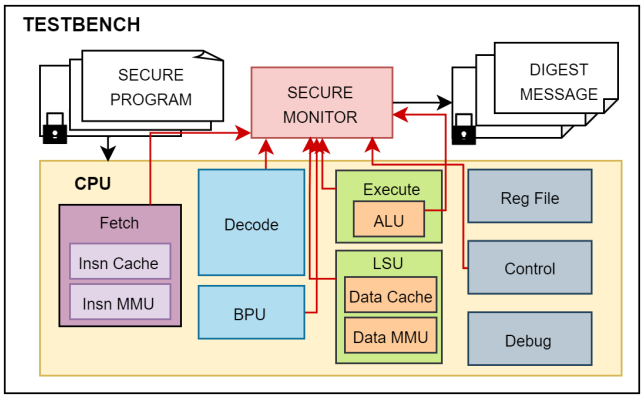

Fig. 2. The proposed pre-silicon verification detection technique.

In case of malicious intrusions, the proposed approach, highlighted in Figure 2, is able to recognize the presence of always-on Hardware Trojans. So what happens is that, an IP designer (the trusted entity of our threat model) gets back

the design from external teams, e.g., DFT engineers, then, he executes the *secure programs* and checks the corresponding digest messages. If the values are different, there is an high risk of Trojans insertion. It is necessary to underline that DFT elements could be introduced either at RTL or Gate Level, i.e., respectively JTAG blocks or Scan Chains. Unless signals name changes, the behaviour of the IP must be the same. For sake of clarity, depending whether the IP team provides an RTL description or a gate level one, the HT-free digest message is computed on the RTL or gate respectively.

V. Case Study and Experimental Results

To demonstrate the effectiveness of the proposed approach, an open-source RISC processor core has been selected. It is an OpenRISC 1000 compliant processor IP core, named *mor1kx*. It is a 32/64-bit load and store RISC architecture. Written in Verilog Hardware Description Language (HDL), it provides an elevated level of flexibility in terms of implementations trade-offs such as area and performance. Being highly configurable, it allows you to customize the core to your exact needs. Thus, the IP core can be configured in three available implementations: Cappuccino, Espresso, Pronto Espresso. In the considered experiments, the IP core is configured with the Cappuccino setup. Indeed, the CPU handles a 6 stage pipeline with a delay slot on jump and branch instructions. Caches and MMUs are optional but powered-on in our experiments. The *secure programs* are 10 different high-coverage assembly programs targeting distinct hardware units. For the attacker point of view, the choice of the programs must be unpredictable. Finally, the *secure monitor* is a SystemVerilog task inside the Testbench. Its role is to dump, at each clock cycle, selected signals' values on a file.

A. Always-on Hardware Trojans

To validate the methodology, three always-on Hardware Trojans have been designed, albeit is a common practice to use existing benchmarks [3] - [2] avoiding in-house Trojans. The reason behind this choice lies in the fact that only eleven always-on Trojans are available on Trust-Hub Trojan Benchmarks [2] and none of these is applied to a pipelined processor cores. From this arises the need of conceive and apply custom Benchmarks. In particular, the devised Hardware Trojans may allow a possible attacker to obtain information from the normal operation process thanks to the unexpected behavior of the processor core. In the following, the three HTs are described, highlighting also the introduced weakness.

Hwtr1: The considered Trojan is a silent code modification lying in the Branch Prediction Unit (BPU). In the target design, the BPU is a Static Branch Predictor, therefore, all backward branches are treated as taken while all forward as not taken. A single line of the Static BPU has been maliciously modified so that all the branches are predicted as not taken. This incorrect behaviour clearly does not alter the functionality of the processor itself while degrading performances; additionally, the erroneous access to memory instructions that should not be fetched modifies the bus behavior providing extra information to the external world. Without resorting to a side-channel analysis based on a timing approach, this behaviour would never be discovered before the fabrication.

Hwtr2: This second Trojan does not sit on a special unit. It origins from the decoding phase and changes a condition flag wiring the execute phase with the control unit. Specifically, each time a nop operation is decoded, the Overflow Flag is unset. As in the previous case, the functional behavior of the processor is not altered, i.e., the output of the computation is correct while provoking issues in the control unit as well as falsifying the content of the Special Purpose Registers. In this particular case, a branch could be wrongly taken thanks to this HT making to execute a wrong piece of code.

Hwtr3: The Trojan is hidden inside the Data Cache. It is implemented through a simple counter which invalidates the content of the cache every 100 clock cycles. It is a silent hardware modification whose aim is to slow down the execution time of a program and downgrade the related performances. As in the previous cases, this HT makes the system bus to transfer information that may not be there, allowing some external attacker to obtain additional information about sensible programs.

B. Signals selection

The proposed technique relies on the choice of specific signals to compute a fingerprint message. Which signals to select is the spotlight of this subsection. For this purpose, seven different experiments have been performed to fulfill the goal with the least number of signals; Table II presents a detailed description.

As one of the purposes of the pre-silicon verification is to detect Trojans, one could think that the amount of dumped signals is not a limiting factor. This is partially true. As the complexity of modern processor core increases, checking and dumping all the signals becomes costly and impractical, requiring a considerable effort even from the verification point of view. Moreover, supposing that the proposed technique is to be extended to an on-line Trojan detection mechanism by moving the fingerprint computation on the physical device, it becomes even more critical issue. Dumping and elaborating all the signals to produce a digest message with a hardware module could be unfeasible. The goal of this work is two-fold: not only proposes a strategy for Hardware Trojans detection, but also suggests an analysis of those signals that aid that detection process. This is valuable when considering possible post-silicon hardware strategies. Therefore, all the experiments have been performed by reducing the amount of signals step by step.

The *secure programs*, i.e., the Verification Test Suite, are represented in Table III and constitute the reference model for those in charge of defending the designs from external attackers. To validate the approach, a golden execution of the entire verification suite was implemented in order to determine the programs' digest messages when no Trojans are maliciously introduced. When the third-party vendors return the design to the trusted entity (e.g., the DFT engineers to

TABLE II
CASE STUDY

	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Proposed Approach
Units Involved	27	10	1	10	10	10
Number of Signals	1831	612	54	30	15	8
Selection Criterion	2-Level Hierarchy	1-Level Hierarchy	CPU Top	Random	Random	Custom
hwtr1	Detected	Detected	Detected	Undetected	Undetected	Detected
hwtr2	Detected	Detected	Detected	Detected	Undetected	Detected
hwtr3	Detected	Detected	Detected	Detected	Undetected	Detected

the CPU developer in our threat model), the latter only needs to rerun the *secure programs* and check the integrity of their digest messages. If even one of these differs, the target design is in all likelihood affected by an external intrusion.

TABLE III
VERIFICATION TEST SUITE

Verification Programs (VP)	**[%] Code Coverage**	**Golden Digest Message**
VP1	81.78	6437e6a...b38af63
VP2	85.20	4534343...acc4545
VP3	83.00	874c345...425ab56
VP4	82.89	a67a665...4323423
VP5	78.99	21243bb...c765546
VP6	85.02	7564563...5634634
VP7	86.01	b676456...5635645
VP8	80.54	6534253...2354543
VP9	83.22	2116576...ab44512
VP10	84.98	ac41244...4235457

A first experiment was implemented by selecting all the input-output signals of the modules, up to the second level of the hierarchy (Scenario A - 27 blocks). Due to the significant number of observed signals, i.e., 1831, as expected, all the Trojans have been detected. Indeed, a difference was observed between the digest messages produced in the HT-free CPU and in the one infected by the Trojan. Consequently, instead of recursively selecting all the modules up to the second level, only the first in the hierarchy was chosen. In this second experiment (Scenario B), from 1831 only 612 signals have been dumped. Also in this case, a discrepancy in the digest messages was observed leading to a complete detection.

The same reasoning applies to the third Scenario (C) where only the input-output signals of the CPU top entity are considered. All the Trojans are discovered. Then, to reduce additionally the signal number, a random approach was attempted. The last two experiments (Scenario D and Scenario E) were carried out with a different selection criteria; instead of recording signals belonging to the target unit, a subset is randomly selected among all the units of the design, i.e., the 10 in the first hierarchy level. In the first Scenario, only two Hardware Trojans were detected with 30 random signals. Finally, by halving randomly the number of signals (Scenario E), all Trojans remained undetected.

The last column of the Table II represents the proposed methodology. It differs from the random approach, focusing on a custom and rational signals selection. This process aims at emphasizing the intrinsic power of target signals in detecting unexpected alteration of the processors normal behavior. As suggested in [12], the features of a hardware device are extracted either from the highly changing signals or those acting as a backbone of the system.

The proposed approach relies on 8 signals, hereinafter named *spy-signals*, carefully selected for their interesting properties. Their advantages are listed below:

1) *Processor Program Counter (PC)*: Since the instructions are unequivocally described by their Program Counters, they offer a convenient way of recording the program context. In [18], the authors present a system of program identification with the use of the Program Counter (PC) as an Integrated Circuit (IC) metric capable of uniquely identifying the systems behaviour. Specifically, they interpret the raw PC values in a form that can uniquely recognize any particular program or application for security purposes. The PC represents the real cornerstone of a processor core. In a slightly different context, the PC is utilized to implement a predictor for dynamic power management [19]. At each clock cycle it gives information regarding the exact execution of the program. Therefore, it represents the best selection choice.
2) *Output of the EX Stage*: The Execute stage (EX) represents the heart of the Datapath since it encloses all the processor computation. Examples of these are:
   - Effective Address computation for load and store;
   - Effective Address computation for branches.
   - Arithmetic Operations (additions, subtractions, multiplications and divisions);
   - Logic Operations (shifts, rotates);

   Commonly, between these different execution units and the pipeline registers, there is a multiplexer that collects the results. The output of this multiplexer, i.e., *The Output of EX stage*, is normally a 32-bit signal (or 64 in 64-bit architectures). Additionally, this signal carries all the execution flow of the program, providing useful details. Therefore, it is a suitable candidate for being added to the list of target *spy-signals*.
3) *Clk*: The clock signal synchronizes the activities of the entire processor core. It precisely marks the time of the programs execution. For this reason it is appropriate for detecting performance-degrading Trojans.
4) *Supervision Register*: The Supervison Register (SR) is a special-purpose supervisor-level register accessible with special instructions in supervisor mode only. By defining the status of the processor, it brings important informa-

tion. The ones more relevant for the purpose of this work are:

- SR[IEE]: Interrupt Exception Enabled
- SR[DCE]: Data Cache Enabled
- SR[ICE]: Instruction Cache Enabled
- SR[DME]: Data Memory Management Unit (MMU) Enable
- SR[IME]: Instruction MMU Enable
- SR[F]: Conditional branch flag
- SR[CY]: Carry flag
- SR[OV]: Overflow flag

5) *Instruction Register:* It tracks all the instructions entering the pipeline. Lying in between the Fetch and the Decode stage, it prevents attackers to insert additional instructions for being decoded but not executed.
6) *Cache Hit-Miss Signal:* This signal controls the behaviour of this speculative unit. An attacker could force a cache miss to load a given sequence of dangerous instructions. They could enter the pipeline as a consequence of a branch misprediction, with the intent of changing the processor functionalities, degrading performances or leaking secret information.
7) *BPU Hit-Miss Signal:* The same reasoning of the previous point applies to Branch Prediction Unit (BPU). The miss or hit signal helps tracking all the processor branches and their respective prediction.
8) *Register File Input Signal:* In a pipelined processor core, the last pipeline stage usually consists of writing into the register file to update it with the content of a destination register. Spying this signal means checking that all the instructions have passed correctly through all the stages of the pipeline.

VI. Conclusion

In recent years, a growing attention has been given to potential threat raised by hardware-based attacks. The risk of Hardware Trojans intrusion increases as the modern System-on-Chips complexity increases. Moreover, due to the high costs of in-house Intellectual Property (IP) block development and fabrication, SoCs commonly make use of IP blocks gathered from third-party vendors. Therefore, as a direct consequence, even more entities have become more involved in all phases of the SoC supply chain, posing serious security concerns. In this paper, a detection technique for always-on Hardware Trojans is presented. The goal of the methodology is to discover malicious hardware intrusions by exploiting a verification-based technique relying on a fingerprint process. To keep track of the processor behavior, security checks are performed already from the Pre-Silicon stages, each time a trusted design is provided to external entities of the SoC supply chain. The fingerprint process counts on a careful selection of signals, those carrying more useful information. The experimental results demonstrate the validity of the proposed approach, albeit exploiting a limited amount of in-house Hardware Trojans. Moreover, the approach is easy to integrate in any SoC development flow since it is only needed to extend the Testbench module with the *Security Module*.
Future work aims at extending the methodology to triggered-type HTs too. Moreover, new benchmarks will be conceived and validated.

References

[1] K. Xiao *et al.*, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, pp. 1–23, 05 2016.

[2] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 471–474.

[3] B. Shakya *et al.*, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, Mar 2017. [Online]. Available: https://doi.org/10.1007/s41635-017-0001-6

[4] A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, "Scalable hardware trojan activation by interleaving concrete simulation and symbolic execution," in *2018 IEEE International Test Conference (ITC)*, Oct 2018, pp. 1–10.

[5] S. Yao *et al.*, "Fastrust: Feature analysis for third-party ip trust verification," in *2015 IEEE International Test Conference (ITC)*, Oct 2015, pp. 1–10.

[6] Y. Wang, T. Han, X. Han, and P. Liu, "Ensemble-learning-based hardware trojans detection method by detecting the trigger nets," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[7] S. Narasimhan *et al.*, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2183–2195, Nov 2013.

[8] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[9] D. Ismari, J. Plusquellic, C. Lamech, S. Bhunia, and F. Saqib, "On detecting delay anomalies introduced by hardware trojans," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–7.

[10] Xiaolong Guo, R. G. Dutta, Yier Jin, F. Farahmandi, and P. Mishra, "Pre-silicon security verification and validation: A formal perspective," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[11] Y. Shi and G. Lee, "Augmenting branch predictor to secure program execution," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 10–19.

[12] E. Papoutsis, G. Howells, A. Hopkins, and K. McDonald-Maier, "Integrating multi-modal circuit features within an efficient encryption system," in *Third International Symposium on Information Assurance and Security*, Aug 2007, pp. 83–88.

[13] N. Potlapally, "Hardware security in practice: Challenges and opportunities," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, June 2011, pp. 93–98.

[14] H. Khattri, N. K. V. Mangipudi, and S. Mandujano, "Hsdl: A security development lifecycle for hardware technologies," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, June 2012, pp. 116–121.

[15] J. A. Roy, F. Koushanfar, and I. L. Markov, "Extended abstract: Circuit cad tools as a security threat," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, June 2008, pp. 65–66.

[16] R. Mohanty, N. Sarangi, and S. Bishi, "A secured cryptographic hashing algorithm," 03 2010.

[17] A. H. M. Ragab, N. A. Ismail, and O. Allah, "An efficient message digest algorithm (md) for data security," *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology. TENCON 2001 (Cat. No.01CH37239)*, vol. 1, pp. 191–197 vol.1, 2001.

[18] K. Appiah *et al.*, "Program counter as an integrated circuit metrics for secured program identification," in *2013 Fourth International Conference on Emerging Security Technologies*, Sep. 2013, pp. 98–101.

[19] C. Gniady, A. R. Butt, Y. C. Hu, and Yung-Hsiang Lu, "Program counter-based prediction techniques for dynamic power management," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 641–658, June 2006.