Using Static Hardware Wrappers to Thwart Hardware Trojans and Code Bugs at Runtime

Juan Portillo
Department of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, TX 78249
Email: juan.portillo@my.utsa.edu

Eugene John
Department of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, TX 78249
Email: eugene.john@utsa.edu

***Abstract*—In the modern information age, security is a basic requirement. In this paper, we address security vulnerabilities that can occur at the lowest levels of computing due to code bugs and hardware Trojans. Some of these vulnerabilities can be detected using modern verification tools. However, more effort is needed to address vulnerabilities which occur at run-time. To this end, we propose a static, stand-alone hardware wrapper that is independent of the attack mechanism. The wrapper is integrated at the register transfer level (RTL) during the pre-silicon stage of integrated circuit (IC) design and provides security protections during use by the end-consumer. The proposed hardware wrapper will not remove or detect these vulnerabilities, but, it can enforce correct behavior by assuming control of external signals during run-time.**

Keywords - wrapper, hardware security, hardware Trojan, formal verification, runtime

I. Introduction

The modern electronics industry faces increased challenges due to low level security attacks on hardware. The attack could occur during runtime by the end user, or, during design and test stages by a rouge insider or untrusted vendor. A compromised design could be the scope of an entire system, a third-party intellectual property core, or simply a component within a logic module. Adversaries who tamper with a design aim to a) steal information (data confidentiality) b) alter device behavior (change of functionality) or c) alter device data (data integrity). These attacks can be performed by exploiting a code bug (henceforth referred to as bug) or activating a stealthily designed hardware Trojan (henceforth referred to as Trojan). In [1] Ordonez et al. demonstrated a real-world scenario where a circuit's behavior could be modified by using fault attacks. A fault attack is one such example which may activate a hardware Trojan. In some cases, these vulnerabilities can be removed using digital circuit verification techniques, however, some vulnerabilities can persist in-the-field. Therefore, a solution is needed which addresses circuit vulnerabilities at run-time. This paper focuses on the aforementioned RTL vulnerabilities in digital circuit designs.

Pre-silicon and post-silicon verification can detect vulnerabilities for removal by the designer at the design stage but does not address run time vulnerabilities. This has motivated research on hardware wrappers by several research groups. Containers have been used to detect a specific type of modification in the design [2]. Wrappers are also used to enforce access control policies by tagging data in Ref. [3]. Basak et al. and Wang et al. implemented security policy checking using a micro-controller [4], [5]. Shila et al. proposed a secure communication framework and hardware wrapper design for FPGA implementations [3]. These solutions monitor and identify invalid circuit behavior and have been used to enforce specific policies. In [19], Hicks et al. extend the use of hardware wrappers to assist software-based security techniques.

In this paper, we propose hardware wrappers that can be integrated at the RTL level during the pre-silicon stage of an IC's design in order to thwart security vulnerabilities after fabrication and during runtime. We demonstrate that a static wrapper can guarantee an IP module's security since it ensures valid behavior by taking over control of a minimal number of input/output (I/O) signals needed to enforce the security policy of interest.

As a proof of concept, we use a simple intellectual property (IP) module within a larger system-on-chip (SoC) design. The IP module contains an ID register which is protected by a self-locking Lock Bit. The security properties of interest ensure that the ID register does not change after the Lock Bit has been set, and that the Lock Bit itself cannot be de-asserted once it has been set.

The hardware wrapper proposed in this research only relies on external I/O of the security critical IP module. A specific internal wrapper logic can be used to enforce security policies of interest. We show, using exhaustive formal checking, that the wrapper guarantees valid behavior of the IP module under all conditions determined by those specific I/O interfaces. Only logic needed to take control of the critical component's I/O is required for wrapper logic design. This reduces the complexity of the security policies needed for testing the wrapper and minimizes the hardware overheads needed for designing the wrapper.

The rest of the paper is organized as follows: we discuss previous and related work in Section II. Section III describes our wrapper design methodology and how it was tested against different vulnerabilities. The results are presented in Section IV, which reveals quantitative measures of how effectively our

978-1-5386-7392-8/18/$31.00 ©2018 IEEE

wrapper concept works, and Section V concludes the paper.

II. Background

Digital hardware can contain bugs or Trojans which compromise the security of a system. The scope of our research considers bugs and Trojans which are introduced during the pre-silicon design and integration stages of the chip design life-cycle. These vulnerabilities can be activated or exploited during run-time [6]. Bugs can be exploited by a malevolent user at the end-user stage of the chip's life cycle [7]. They only pose a security threat if they are discovered by, or, revealed to, an attacker. The attack might be limited in scope since the flaw is unintentional. However, even a small flaw could result in a catastrophic security vulnerability [6]. Trojans can also be exploited at the end-user stage, but unlike bugs, they must be activated (Trojan trigger) by a specific input. The effect of this vulnerability (Trojan payload) can be determined at design time and cause a range of security threats [8] [9].

Pre-silicon and post-silicon verification can detect vulnerabilities for removal by the designer before delivery to the end-user [6]. Several authors have investigated this approach. With regard to simulation techniques, finding bugs and Trojans is challenging since these vulnerabilities tend to lie in the corner cases of test patterns [10].

In [11] and [12], the authors use formal verification to detect Trojans by finding their trigger activation mechanisms. In [13] formal proofs are used to detect invalid modification to circuit registers. In [14] and [15], formal proofs are used to detect invalid modification to circuit paths. In [16], formal proofs are used to detect key leaks due to an encoded Trojan payload. In [17], the authors demonstrate that it is possible to use several proof design methodologies to detect the same Trojan design, namely, corruption of components or paths. However, formal tools continue to have scalability issues due to the state space explosion for formal properties [10]. In [18], verifying large circuits is addressed by breaking down the design into finely grained security properties.

III. Methods

We begin with a circuit model without vulnerabilities and verify correctness (simulation and formal verification). Then, we insert bug and Trojan vulnerabilities and verify (simulation only) that the circuit has been corrupted. After bug and Trojan insertion, we attempt to detect bug and Trojan security vulnerabilities to the fullest extent that our simulation and formal verification tools and techniques will allow. We then design a hardware wrapper for thwarting the bug and Trojan vulnerabilities and verify (simulation and formal verification) to determine if the circuit's security properties have been enforced.

A. The Circuit Model

We assume a hypothetical scenario where our IP module is within a System-on-Chip (SoC) containing trusted and untrusted modules. Figure 1 shows the hypothetical SoC circuit model. Only trusted components (Crypto Module) are allowed

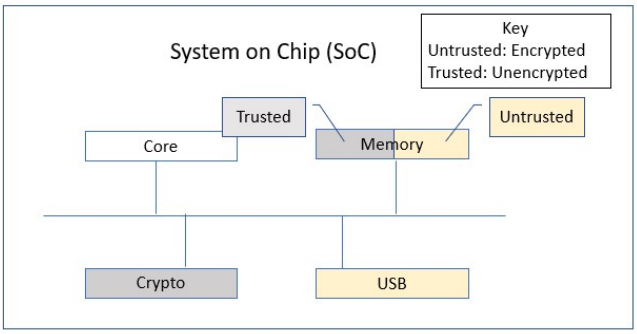

Fig. 1. A hypothetical SoC circuit model with an untrusted IP module

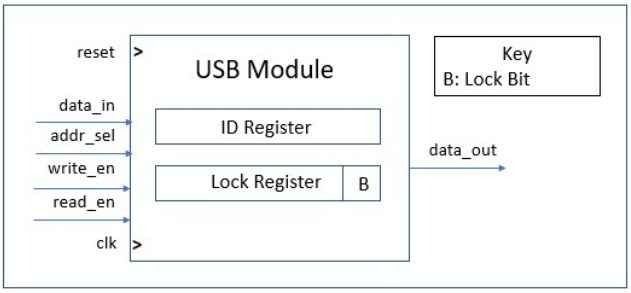

Fig. 2. Circuit model under test with ID Register with Self-Locking Lock Bit

to access unencrypted data (Trusted Memory), while untrusted components (USB Module) must only access encrypted data (Untrusted Memory). Our IP module is a USB IP which is untrusted. Access control policies are enforced by an ID register within each module. The USB Module contains an ID register with a self-locking Lock Bit. Figure 2 shows the untrusted IP module with ID register and self-locking Lock Bit. Only the ID security register and Lock Bit of this system was designed and verified for correct functionality for this research as proof of concept. The untrusted IP component must adhere to the following security policies:

1) Security Properties: "The ID register (A) must not change if the Lock Bit (B) has been set"

$$\forall t, \quad A \neq A', \quad iff \quad B = 1 \tag{1}$$

"The Lock Bit (B) must not change once it has been set"

$$if B = 1, Henceforth \quad B \neq B' \tag{2}$$

2) A circuit with code bug: A bug was deliberately designed into the model. Invalid circuit behavior is defined as a self-locking lock bit which does not remain locked after being set. We use standard verification methods including simulation (functional) and formal verification (formal proofs) to detect the presence of the bug. For simulation, we used Modelsim Functional Simulator and for formal verification we used Cadence Jasper Gold Formal Security Path Verification (SPV).

3) A circuit with Trojan: A Red/Blue teaming exercise was conducted to provide a diverse sample of Trojans in an unbiased manner. The Trojan trigger designs from the Red/Blue team exercise are: a simple combinational trigger using RTL-level combinational logic, a complex combinational trigger using several layers of combinational logic, a synchronous trigger design using clock-dependent logic such as counters, asynchronous Trojan trigger designs independent of the clock (such as latches), a trigger which induces a race condition using a clock faster than the system clock, a trigger which induces a timing glitch to emulate a reset signal, a Linear Feedback Shift Register based trigger which acts as a Pseudo Random Number Generator, a 'always-on' trigger which uses a HDL switch statement to turn on malicious gates only during synthesis, and 'abnormal logic' which uses mixed edge and level logic. All Trojan payloads were designed to either A) violate Eq. 1 or B) violate Eq. 2, thereby violating Eq. 1.

B. Design and Test of a Hardware Security Wrapper

A diverse set of Trojans where designed into the circuit model. The corrupt circuit model without a hardware wrapper was tested against the security properties using standard simulation and formal verification. Next, a wrapper was built around the corrupt circuit model and was again tested against the security properties.

1) Properties for Security: Standard verification methods (functional and formal) were used to detect invalid behavior due to malicious Trojans. Modelsim functional simulation tested for the presence of a Trojan circuit by verifying correct functionality. If Eq. 1 and 2 are observed to be violated then there exists a bug or Trojan. Taint properties (formal proofs) given by Eq. 3 - 5 in Jasper Gold SPV were used to detect invalid behavior of the ID Register. Notation for taint property equations are as follows: t is time, T is the initial cycles for the pre-condition, I is the primary N-bit input, O is the primary N-bit output, a is a one-bit address select signal which chooses between the ID register ($a = 0$) and Lock Bit register ($a = 1$), and w is a one-bit write enable signal.

$$(Pre-condition)\ if\ t \leq T:\ I = 1,\ a = 1,\ w = 1 \quad (3)$$

$$if\ t > T,\ and,\ a = 0,\ then,\ O \neq I \quad (4)$$

$$if\ t > T,\ and,\ a = 1,\ then,\ O \neq I \quad (5)$$

2) Wrapper Design: In this research, a hardware wrapper design was integrated into the circuit model to thwart a bug exploit or Trojan attack during run-time (Fig. 3). The wrapper design selects the address of the Lock Bit ($address\_sel$) and overrides control of the write enable ($write\_en$) with the signal, write enable prime ($write\_en'$). Once the Lock Bit has been set, the wrapper logic will deassert $write\_en'$ from this point on. Henceforth, $write\_en'$ cannot be asserted and the ID registers cannot be modified. This allows Eq. 1 and Eq. 2 to be satisfied.

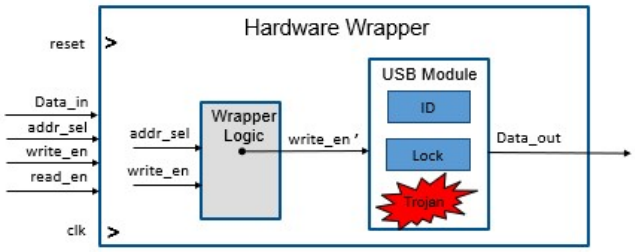

Fig. 3. Security Wrapper with IP and Trojan

TABLE I
TEST RESULTS OF A CIRCUIT WITH TROJAN INSERTION, WITH AND WITHOUT, A HARDWARE WRAPPER

		Without Security Wrapper		Using Security Wrapper	
	Trojan Circuit	**Sim**	**Formal Proof**	**Sim**	**Formal Proof**
1	simple combinational	-	x	+	+
2	complex combinational	-	x	+	+
3	synchronous	-	x	+	+
4	asynchronous	-	x	+	+
5	race condition and delay	-	-	+	+
6	linear feedback shift register	-	-	+	+
7	always on	-	***	+	***
8	abnormal logic	-	dnc	+	dnc
-	Not Detected				
x	Detected				
+	Thwarted				
***	gate level trojan				
dnc	did not compile				

3) Wrapper Verification: The corrupt circuit with hardware wrapper was simulated (functional verification) to detect the presence of invalid behavior due to a bug or Trojan. If Eq. 1 and 2 were not violated during simulation, then we concluded that the wrapper is effective even in the presence of a bug or Trojan vulnerability. The corrupt circuit with hardware wrapper was formally verified (logical proofs) to detect the presence of invalid behavior due to a bug or Trojan. If Eq. 3 - 5 were not violated, then we concluded that neither the bug nor Trojan did not corrupt circuit behavior. The bug free and Trojan free circuit design was also tested to determine that the wrapper did not affect normal behavior.

IV. RESULTS

To start, simulation and formal verification successfully detected the presence of all bugs. Our hardware wrapper solution is then compared against pre-silicon verification techniques for comparison (Table I). In this research, the wrapper successfully thwarts a bug as effectively as simulation and formal verification techniques. In practice, a wrapper can complement pre-silicon verification.

All Trojan payloads used in this research modify the ID Register of the IP module. The Trojan trigger varies across the different Trojan designs from the Red/Blue teaming exercise. There were a total of 8 different Trojan design categories, each with one or more Trojan design. Of the eight categories, six were successfully thwarted by the wrapper. Trojans which were not thwarted by the hardware wrapper used 'always on' and 'abnormal logic'. Although our verification tools did not exercise the synthesis based 'always on' Trojan, in theory, our wrapper thwarts any RTL-based vulnerability that might be activated using this mechanism. Trojan designs using 'abnormal logic' resulted in compile errors during formal testing, and therefore, results for this specific Trojan design were inconclusive. For Trojan designs where data was inconclusive, other counter-measures can be taken, such as, equivalence testing (always on), or, observation of the formal verification tool's behavior (abnormal logic). The same hardware wrapper design which thwarts bugs also successfully thwarts most of the Trojans from the Red/Blue teaming exercise. This supports the concept of a static wrapper as an effective bug and Trojan counter-measure.

V. Conclusion

The problem of runtime security threats is a serious concern to the electronics industry. A viable solution is the integration of hardware wrappers during design time. The wrapper protects against RTL code design flaws (code bugs) and stealthy malicious RTL hardware circuits (hardware Trojans). The wrapper will not disrupt the normal function of the circuit under protection. More significantly, it can thwart a hardware Trojan payload without the need to detect or remove the malicious circuit design. A wrapper provides security where standard verification falls short. Unlike pre-silicon verification, a hardware wrapper will not eliminate or repair a corrupt design. Instead, the wrapper will take control of external signals so that valid values and signal timings are enforced to ensure the correct and secure behavior of the circuit. This will circumvent the vulnerability and force the invalid circuit to always behave correctly. Our results demonstrate that a wrapper is effective at thwarting RTL code bugs and RTL hardware Trojans even when simulation cannot detect them. We have demonstrated that the same hardware wrapper is effective at thwarting, simultaneously, a code bug and a hardware Trojan. The wrapper thwarted our hardware Trojans when formal verification results in false negatives with the exception of two Trojan designs. Finally, the wrapper can assure security even when the vulnerability (bug exploit or Trojan payload) is unknown. In addition, the wrapper can be fabricated at a trusted integrator to further enhance trust. The static, stand-alone hardware wrapper design proved to be an effective countermeasure to code bug exploits and hardware Trojans that are a runtime security threat to digital integrated circuits.

References

[1] F. E. Potestad-Ordóñez, C. J. Jiménez-Fernández and M. Valencia-Barrero, "Vulnerability Analysis of Trivium FPGA Implementations," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 12, pp. 3380-3389, Dec. 2017.

[2] R. Drechsler and U. Kühne, "Safe IP Integration Using Container Modules," 2014 Fifth International Symposium on Electronic System Design, Surathkal, 2014, pp. 1-4.

[3] D. M. Shila, V. Venugopalan and C. D. Patterson, "FIDES: Enhancing trust in reconfigurable based hardware systems," 2015 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, 2015, pp. 1-7.

[4] A. Basak, S. Bhunia, T. Tkacik and S. Ray, "Security Assurance for System-on-Chip Designs With Untrusted IPs," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 7, pp. 1515-1528, July 2017.

[5] X. Wang, Y. Zheng, A. Basak and S. Bhunia, "IIPS: Infrastructure IP for Secure SoC Design," in IEEE Transactions on Computers, vol. 64, no. 8, pp. 2226-2238, Aug. 1 2015.

[6] S. Bhunia, M. S. Hsiao, M. Banga and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," in Proceedings of the IEEE, vol. 102, no. 8, pp. 1229-1247, Aug. 2014.

[7] M. Farkash, B. Hickerson and B. Samynathan, "Data mining diagnostics and bug MRIs for HW bug localization," 2015 Design, Automation and Test in Europe Conference and Exhibition (DATE), Grenoble, 2015, pp. 79-84.

[8] J. Zhang and Q. Xu, "On hardware Trojan design and implementation at register-transfer level," 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Austin, TX, 2013, pp. 107-112.

[9] H. Salmani, M. Tehranipoor and R. Karri, "On design vulnerability analysis and trust benchmarks development," 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, 2013, pp. 471-474.

[10] D. L. Perry and H. D. Foster, Applied Formal Verification, New York, USA: McGraw -Hill, 2005

[11] A. Waksman, J. Rajendran, M. Suozzo and S. Sethumadhavan, "A red team/blue team assessment of functional analysis methods for malicious circuit identification," 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2014, pp. 1-4.

[12] J. Zhang, Feng Yuan, Lingxiao Wei, Zelong Sun and Q. Xu, "VeriTrust: Verification for hardware trust," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2013, pp. 1-8.

[13] J. Rajendran, V. Vedula and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2015, pp. 1-6.

[14] C. Krieg, M. Rathmair and F. Schupfer, "A Process for the Detection of Design-Level Hardware Trojans Using Verification Methods," 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), Paris, 2014, pp. 729-734.

[15] M. Rathmair, F. Schupfer and C. Krieg, "Applied formal methods for hardware Trojan detection," 2014 IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne VIC, 2014, pp. 169-172.

[16] J. Rajendran, A. M. Dhandayuthapany, V. Vedula and R. Karri, "Formal Security Verification of Third Party Intellectual Property Cores for Information Leakage," 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), Kolkata, 2016, pp. 547-552.

[17] J. Portillo, E. John, "Enhancing Trojan detection by finding LTL and Taint properties in RTL circuit designs: A Case Study", Accepted for: Computational Science and Computational Intelligence Conference and International Symposium and Cyber Warfare (CSCI-ISCW) 2018, Las Vegas, NV, USA, Dec 2018.

[18] J. Portillo, E. John and S. Narasimhan, "Building trust in 3PIP using asset-based security property verification," 2016 IEEE 34th VLSI Test Symposium (VTS), Las Vegas, NV, 2016, pp. 1-6.

[19] M. Hicks, C. Sturton, S. T. King and J. M. Smith, "SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs," in Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.517-529, March 2015.