INTEGRATION the VLSI journal xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Runtime hardware Trojan monitors through modeling burst mode communication using formal verification

Faiq Khalid[a,*], Syed Rafay Hasan[b], Osman Hasan[c], Falah Awwad[d]

[a] Department of Computer Engineering, Vienna University of Technology, Vienna, Austria
[b] Department of Electrical and Computer Engineering, Tennessee Tech University, Cookeville, TN, USA
[c] School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan
[d] Electrical Engineering Department, United Arab Emirates University, Al-Ain, United Arab Emirates

ARTICLE INFO

Keywords:
Hardware Trojans
Runtime monitors
Network on Chip (NoC)
Burst mode communication
Formal verification
Model checking

ABSTRACT

Globalization trends in integrated circuit (IC) design using deep sub-micron (DSM) technologies are leading to increased vulnerability against malicious intrusions. Various techniques have been proposed to detect such threats during design or testing phases of ICs. However, due to infinitely many possibilities of Trojans, there exists a possibility that some of these intrusions go undetected. Therefore, runtime Trojan detection techniques are needed to detect the Trojans for complete operation lifetime as a last line of defense. In this paper, we proposed a generic methodology, which leverages the burst mode communication protocol, to detect the intrusions during runtime. Our methodology has three phases: 1) behavioral modeling of design specifications along with its verification using linear temporal logic (LTL) in the model checker. 2) Counterexamples generated in phase 1 are used to insert run-time monitors at vulnerable paths. 3) Embed run-time monitors into the system and validate it. Unlike the other state-of-the-art techniques, the proposed methodology can be easily used to design the runtime monitoring setup without having netlist information of IP modules. We validated our approach by applying it on the AES Trojan benchmarks that utilize intermodule interface to communicate with other modules in the system on chip (SoC).

1. Introduction

With the globalization of integrated-circuit chip design-process, the chances of malicious hardware design intrusion, known as hardware Trojan, have grown tremendously [1–4]. Hardware Trojans can lead to many unwanted activities, including leaking confidential information, changes in the timing characteristics of the circuits, malfunctioning, denial of service and counterfeiting [2,5,6]. Various techniques have been developed to detect hardware Trojans. Some of the prominent works include micro-architecture modification to improve triggering of the potential Trojan payload during testing phase [7] and the usage of inherent error detection of quasi delay insensitive (QDI) architectures to detect malicious intrusions [8–10].

However, the intruder may come with ingenious techniques to overshadow hardware Trojan detection techniques. For example, in a SoC design, some hard or firm IPs may hide Trojans depending on the aging of the chip [42]. The possibility of detecting these Trojans during the test phase is very low, and they may get activated once the chip is in use [11]. Runtime approaches, on the other hand, could monitor an IC for its entire operational lifetime, providing a last-line of defense [43]. Therefore, specialized techniques have been developed to detect the Trojans during runtime [12–16]. The main drawback of the runtime techniques has been the large overhead [17], i.e., area overhead [2,44] in the path delay characterization [18]. Therefore, in order to reduce the overhead, Forte et al. proposed a temperature sensor based methodology to detect the Trojans during runtime [19]. This methodology analyzes the abnormal behavior of built-in temperature sensors of ICs to detect the malicious activities. Similarly, Bao et al. have improved the temperature tracking by considering the temperature change due to power leakage [20]. These techniques require a precise calibration over the environmental changes and process variations, and also rely on the premise that triggering of payload results in a substantially higher current flow. Zhao et al. exploited the dynamic thermal management techniques of ICs to detect Trojans during runtime [21]. A key feature of this technique is to analyze the thermal profile of the ICs to obtain the dynamic thermal/power parameters using the Chaos theory and hence small changes in current flow could be detected. However, this approach inherits the overhead of the classification algorithm and majority voting schemes and thus compromises the performance of ICs. Recently, Ngo et al. have proposed a

* Corresponding authors.
E-mail addresses: faiq.khalid@seecs.edu.pk (F. Khalid), shasan@tntech.edu (S.R. Hasan).

https://doi.org/10.1016/j.vlsi.2017.11.003
Received 23 November 2016; Received in revised form 22 October 2017; Accepted 8 November 2017

methodology to use the hardware property checkers (HPC) for the runtime Trojan detection [22]. In this methodology, the first step is to identify and verify the critical behavioral invariants using assertion based property specification language. These verified critical behavioral invariants are used to design the HPC, which are then embedded in the IC to verify the properties during runtime. This method is vulnerable to Trojan insertions at the netlist or layout levels and presumes access to IP modules, which eventually results in its limited utility for SoC designs that require third party IPs.

In this paper, we propose a generic methodology to design runtime monitoring circuits, which utilizes information from SoC integration phase only. We treat IPs in the SoC as black boxes with no access to their internal details are presumed. The proposed methodology consists of two phases: the first is to model the IP modules by translating their functional characteristics into a behavioral model and obtain linear temporal logical (LTL) properties. The behavioral model is then verified using nuXmv [23], which is a symbolic model verification (SMV) model checker to detect the vulnerable paths against the critical behavioral invariants. In the next phase, these vulnerable paths are analyzed to design the runtime monitors. In order to illustrate the proposed methodology, we have modeled advanced encryption standard (AES) hardware Trojans benchmarks, available on trust-hub website [24], in SMV and extracted their counterexamples, in case of LTL property failures.

This paper presents a runtime-monitoring unit, which is designed by analyzing these counterexamples. We demonstrated that our monitoring unit can detect all the AES hardware Trojan benchmarks that utilize, in some form, the communication network. We proposed to divide the system into two modes of operation, i.e., normal and testing, for systems where the burst mode communication is not inherently used. The system may be switched to test mode by the user in the presence of suspicious system activity, which can be activated automatically using some sort of optimum scheduling mechanism. In this mode, the system is forced to use the burst mode communication protocol for testing communication and runtime monitors to detect any abnormal behavior. Moreover, in order to reduce the overhead we also propose three different approaches to place the runtime monitors, which are namely; global, region based and channel based runtime monitoring setup. The overhead comparative analysis shows that effectiveness of each approach depends upon the communication topology of network on chip (NoC) and number of communicating modules. In comparison with the state-of-the-art run-time Trojan detection techniques, our approach does not require the access to IP design. Moreover, our approach assumes that the defender has access to the SoC integration unit only, which to the best of our knowledge, is a most practical and constrained assumption. Our analysis also shows that power and test-time overhead is comparable in most cases especially with the proposed region-based runtime monitors.

The main contributions of this paper are as follows:

1. A methodology is proposed to analyze the effects of intruded third party IP modules in NoC through a burst mode communication protocol.
2. Our run time hardware Trojan detection technique is implemented at the SoC integration unit only and no presumptions about IP modules are made.
3. A novel design of runtime monitors to detect the Trojans in NoC is proposed, and practically demonstrated.
4. In order to reduce the overhead, we propose three different approaches to place the runtime monitors, which are namely; global, region based and channel based runtime monitoring setup
5. We propose an alternative solution that includes the the first come first serve policy with monitors to handle the active channels and IPs because in most of the cases, not all IPs in a NoC fail at the same time.
6. We generalized the comparative analysis for the overhead of the proposed runtime monitoring setup.The rest of the paper is organized as follows: Section 2 provides some preliminary concepts required to understand the paper better. An overview of the proposed methodology is given in Section 3. Section 4 presents modeling and verification through the burst mode communication protocol. Section 5 provides the vulnerability analysis of the burst mode communication. Section 6 explains the proposed runtime monitoring solution, simulated results and comparison with different proposed runtime monitoring setups with respect to the NoC topologies. Section 7 discusses the pros and cons of the proposed approach in comparison to the state-of-the-art techniques. Section 8 concludes the paper.

2. Preliminaries

This section provides a short introduction to some of the preliminaries to facilitate the understanding of the rest of the paper. To elaborate on the communication mechanism of standard Bus protocols' implementation, we are describing fundamental three handshaking protocols and four most commonly network on chip topologies.

2.1. Handshaking protocol

In the handshaking protocol, the sender and receiver modules assert and negate the request (R) based on the corresponding assertion and negation of acknowledgment (ACK) signals. Based on the negation of request and acknowledgment signals, the handshaking protocol can be divided into the following three categories [25].

2.1.1. Full Handshake (FH)

In the full handshake protocol, the sender and receiver modules wait for the acknowledgment from the other module, before initiating or terminating their respective signals. The sender and receiver modules communicate with each other using R (request) and ACK (acknowledgment) signals. First, the sender module asserts a request, which is detected by the receiver module. After verifying that the signal is valid, the receiver module asserts an acknowledgment signal. The sender module negates the request and does not assert a new request until the receiver module negates its acknowledgment signal.

The state-space model for the full handshake protocol is shown in Fig. 1 [25]. In this model, the sender module has three states: "*IDLE*", "*Assert R*" and "*Negate R*", which represent the idle mode, assertion and negation of request (R) signal, respectively. The receiver module has two states: "*Negate ACK*" and "*Assert ACK*", which represent the assertion and negation of acknowledgment (ACK) signal, respectively. In this protocol, the sender receives the data ("*D = 1*") to send, it asserts the request ("*R = 1*") and waits for the corresponding acknowledgment from the receiver ("*ACK = 1*"). After the completon of a data transaction, the data signal is negated ("*D = 0*"), the request signal is negated ("*R = 0*") and the corresponding acknowledgment signal is also negated ("*ACK = 0*").

2.1.2. Partial Handshake I (PH-I)

The second type of protocol is the partial handshake. In this type of handshake, the sender and receiver modules do not wait for each other

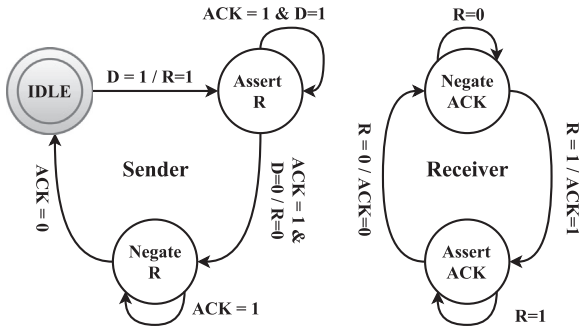

Fig. 1. State-space model of Full Handshake Protocol [25].