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

Faiq Khalid\textsuperscript{a,}\textsuperscript{c}, Syed Rafay Hasan\textsuperscript{b}, Osman Hasan\textsuperscript{c}, Falah Awwad\textsuperscript{d}

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

\textbf{ARTICLE INFO}

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

\textbf{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 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
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 handshake 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 completion 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].](image-url)
before they negate their respective signals and move on with the handshaking process. When the sender module asserts its request signal, the receiver module acknowledges it with a single clock wide pulse. In this case, the receiver is not concerned when the sender will negate its request. On the other hand, to make the protocol work properly, the sender needs to keep the request signal negated for at least one clock cycle as otherwise the receiver cannot differentiate between a new and a preceding request [25].

The state-space model for the full handshake protocol is shown in Fig. 2 [25]. In this state-space model, the sender module has two states: “Assert R” and “Negate R”, which represent the assertion and negation of request (R) signal, respectively. The receiver module has three states: “IDLE”, “Assert ACK” and “End”, which represent the idle state, assertion of acknowledgment (ACK) signal and end of data communication, respectively. In this protocol, whenever the sender receives the data (“D = 1”), it asserts the request (“R = 1”) and waits for the corresponding acknowledgment, which is asserted by the receiver (“ACK = 1”). After completing the data communication, the sender negates the data signal (“D = 0”), the sender negates the request (“R = 0”) and the receiver ends the communication without the negation of acknowledgment. Moreover, the receiver can assert a new request without waiting for the negation of acknowledgment signal.

2.1.3. Partial Handshake II (PH-II)

In this protocol, the sender asserts its request signal consisting of a single clock wide pulse, which is acknowledged by the receiver with a single clock wide pulse. The state-space model for the full handshake protocol is shown in Fig. 3 [25]. In this state-space model, the sender module has three states: “IDLE”, “Assert R” and “End”, which represents the idle state, assertion of request (R) signal and the end of data communication, respectively. The receiver module has three states: “Negate IDLE”, “Negate ACK” and “End”, which represent the idle state, assertion of acknowledgment (ACK) signal and the end of data communication, respectively. In this protocol, whenever the sender receives the data (“D = 1”), it asserts the request (“R = 1”) and waits for the corresponding acknowledgment, which is asserted by the receiver (“ACK = 1”). After completing the data communication, the sender negates the data signal (“D = 0”) and the sender negates the request (“R = 0”) and the receiver ends the communication without the negation of acknowledgment. Furthermore, the sender does not require to negate the request signal but it can start asserting of a new request even without waiting for the negation of the acknowledgment signal.

2.2. Network on chip topology

Network on Chip (NoC) topology is an arrangement of various communicating modules on chip [26]. Some of the common implementations are shown in Fig. 4.

1. Ring: A network topology is set up in a circular fashion in such a way that they make a closed loop as shown in Fig. 4a. This way data travels around the ring in one direction and each device on the ring acts as a repeater to keep the signal strong as it traverses. Each device incorporates a receiver for the incoming signal and a transmitter to send the data on to the next device in the ring. However, in this topology, if one module stops working, the entire network gets affected or stops working [26].

2. Fully Connected Mesh: In this communication network each node is connected to others as shown in Fig. 4b. Its major disadvantage is that the number of communication channels (Nch) grows quadratically with the number of communicating modules [26].

3. Partially Connected Mesh: The type of network topology in which some of the modules are connected to more than one module in the network through point-to-point link, which allows to take advantage of some of the redundancy that is provided by a physical fully connected mesh topology without the expense and complexity required for a connection between every node in the network [26]. Based on the neighboring schemes, this topology can be implemented in the following different ways:

- 4-connectivity: In this topology, a communicating module can communicate with up to four immediate neighbors, as shown in Fig. 4c, where, the central modules can communicate with their neighbors in north, south, east and west [26].
- 8-connectivity (Mesh): In this topology, a communicating module can communicate with up to eight immediate neighbors, as shown in Fig. 4d, where, the central modules can communicate with their neighbors in north, south, east, west, northeast, northwest, southeast and southwest [26].

2.3. nuXmv model checker

In this work we used a model-checking tool, which facilitates in the modeling of the system, verifying its salient properties and generating counterexamples when verification of a property fails. We used the nuXMV model checker for our work, because of its distinguishing
ability to handle real numbers and implicit handling of state counters. Thus, the continuous values of delays and uncertainties can be modeled properly, which cannot be done in other model checking tools in such a straightforward manner.

The nuXmv symbolic model checker [23] extends the capabilities of the NuSMV by complementing its verification techniques using SAT algorithms for finite state systems. For infinite state systems, it introduces new data types of Integers and Reals and also provides the support of Satisfiability Modulo Theories (SMT), using MathSAT, for verification [27].

The system that needs to be modeled is expressed in the nuXmv language, which supports the modular programming approach where the overall system is divided into several modules that interact with one another in the MAIN module. The properties to be verified can be specified in nuXmv using the Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). The LTL specifications are written in nuXmv with the help of logical operations like, AND (\&), OR (\lor), Exclusive OR (xor), Exclusive NOR (xnor), implication (\rightarrow) and equality (\equiv \rightarrow), and temporal operators, like Globally (G), Finally (F), next (\nu) and until (\mu). Similarly, the CTL specifications can be written by combining logical operations with quantified temporal operators, like exists globally (Ex), exists next state (Es) and for-all finally (Af). In case a property turns out to be false, a counterexample in the execution trace of the FSM is provided.

3. Proposed methodology

This section explains our proposed methodology for runtime hardware Trojan detection through modeling burst mode communication using formal verification techniques.

3.1. Threat model

Developing and using appropriate threat models is one of the foremost steps in developing any methodology for detecting intrusions in the domain of hardware security. Therefore, based on vulnerabilities and potential threat in integrated circuit design cycle, researchers have broadly categorized the threat models for hardware security in ICs into 7 different classes, as shown in Table 1 [28]. In the proposed methodology, we are using the threat model “A”, which states that the third-party IP (3PIP) vendors are not trusted and designers are using these 3PIPs to develop the SoC because it is almost impossible for SoC developers to develop all necessary IPs in house. This adversarial model is one of the most commonly used one due to the widespread usage of System on Chips (SoCs). Third party IPs are widely used to reduce time to market and thus the utilization of 3PIPs is encouraged, and hence the SoC integration can be done in a trusted facility while the 3PIPs are not trusted. Hardware Trojans can be inserted into 3PIPs by IP vendors during IP design to steal security information/data from other IPs in the system. A Trojan can be hidden during the normal functional operation of the 3PIP supplied as register transfer level (RTL) code. When such a Trojan exists in an IP core, then all the fabricated ICs with this IP core would contain Trojans. The only trusted component would be the specification from the SOC designers that defines the function, primary input and output, and other information about the 3PIP that they intend to use in their systems. Moreover, the specification and source code, provided by the vendor may contain Trojans.

3.2. Targeted Trojans

Typically, hardware Trojans have multiple types of payload but leaking the confidential information is one of the most dangerous types in terms of privacy and security. Based on the triggering of such payloads, these Trojans can be categorized as: “Physical access”, “near the device”, “near a communication channel”, and “far away” [29]. (1) Physical access means that the attacker can physically interact with the device. (2) Near the device means that the attacker can get close enough to interact with the device, e.g., within th range of wireless control. (3) Near a communication channel implies that the interaction with the device does not occur directly but rather over some communication channel. (4) Far away means that the attacker does not have access to the Trojan physically or through any of its communication channels. The payload of these Trojans, leaking the information, can be implemented in two different ways: The first method is to establish the covert channel to leak information via a RF signal. Considering the access of device for an attacker, these implementations require access to the device physically or they have to be near the device for receiving RF signals since, because of low power design, the strength and range of RF signals are usually kept low [29,30]. The second method is to leak the information through an existing communication channel, e.g., Trojans presented in [31,32], leak information through the Advanced Micro-controller Bus Architecture (AMBA) bus. The leakage of information via a covert channel either requires additional area or consumes extra power, which can be detected using the typical side channel analysis. However, leaking the information via existing communication channel consumes less power and has less activation time and often remains undetected in typical side-channel based hardware Trojan detection [33]. Therefore, in this paper, we propose a methodology for runtime detection of the Trojans that leak the confidential information via existing communication channels, have a short activation time and communicate the leaked information in small chunks without violating the protocol, e.g., the benchmark AES Trojans, T100, T1100 and T300 leak a single bit of secret key over many clock cycles [34,35]. Therefore, the information leakage over the communication channels requires a very small number of clock cycles that violate the minimum number clock cycles for a particular burst mode communication, which can be controlled or changed by the designer or in some cases by users during runtime [36].

3.3. Proposed runtime HT detection methodology

In order to detect the above-mentioned payloads, the proposed methodology exploits the timing and communication behavior to identify the abnormal communication. We propose to monitor the minimum number of clock cycles required in burst mode communication. Thus the proposed methodology leverages upon the formal modeling of the burst mode communication to identify the vulnerabilities in the communication network, and its requires the following four steps, as shown in Fig. 5: (1) Model Translation, (2) Model Verification, (3) Vulnerability Analysis and (4) Runtime detection or monitoring.

3.3.1. Model translation

The first phase of the proposed methodology is to extract the behavior and specification or design constraints from the given system. This behavior/functionality and design specifications of the system are
manually expressed as a formal model and linear temporal logic (LTL) properties, respectively. The accuracy of translation is highly dependent on the translated properties, which should be generic and independent of the given case studies [37]. For example, any communication channel should hold the functional and behavioral properties of a specific protocol that it is using for communication. Thus, in the proposed methodology, we exploit the behavioral anomalies to analyze the vulnerabilities against hardware Trojan benchmarks. Since, our threat model presumes 3PIP modules as black boxes, therefore, SoC designers have access to the functional and behavioral information of the 3PIP only and not to the exact implementation. Hence, we resort to SoC integration phase of the design cycle to model 3PIP as a black box, which communicates with other SoC components using a certain communication protocol. Among the given SoC integration’s hardware units, which includes communication BUS, clock distribution network (CDN), power distribution network (PDN) etc., we choose the most commonly used integration unit for information leakage attacks, like communication channel, and analyze whether the modeling of this unit can lead to design the modules for runtime Trojan detection or not. The major challenges in modeling the black box communication modules are protocol and communication behavior accuracy. The protocol modeling and analysis does not involve any random behavior because for a particular input it operates in certain way, i.e., if a request signal is received it tries to establish a communication channel based on the certain predefined policy, such as handshake protocols or other synchronization strategies. However, once the channel is established then the time required for the communication is dependent upon the size of communication data which is usually non-deterministic. Thus, it makes the timing behavior non-deterministic, e.g., in our proposed solution, the required time per burst mode communication is non-deterministic. Similarly, the behavior of the benchmark Trojans that we have used to identify the anomalies is also non-deterministic, e.g., AES T1200 leaks the information when the input data state (that is non-deterministic) is 128'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF. Therefore, in order to capture this behavior we use the non-deterministic assignment of burst mode values.

3.3.2. Model verification

In the next phase, the translated behavioral and functional models are verified in a model checker based on the translated LTL properties. This is the most important step because the accuracy of the formal model is mainly judged based on the properties that are used in its verification. We propose to verify the translated model in three steps: In the first step, the model is tested for the functional properties of the communication channel and protocol, e.g., if a module sends a request it will eventually establish the communication channel. If these properties fail in this phase then the model is modified according to the counterexample and this process is repeated till the model verifies all of these properties. In the next step, it checks the timing behavior of the communication channel and protocol, e.g., whenever a module asserts a request to communicate then it takes only 5 clock cycles to establish the communication channel, which is obtained from an established timing property for full handshaking protocols [25,38,39]. Similar to the previous step, the model is updated if a counterexample is generated and this process is repeated until the formal model verifies these timing properties. In the third step, we embed the burst mode communication behavior in the model which is also taken from its basic communication behavior and if the model does not hold the properties of this behavior then it is revised until it verifies the behavior.

3.3.3. Vulnerability analysis

The next phase is to analyze the verified design behavior for some possible benchmark vulnerabilities, i.e., for our case study we used AES Trojan benchmarks available on trust-hub.org [24]. Among various types of hardware Trojan benchmarks available on [24], it is observed that most types of AES Trojans require inter-module communication. Therefore, in this work, we restricted our modeling to the communication channels. This decision is primarily based on the verification of the system model, in conjunction with the LTL or CTL properties to address the vulnerabilities in the design. In case of property failure, the model checker generates the counterexamples, which are translated into the potential attack paths.

3.3.4. Design of runtime monitors/improvements against vulnerabilities

The identified vulnerable paths of the systems are used to recognize the potential nodes in the design where hardware Trojan monitors should be inserted. A trade off among the different possible choices of run-time monitors is made in this design step. Our goal is to detect the hardware Trojan during run-time for a possible abnormal behavior that goes undetected during normal testing phase. The proposed runtime monitors are embedded into the design to observe the identified potential vulnerable nodes. The improved design models along with runtime monitors are verified against the identified vulnerabilities in the design. This recommended hardware model, which contains runtime monitors, is implemented as a hardware design using the CAD tools. Finally, a comparison of run-time monitor based hardware is made against the original implementation of system to analyze the
performance, power and area overhead. This overhead is then used to make the proper trade-offs for runtime monitors with respect to its performance, efficiency and reliability.

4. Modeling and verification of burst mode communication

This section briefly explains all the assumptions and parameters and mathematical modeling that are required to translate and verify behavior of burst mode communication in NoC.

4.1. Modeling in nuXmv

In order to illustrate the usefulness of our proposed methodology, we modeled the burst mode communication in nuXmv for different number of communicating modules. Fig. 4 shows the conceptual diagram modeling and simulation setup of burst mode communication with six communicating modules and protocols. In each NoC setup, there are six IP modules that are operating at different frequencies and are communicating with each other in burst mode using one of the handshaking protocols, described in Section 2. For the six communicating modules, we modeled a ring, fully, 4-connected and 8-connected mesh network topology, respectively, as shown in Fig. 4. To illustrate the possible hardware Trojan behavior, we tested these network with one randomly chosen intruded module in each case. We have provided the complete nuXmv models in our website [40].

In order to extract the behavioral and timing models of burst mode communication, first we obtained the clock cycle overhead for each implemented handshake protocol, shown in Fig. 4. For numerical analysis, without loss of generality, we considered that data communication is performed in burst mode for 1000 clock cycles at a time. This analysis is general and can be applied to any number of clock cycles per burst mode. Fig. 6 shows an overhead of 11, 8 and 5 clock cycles for full handshaking, partial handshaking I and partial handshaking II, respectively. Hence, the minimum time between any two requests must be equal to or greater than (1000 + overhead) clock cycles.

To extract the timing properties, we formalize the following definitions for burst mode communication based on the handshake protocol, for critical timing components. The nuXmv modeling of these timings is explained later.

1. **Request Time** ($T_{req}$): Time between two consecutive requests in handshaking protocols.
2. **Request Assertion Time** ($T_{req-a}$): Time required to assert the requests in handshaking protocols. For example in Fig. 6, it is 2 clock cycles, with the assumption of two-flop synchronizer.
3. **Request De-assertion Time** ($T_{req-d}$): Time required to de-assert the requests in handshaking protocols. This parameter is not required for partial handshaking protocol II.
4. **Acknowledgment Time** ($T_{ack}$): Time between two consecutive acknowledgments in handshaking protocols.
5. **Acknowledgment Assertion Time** ($T_{ack-a}$): Time required to assert the acknowledgments in handshaking protocols.
6. **Acknowledgment De-assertion Time** ($T_{ack-d}$): Time required to de-assert the acknowledgments in handshaking protocols. This parameter is required for full handshake protocol only.
7. ** Burst Mode Time** ($T_{burs}$): Number of clock cycles per burst mode data communication.
8. **Number of Communication Channels** ($N_{ch}$): It is defined as the total number of communication channels in NoC. It depends upon the NoC topology and can be calculated from the Eqs. (1)–(4) for respective on chip network.

- In **Ring** topology, the total number of communication channels ($N_{ch}$) are equal to total number of the communicating modules ($N_{m}$) as shown in Eq. (1):

$$N_{ch} = N_{m}$$

(1)

- In **Fully Connected (Mesh)** topology, the total number of communication channels ($N_{ch}$) can be calculated from Eq. (2), which is derived form the statistical analysis of the number of communicating modules along with the second order curve fitting algorithm.

$$N_{ch} = \frac{N_{m}(N_{m} - 1)}{2}$$

(2)

- For **4-Connected** mesh network topology, we statistically analyzed the number of communicating modules to derive Eq. (3) which represents the total number of communication channels ($N_{ch}$). The first step of the statistical analysis is to obtain the database of the total number of channels for different number of communicating modules as shown in Fig. 7a. In the next step, we applied the 2nd order polynomial curve fitting technique to obtain the following equation.

$$N_{ch} = \text{ceil}(0.0033(N_{m})^2 + 1.6136(N_{m}) - 2.6548)$$

(3)

- Similarly, for **8-Connected** mesh network topology, we derived Eq. (4) to calculate the total number of communication channels ($N_{ch}$) as a result of statistical analysis shown in Fig. 7b.

$$N_{ch} = \text{ceil}(0.0102(N_{m})^2 + 2.8011(N_{m}) - 5.1398)$$

(4)

4.1.1. State-space modeling

In order to model handshaking protocols with burst mode communication, we have modified the typical state machines of handshaking protocols presented in Figs. 1–3. The brief description of the modified state-space modeling of handshaking protocols is given below:

In the full handshake protocol, each module waits for the acknowledgment from other module, before initiating or terminating their respective signals. Whenever, the sender module wants to communicate with the other available module, it asserts R (request) signal to start communication, which is detected by the receiver module and waits for the acknowledgment in Assert Req state. After seeing the
asserted R signal, the receiver module asserts an ACK (acknowledgment) signal to acknowledge the communication request. Once the channel is established, the system starts communication and waits until the data burst is received. However, the size of data burst must be greater than the minimum size which ensures its burst mode communication. After the successful communication, system negates R and ACK signals. The state-space model of full handshake protocol with burst mode communication is shown in Fig. 8.

Similarly, partial handshake protocols follow the same procedure to establish the communication channel, as shown in Figs. 9 and 10. However, after the successful communication, the negation of request and acknowledgment is different from the full handshaking protocol. In partial handshake protocol I, the negation of ACK is not required, thus the receiver does not negate the acknowledgment and the sender directly transits to IDLE state from Negate Req without receiving ACK = 0, as shown in Fig. 9. In partial handshake protocol II, the negation of R and ACK signals is not required, thus the sender and receiver directly transit to IDLE state from Negate Req and Assert ACK without receiving ACK = 0 and R = 0, respectively, as shown in Fig. 10.

4.2. Verification in nuXmv

We used Version 1.0.1-win64 of the nuXmv model checker along with the Windows 10 Professional OS running on a core i5 processor, 2.93 GHz (4 CPUs), with 8 GB memory for our experiments. In order to verify the vulnerabilities of burst mode communication, we formulated the following properties. These properties cover the cases related to intrusions that may affect the data rate and timing constraints of control signals.

4.2.1. Deadlock

A deadlock state in a system is defined as an undesired cyclic behavior, where system gets stuck in one of the possible states forever. This can be modeled for the case when a module CDC0 is communicating with module CDC1 by checking if CDC0.req eventually receives an acknowledgment.

![Fig. 7. Statistical analysis of partially connected mesh.](image)

![Fig. 8. State-space model Full Handshake Protocol.](image)

![Fig. 9. State-space model Partial Handshake Protocol I.](image)

![Fig. 10. State-space model Partial Handshake Protocol II.](image)
\[ G(\text{CDC0. req} = \text{TRUE}) \rightarrow \neg F(\text{CDC0. ack0} \& \text{CDC1. turn}) \]  

4.2.2. Data rate

Data rate is defined as the number of data transaction cycles communicated in one second. In case of our setup, the number of data transaction cycles for one communication cycle must be greater than or equal to minimum clock cycles required for burst mode communications.

\[ G(\text{CDC0. ack0} = \text{TRUE}) \rightarrow \neg (\text{CDC0. dcycles} > \text{Dburstmode}) \]  

4.2.3. Timing verification

Following properties are based on minimum delay constraints of the handshake communication shown in Fig. 6.

In handshake protocol, the minimum time between two communication requests is time to complete data transaction. It must be greater than or equal to the sum of request assertion and negation time (\(T_{a-req} + T_{a-neg}\)) of the sending module CDC0, acknowledgment assertion and negation time (\(T_{a-ack} + T_{a-nack}\)) of the receiver module (CDC1) and the minimum clock cycles required for burst mode data communication (\(T_{\text{min-burst}}\)) as shown in Eq. (5):

\[ T_{eq} \geq T_{a-req} + T_{a-ack} + T_{\text{min-burst}} + T_{a-neg} + T_{a-nack} \]  

(5)

This is represented in nuXmv as follows:

\[ G(\text{CDC0. treq} > \text{CDC0. tareq} + \text{CDC1. taack} + T_{\text{min-burst}} + \text{CDC0. tdreq} + \text{CDC1. ttaack}) \]  

(III)

In the partial handshake protocol II, negation of request signals is not required as can be seen in Fig. 6. Therefore Eq. (5) can be modified as (6) for this protocol.

\[ T_{eq} \geq T_{a-req} + T_{a-ack} + T_{\text{min-burst}} \]  

(6)

The time between any two consecutive communication requests must be greater than or equal to the sum of time required to establish the channel (\(T_{a-req} + T_{a-ack} + T_{\text{min-burst}}\)), where \(T_{a-req}\) is considered with respect to the sender module or CDC0, and \(T_{a-ack}\) is considered with respect to the receiver module or CDC1, and \(T_{\text{min-burst}}\) is the minimum clock cycles required for the burst mode data communication.

\[ G(\text{CDC0. treq} > \text{CDC0. tareq} + \text{CDC1. taack} + T_{\text{min-burst}}) \]  

(IV)

5. Vulnerability analysis/verification of burst mode communication

In order to identify the potential vulnerable paths in the systems, we assumed that each module of experimental setup is encrypted using AES, and one of them is intruded. These modules are communicating through Code-Division Multiple Access (CDMA). To insert the Trojan behavior in AES, we modeled the behavior of some standard AES benchmark Trojans available on trust-HUB [24]. All AES Trojans are modeled as the non-deterministic change in burst size of the communication data depending upon their implementation and behavior, e.g., the T100 generates the burst of 11 clock cycles. After the intrusion, the third phase is repeated again and if it generates the counterexample then it is considered as the detection of abnormal behavior. Most of the available benchmarks leaks the encryption key which can be done using two methods: Firstly, it can either use RF antennas or it can use the existing communication buses to leak the encrypted key. In this paper, we are targeting the leakage of encrypted key using communication buses. In the AES T100 benchmark, Trojan leaks a single bit over many clock cycles through the Code-Division Multiple Access (CDMA) and the Trojan is triggered only for 2 clock cycles. We translated this Trojan behavior into the full handshake burst mode communication. We modeled this behavior by non-deterministically assigning the following burst mode values 0, 2, 100, 1000, 2000, 3000 after the channel is established. So if the number of data transaction cycles are less than 1000 for one communication cycle then it is highly probable that it is intruded and the information is leaked. Thus the intruded model is verified based on LTL properties (I), (II), (III) and (IV). During the verification, nuXmv generates the counterexamples, which are further analyzed to identify the Trojans paths and nodes. Table 2 identifies the properties that fail for each Trojan. It shows that AES T1000, T200, T2100, T400, T500 and T600 responded to the properties we stated, as shown by \(\times\) in Table 2.

Vulnerability analysis is illustrated with a representative counterexample, generated during verification of intruded full handshake protocol with burst mode communication, as shown in Fig. 11. Typically, intruders leak very few data bits during a communication cycle to avoid hardware Trojan detection measures. As an example, AES benchmark T100 leaks only 2 bits of secret key per communication link. In Fig. 11, the red highlighted transitions show the counterexample generated due to AES T1000 in full handshake protocol with burst mode communication. Under the normal operation, the system negate the request after waiting the \(T_{\text{min-burst}}\) clock cycles. Since AES T100 leaks 2 bits per communication, the sender and receiver negate the request and acknowledgment before the \(T_{\text{min-burst}}\) clock cycles which violate the minimum time required between two consecutive requests of Eq. (5).

Table 2

<table>
<thead>
<tr>
<th>AES Trojans</th>
<th>Deadlock</th>
<th>Data rate</th>
<th>Timing properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1000</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1200</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1300</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1400</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T1500</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T200</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T2000</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T2100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T300</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T400</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T500</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T600</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T700</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T800</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>T900</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

![Fig. 11. Counterexample for Full Handshake Protocol.](Image)

For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)
5.1. Counterexample analysis

In the next step of the proposed methodology, the counterexamples, which are obtained from analyzing different Trojan benchmarks, are shown in Table 2. The four columns in this table represent the results of four different LTL properties and each row belongs to a different AES Trojan benchmarks. A check mark (✓) in a cell means that the counterexample is not generated when that particular Trojan is tested against a specific property and a cross mark (✗) in the cell means that the counterexample is generated, and that the failure is identified. Since only one failure of the defined property is enough to detect the Trojans, therefore, these results show that 10 out of given 17 AES Trojan benchmarks can be detected just by properly monitoring the bus protocol. The reason for other 7 AES Trojan benchmarks not being detected is that they do not use a communication medium to leak the information, as in this paper, we are only considering the Trojans that use an exiting communication medium.

In the proposed methodology, the generated counterexamples are analyzed to identify the potential attack path or vulnerable signals, which require runtime monitoring. In the counterexample shown in Fig. 11, the AES Trojan benchmark T100 fails the data rate property (I), whereas, the red and black states in Fig. 11 have been explained earlier in this section. The analysis of all the counterexamples generated in Table 2 shows that all the Trojan benchmarks that generate the counterexamples, fail the the data rate property (I). Therefore, we propose a runtime monitoring approach that monitors the data rate, to detect the data communication, which requires lesser time than the minimum time required for a burst mode communication \(T_{\text{min-burst}}\).

6. Proposed runtime solution for vulnerabilities in burst mode communication

In this work, we designed a runtime monitoring unit, which detects the abnormal behavior of burst mode protocol during runtime. The premise is that, as shown in Fig. 5, after generating the counterexamples, we are able to analyze the vulnerable paths or modules of the design using the approach explained in Fig. 11. Since here our counterexamples suggest to monitor the signals involved in burst-mode communication protocol, therefore, we inserted our run-time monitors to monitor data communication during burst mode. Fig. 12 shows the complete design of our proposed global runtime monitoring solution. It consists of four units; the first unit selects the channel under observation, the second unit selects the direction of communication in the channel, the third unit is a counter to compute the clock cycles during burst mode data transaction. The last unit is a detector circuit which provides information if the number of clock cycles used by burst mode data transaction is within the acceptable limits. If the counter value is not within the predefined limits then this monitoring unit discards the communication.

In order to reduce the hardware overhead we proposed following three approaches to embed the runtime monitoring unit in NoC.

\[
\text{Area}_{\text{overhead}} = n_{\text{controller}} + n_{2\text{to1mux}} + n_{8\text{to1mux}} + n_{\text{counter}}
\]

where:

- \(n_{\text{controller}}\)
- \(n_{2\text{to1mux}}\)
- \(n_{8\text{to1mux}}\)
- \(n_{\text{counter}}\)

6.1. Channel based runtime monitoring (CBRM)

In this approach, one monitoring unit for each communication channel, is embedded into NoC. Fig. 13 shows the CBRM setup for different NoC topologies, in which a single runtime monitor is associated with a communication channel. The area overhead of this runtime monitoring setup can be calculated from the Eq. (7).

\[
\text{Area}_{\text{overhead}} = n_{\text{controller}} + n_{2\text{to1mux}} + n_{8\text{to1mux}} + n_{\text{counter}}
\]

where:

- No.of controller \(n_{\text{controller}}\)
- No.of 2 to 1 Mux \(n_{2\text{to1mux}}\)
- No.of 8 to 1 Mux \(n_{8\text{to1mux}}\)
- 2\text{nd} hit counter \(n_{\text{counter}}\)

6.2. Global runtime monitoring (GRM)

In this approach, a generic runtime monitoring circuit is designed, which is used to monitor the whole NoC. It has access to all channels and can analyze them one by one. Fig. 14 shows the GRM setup for different NoC topologies in which single runtime monitor has access to all channels. However, due to very large number of communication channels, it may miss the rare malicious events, i.e., a data communication which is completed in less than the minimum time required for a burst mode communication \(T_{\text{min-burst}}\). Its area overhead can be calculated from the same equation (Eq. (7)), with the following changes in each parameter:

- No.of controller \(n_{\text{controller}}\)
- No.of 2 to 1 Mux \(n_{2\text{to1mux}}\)
- No.of 8 to 1 Mux \(n_{8\text{to1mux}}\)
- 2\text{nd} hit counter \(n_{\text{counter}}\)

\[
m = \text{ceil}(\log\left(T_{\text{min-burst}}\right))
\]
where $T_{\text{min-burst}}$ and $N_b$ represent the minimum data transaction cycles required to complete the burst mode and total number of communication channels, respectively.

### 6.3. Region based runtime monitoring (RBRM)

This approach is a compromise between the overhead of CBRM and lesser monitoring resolution in GRM. In RBRM, the whole network is divided into regions such that each region has an equal number of communication channels. For each region, a runtime monitoring unit is inserted. Fig. 15 shows the CBRM setup for different NoC topologies. For example, in Fig. 15b, the network is divided into three regions (red, blue, and green). Each region has an equal number of channels and one runtime monitoring unit which monitors the 5 channels. In the proposed approach, we divided the region in such a way that each associated channel is close to the respective monitors. If the number of communication channels is a prime number then the remaining channel is observed by CBRM, e.g., if the number of communication channels are 31, then based on 30 channels, the network can be divided into equal regions but for an additional channel a separate monitors is installed in the NoC.

These RBRM units can reduce the possibility of missing the rare malicious activities, i.e., a data communication which is completed in less than the minimum time required for a burst mode communication ($T_{\text{min-burst}}$), because there is one monitoring unit for a particular region. This increases the performance, overhead which can be calculated by using Eq. (7), with the following changes in each parameter:

- No. of controller ($n_{\text{controller}}$) = $nr + (N_b - (nr \times ncr))$
- No. of 2 to 1 Mux ($n_{\text{2to1mux}}$) = $nr + (N_b - (nr \times ncr))$
- No of 8 to 1 Mux ($n_{\text{8to1mux}}$) = $\text{ceil}(nr \times \frac{N_b}{nr} - ncr)$
- $2^{n}$ bit counter ($n_{\text{counter}}$) = $nr + (N_b - (nr \times ncr))$
- $m = \text{ceil}(\log(T_{\text{min-burst}}))$

where $nr$ and $N_b$ represent the number of regions and total communication channels in the whole NoC, respectively. Total communication channels in a region ($ncr$) can be calculated from the following Eq. (8):

$$n_{cr} = \text{Floor}\left( \frac{N_b}{nr} \right)$$

All the above mentioned runtime monitoring setups are designed on the basis of burst mode property verification thus it can only detect the intrusion if the system is using the burst mode communication protocol, which is the case with most of the standard bus protocols. However, the proposed runtime monitors requires a certain time overhead detection, which is defined as the time required to extract the communication behavior of burst mode communication and for different monitoring setups is given below:

1. For Channel Based Runtime Monitoring (CBRM), the waiting time overhead is equal to the minimum data transaction cycles required to complete the burst mode ($T_{\text{min-burst}}$), because in this approach each channel has its own monitors and it can test the whole system within $T_{\text{min-burst}}$.

2. Similarly, for Global Runtime Monitoring (GRM) the test time overhead can be calculated from Eq. (9) as in this approach the monitoring setup tests one channel at a time. Therefore, its test time overhead is quite large.

$$\text{testtime}_{\text{overhead}} \geq N_b \times T_{\text{min-burst}}$$

where $T_{\text{min-burst}}$ and $N_b$ represent the minimum data transaction cycles required to complete the burst mode and total number of communication channels, respectively.

3. For Region Based Runtime Monitoring (RBRM) the test time overhead can be calculated from Eq. (10) as in this case the runtime monitor scans all the communication channels in the region, one by one.

$$\text{testtime}_{\text{overhead}} \geq ncr \times T_{\text{min-burst}}$$

where $T_{\text{min-burst}}$ and $ncr$ represents the minimum data transaction cycles required to complete the burst mode and total number of communication channels in a region, respectively.

Although, the proposed methodology provides a comprehensive mechanism runtime HT detection but to detect the anomalies in other communication protocols is out of the scope. Since the main contribution of this paper is to prove the viability that Trojans in 3PIP can be detected via SoC integration units, which is the NoC fabric in this case, hence no further investigation is made in this direction.
6.4. Simulations in CAD tools

The last step of our proposed methodology is to embed the runtime monitoring unit in the digital system. Our experimental setup consists of four IP Modules with six communication channels (Fig. 16a) and each communication channel consists of three control signals (Fig. 16b): request (R), acknowledgment (ACK) and data control (D).

We embedded the system with all the three proposed approaches to analyze their overhead. In the first approach, each communication channel has one monitoring unit. In the RBRM approach, we have the setup into two parts in such a way that each part has its own monitoring unit. In the GRM approach, runtime monitoring is done by using a global monitoring unit. The simulation of the above-mentioned approaches is done in Cadence tool suite using the 90 nm CMOS IHP technology. For the gate level implementation, static CMOS logic has been used. Transistor sizing in different gates, except for the feedback inverter of each gate, has been done using the traditional matching of the propagation delay of pull up and pull down networks. In this experiment, the minimum data transaction cycles for the burst mode is kept at 50.

Fig. 17 shows the full handshaking protocol behavior of channels 1 and 2 for the burst mode communication. The behavior of channel 1 is represented by the data out signal of IP module 3 and request signal of IP module 3, which shows that the minimum number of clock cycles required for data communication is much greater than the minimum number of data transaction cycles required for the burst mode communication, due to which the channel is not intruded. Since the IP module 2 is intruded with one of the previously mentioned AES benchmark Trojans therefore the communication behavior of channel 2 is abnormal as shown by the data out signal of the IP Module 2 and Request signal of IP module 2 to 1. The pulse at the output of runtime monitoring unit shows that this communication channel is not fulfilling the data rate requirement of burst mode, which confirms that IP module 2 is intruded.

6.5. Design metric comparison

In this section, we analyze and compare different scenarios in which our experimental setup can operate. Three different approaches are proposed to develop runtime monitors. Based on the protocol, embedded runtime monitoring units and total number of communicating modules, we have created 45 different scenarios for a particular NoC topology, i.e., Tables 4 and 5 show the 45 scenarios for ring, fully and partially connected mesh NoC topologies. In order to measure the detection precisions of different approaches, we simulated 10 abnormal communication behavior per 1000 clock cycles. This analysis helped us in quantifying the trade-offs among our proposed approaches.

The careful analysis of Tables 4 and 5 shows that power consumption of system with CBRM unit is quite high as compared to the global and RBRM unit because there must be one monitoring unit per communication channel. This approach does not require any channel selection module. It can also detect the abnormal behavior with high precision as shown in Table 3 (100%) and its test time overhead is equal to minimum data transaction cycles required to complete the burst mode communication. For example, in Table 6 its test time is 50 because $T_{\text{min burst}}$ is kept to 50. The power consumption of this approach is very high, which increases considerably with the increase in number of channels and communicating modules, as shown in Fig. 18. Similarly, its area overhead is almost up to 17 times and 25 times more than RBRM and GRM, respectively, as shown in Fig. 19.

The GRM approach fails to detect some rare events, i.e., a data communication which is completed in less than the minimum time required for a burst mode communication ($T_{\text{min burst}}$), because it has to check the overall communication channel one by one. So it is highly probable that this approach fails to detect the abnormal behavior of channels other than the channel under observation. Therefore, its detection precision is almost 80% as shown in Table 3. Unlike the
Fig. 18. Power Consumption of runtime monitoring setup using Full Handshaking Protocol in different NoC Topologies.

Fig. 19. Area overhead of runtime monitoring setup for different NoC Topologies.

Table 4
Power (μW) overhead of different runtime monitoring setups for partially connected mesh topology.

<table>
<thead>
<tr>
<th>Topology</th>
<th>4-connectivity (Mesh)</th>
<th>8-connectivity (Mesh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitors</td>
<td>N&lt;sub&gt;m&lt;/sub&gt;</td>
<td>CBRM</td>
</tr>
<tr>
<td>(FH-I)</td>
<td>2 21.5 21.5 21.5</td>
<td>2 21.5 21.5 21.5</td>
</tr>
<tr>
<td></td>
<td>4 70.5 77.5 51.5</td>
<td>4 101.75 83.1 77.39</td>
</tr>
<tr>
<td></td>
<td>9 152.5 92.37 77.5</td>
<td>9 298.68 112.35 100.32</td>
</tr>
<tr>
<td></td>
<td>16 304.25 113.25 102.5</td>
<td>16 395.5 170.81 140.25</td>
</tr>
<tr>
<td></td>
<td>25 613.5 290.56 190.5</td>
<td>25 650.17 390.25 275.84</td>
</tr>
<tr>
<td>(PH-I)</td>
<td>2 18.5 18.5 18.5</td>
<td>2 18.5 18.5 18.5</td>
</tr>
<tr>
<td></td>
<td>4 62.5 42.3 45.39</td>
<td>4 91.65 74.5 69.5</td>
</tr>
<tr>
<td></td>
<td>9 142.31 87.59 71.25</td>
<td>9 201.84 115.25 91.25</td>
</tr>
<tr>
<td></td>
<td>16 291.53 101.31 90.74</td>
<td>16 350.12 151.23 121.11</td>
</tr>
<tr>
<td></td>
<td>25 592.32 271.01 170.21</td>
<td>25 661.23 354.25 253.75</td>
</tr>
<tr>
<td></td>
<td>4 65.23 57.5 48.21</td>
<td>4 92.53 71.25 67.82</td>
</tr>
<tr>
<td></td>
<td>9 132.56 84.59 69.52</td>
<td>9 190.25 105.25 91.23</td>
</tr>
<tr>
<td></td>
<td>16 291.56 102.61 91.25</td>
<td>16 361.23 160.21 131.65</td>
</tr>
<tr>
<td></td>
<td>25 590.35 274.15 170.26</td>
<td>25 612.51 375.21 255.41</td>
</tr>
</tbody>
</table>

Table 5
Power (μW) overhead of different runtime monitoring setups for fully connected mesh and ring topology.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Ring</th>
<th>Fully connected mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitors</td>
<td>N&lt;sub&gt;m&lt;/sub&gt;</td>
<td>CBRM</td>
</tr>
<tr>
<td>(FH)</td>
<td>2 21.5 21.5</td>
<td>21.5</td>
</tr>
<tr>
<td></td>
<td>4 40.23 37.5</td>
<td>32.4</td>
</tr>
<tr>
<td></td>
<td>9 92.4 57.5</td>
<td>51.69</td>
</tr>
<tr>
<td></td>
<td>16 190.25 78.5</td>
<td>61.25</td>
</tr>
<tr>
<td></td>
<td>25 256.8 205.21</td>
<td>152.26</td>
</tr>
<tr>
<td>(PH-I)</td>
<td>2 18.5 18.5</td>
<td>18.5</td>
</tr>
<tr>
<td></td>
<td>4 35.6 31.26</td>
<td>29.65</td>
</tr>
<tr>
<td></td>
<td>9 81.2 50.13</td>
<td>47.59</td>
</tr>
<tr>
<td></td>
<td>16 172.81 63.12</td>
<td>54.24</td>
</tr>
<tr>
<td></td>
<td>25 241.32 180.26</td>
<td>141.23</td>
</tr>
<tr>
<td></td>
<td>4 35.62 31.25</td>
<td>28.21</td>
</tr>
<tr>
<td></td>
<td>9 85.2 51.36</td>
<td>47.59</td>
</tr>
<tr>
<td></td>
<td>16 175.25 65.21</td>
<td>52.69</td>
</tr>
<tr>
<td></td>
<td>25 221.36 192.36</td>
<td>141.25</td>
</tr>
</tbody>
</table>

Table 6
Test Time (data transaction cycles) overhead of runtime monitoring setup for different NoC topology.

<table>
<thead>
<tr>
<th>Topology</th>
<th>N&lt;sub&gt;m&lt;/sub&gt;</th>
<th>N&lt;sub&gt;ch&lt;/sub&gt;</th>
<th>ar</th>
<th>Test time overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitors</td>
<td></td>
<td></td>
<td></td>
<td>CBRM</td>
</tr>
<tr>
<td>Ring</td>
<td>2 1 1</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4 4 2</td>
<td>50</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>9 9 3</td>
<td>50</td>
<td>50</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td>16 16 4</td>
<td>50</td>
<td>50</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>25 25 5</td>
<td>50</td>
<td>50</td>
<td>1250</td>
</tr>
<tr>
<td>Fully Connected Mesh</td>
<td>2 1 1</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4 4 2</td>
<td>50</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>9 9 3</td>
<td>50</td>
<td>50</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>16 120 4</td>
<td>50</td>
<td>50</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td>25 300 5</td>
<td>50</td>
<td>50</td>
<td>6000</td>
</tr>
<tr>
<td>4-Connected Mesh</td>
<td>2 1 1</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4 4 2</td>
<td>50</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>9 9 3</td>
<td>50</td>
<td>50</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>16 25 4</td>
<td>50</td>
<td>50</td>
<td>11250</td>
</tr>
<tr>
<td></td>
<td>25 40 5</td>
<td>50</td>
<td>50</td>
<td>20000</td>
</tr>
<tr>
<td>8-Connected Mesh</td>
<td>2 1 1</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4 7 2</td>
<td>50</td>
<td>50</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td>9 21 3</td>
<td>50</td>
<td>50</td>
<td>1050</td>
</tr>
<tr>
<td></td>
<td>16 43 4</td>
<td>50</td>
<td>50</td>
<td>2150</td>
</tr>
<tr>
<td></td>
<td>25 72 5</td>
<td>50</td>
<td>50</td>
<td>3600</td>
</tr>
</tbody>
</table>

Fig. 20. Proposed Global Runtime Monitoring Unit with dynamic active number of Channels.
CBRM its power and area overhead is not dependent on the total number of communication channels and communicating modules. Therefore, the power overhead of this approach is considerably less than the previous case as shown in Fig. 18. Similarly its area overhead is up to 2 and 25 times less than RBRM and CBRM, respectively as shown in Fig. 19. However, its test time overhead is up to 5 and 30 times more than the RBRM and CBRM respectively, because in the runtime test mode, it scans all the channels one by one. Thus its test time overhead increases with the increase in number of channels and communicating modules as shown in Table 6.

RBRM approach is a compromise between high precision, high overhead CBRM and low precision, low overhead GRM. RBRM approach uses the multiple runtime monitoring units to get the higher precision upto 90% and lower test time overhead compared to GRM as shown in Figs. 18 and 19 and Tables 4–6. So it is generally a trade-off between performance overhead and efficiency of monitoring unit.

Table 7 shows the summary of the comparison with some of the state-of-the-art techniques. This comparative analysis is done based on

### 6.6. Dynamic runtime monitoring

The main objective of the proposed methodology is to monitor every communication channel to detect abnormal burst mode communications. We achieved this goal by introducing three topologies, i.e., channel based run-time monitor (CBRM), region based run-time monitor (RBRM) and global region monitor (GRM), to monitor the traffic on each channels. The CBRM setup provides 100% detection accuracy because it always monitors a particular channel. On the other hand, with the increase in the number of monitoring channels, the detection rate decreases, thus, the accuracy of RBRM and GRM decreases to 90% and 80%, respectively. However, in most of the cases, not all IP’s in a NoC fail at the same time. Therefore, we propose an alternative solution that includes the the first come first serve policy with monitors to handle the active channels and IP’s, as shown in Fig. 20. Since, the detection rate in the proposed methodology is inversely proportional to the number of monitoring channels per monitor, e.g., in our proposed methodology, the increase in the number of communication channels per monitor decreases the detection rate, as shown in Fig. 21a and b. The inclusion of first come first serve policy in monitors allows them to select number of monitoring channels during the runtime to increase the detection rate at the expense of increasing the power and area overhead.

For illustration, we implemented the first come first serve (FCFS) policy with multiple GRMs and the proposed RBRM for various number of channels per monitor (N/monitor). The experiment shows that detection rate remains around 85% for both FCFS based GRM and the proposed RBRM when N/monitor is less then 10. But, if (N/monitor) ≥ 10 then detection rate decreases exponentially for FCFS based GRM and linearly for the proposed RBRM, as shown in Fig. 21b. If we increase N/monitor to 18 then the detection rate decreases to 4% and 15% for FCFS based GRM and proposed RBRM, respectively. Similarly, the power overhead due to additional control blocks in FCFS based GRM is almost 10% more than the proposed RBRM as shown in Fig. 22. Moreover, in some cases, this power overhead approaches to 100 μW, which is almost 29% more than the proposed RBRM. Hence, for smaller number of modules and less complicated NoC, the suggested FCFS based GRM provides a comparable performance to the proposed RBRM. However, for more complicated NoC, RBRM seems to be the better solution because its power overhead is quite less.

### 7. Comparison with state-of-the-art techniques

Table 7 shows the summary of the comparison with some of the state-of-the-art techniques.

---

**Table 7**

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Detection Stage</th>
<th>Hardware overhead</th>
<th>Power</th>
<th>Test time</th>
<th>Trojan coverage</th>
<th>Golden IC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Solution</td>
<td>CBRM</td>
<td>Design, Testing and Runtime</td>
<td>N_ch × monitoring unit</td>
<td>N_A × PMU</td>
<td>T_eq</td>
<td>Communication based Trojans</td>
</tr>
<tr>
<td></td>
<td>GRM</td>
<td>1 monitoring unit</td>
<td>PMU</td>
<td>N_A × T_eq</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBRM</td>
<td>n × monitoring unit</td>
<td>n × PMU</td>
<td>n × T_eq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li et al. [18]</td>
<td>Design and Test</td>
<td>Ring Oscillator and Shadow Register</td>
<td>Low</td>
<td>High^b</td>
<td>Delay based Trojans</td>
<td>Yes</td>
</tr>
<tr>
<td>Lodhi et al. [8]</td>
<td>Design and Test</td>
<td>Controller and Vector insertion ports</td>
<td>Low</td>
<td>High</td>
<td>Delay based Trojans</td>
<td>Yes</td>
</tr>
<tr>
<td>Cao et al. [41]</td>
<td>Design and Test</td>
<td>Active current sensors</td>
<td>High</td>
<td>High</td>
<td>Delay based Trojans</td>
<td>Yes</td>
</tr>
<tr>
<td>Forte et al. [19]</td>
<td>Design, Testing and Runtime</td>
<td>Temperature Sensors &amp; Thermal profiling algorithm</td>
<td>High</td>
<td>Low</td>
<td>Only for the Trojans with high power consumption</td>
<td>No</td>
</tr>
<tr>
<td>Zhao et al. [21]</td>
<td>Design, Testing and Runtime</td>
<td>Temperature Sensors, Thermal profiling &amp; classification algorithm</td>
<td>High</td>
<td>Low</td>
<td>Require off line computation</td>
<td>No</td>
</tr>
<tr>
<td>Ngo et al. [22]</td>
<td>Design, Testing and Runtime</td>
<td>Hardware Property Checker</td>
<td>Almost Equal^c</td>
<td>Low</td>
<td>Only for Design Stage Intrusion</td>
<td>No</td>
</tr>
</tbody>
</table>

^a Lower than the proposed solution.  
^b Higher than the proposed solution.  
^c Equal to the proposed solution.
five parameters: Detection Stage indicates the stage at which a particular technique can detect the Trojan at which stage of the IC design. Hardware Overhead provides the extra components, which are required for a particular runtime monitoring technique. Similarly, Power Overhead provides the extra power consumption due to a particular runtime technique. Test Time is the time required to monitor and detect the Trojans by a particular technique. We compared the state-of-the-art techniques to the proposed runtime monitoring setup and the results are provided in Table 7, where P_{power} represents the power consumption of a proposed monitoring unit. The final parameter, in the last column of Table 7, is the requirement for Golden ICs, that means when the given technique requires the golden circuit to detect the Trojans during runtime or not. In a nutshell, our technique is better than other state-of-the-art techniques in following ways:

1. Our technique does not require golden circuit power traces, which is the case for many other alternatives [18,41].
2. Unlike [19], our technique does not presume that the Trojan pay activation will provide a very high change in the power dissipation.
3. In contrast to [21], our technique does not require any off line computation.
4. Our technique is applied at the SoC integration level and no knowledge of IP design is presumed.
5. Area, power and test time overhead is comparable especially with the case of RBRM.
6. The proposed technique can be used for any known bus system by adding the burst mode communication with the normal operational mode.

8. Conclusion

In this paper, we proposed a generic methodology to identify potential attacks and design the runtime monitors to detect the hardware Trojans through formal modeling of burst mode communication. The proposed approach leverages the formal hardware verification to analyze the behavior of the hardware Trojan and developed its unique properties utilizing LTL. Through the counter-examples obtained from Model checking, we identified the vulnerable paths due to hardware Trojan. Then, based on the analysis of these vulnerable paths, we proposed a method to design runtime monitors and three different approaches to embed them as monitors in the NoC. Furthermore, we also presented a methodology to utilize these burst mode runtime monitors in runtime intrusion detection through all communication channels. The statistical and experimental analysis shows that GRM approach has less area and power overhead as compared to RBRM and CBM approaches. The statistical and experimental analyses show that GRM approach has 2 and 25 times less power and area overhead as compared to RBRM and CBM approaches at the expense of 5–30 times increment in test time overhead. So it is generally a trade-off between the performance and efficiency of the monitoring unit. Unlike the other state-of-the-art techniques, the proposed methodology can easily be used to design the runtime monitoring setup without having netlist information of IP modules and any presumptions related to high power consumption of activated hardware Trojan.

References


