The ISC Int'l Journal of Information Security

ISeCure

January 2021, Volume 13, Number 1 (pp. 29–45)

http://www.isecure-journal.org

A Trusted Design Platform for Trojan Detection in FPGA Bitstreams Using Partial Reconfiguration

N. Shekofte [1], S. Bayat-Sarmadi [1,*], and H. Mosanaei-Boorani [1]

[1] *Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

ARTICLE INFO.

Article history:
Received: July 31, 2019
Revised: June 11, 2020
Accepted: November 1, 2020
Published Online: November 7, 2020

Keywords:
Hardware Trojan, Trusted Design Platform, Partial Reconfiguration, FPGA

Type: Research Article

doi: 10.22042/isecure.2020.196541.477

ABSTRACT

Hardware Trojans have emerged as a major concern for integrated circuits in recent years. As a result, detecting Trojans has become an important issue in critical applications, such as finance and health. The Trojan detection methods are mainly categorized into functional and side channel based ones. To increase the capability of both mentioned detection methods, one can increase the transition activity of the circuit. This paper proposes a trusted platform for detecting Trojans in FPGA bitstreams. The proposed methodology takes advantage of increased Trojan activation, caused by transition aware partitioning of the circuit. Meanwhile, it benefits partial reconfiguration feature of FPGAs to reduce area overhead. Experimental studies on the mapped version of s38417 ISCAS89 benchmark show that for the transition probability thresholds of 10^{-4} and 2×10^{-5}, our method increases the ratio of the number of transitions (TCTCR) in the Trojan circuit by about 290.93% and 131.48%, respectively, compared to the unpartitioned circuit. Similar experiments on s15850 for the transition probability thresholds of 10^{-4} and 2×10^{-5} show an increase of 290.26% and 203.11% in TCTCR, respectively. Furthermore, this method improves the functional Trojan detection capability due to a significant increase in the ratio of observing wrong results in primary outputs.

1 Introduction

Hardware security and trust have received significant attention in recent years [1, 2]. One way to ensure that an integrated circuit (IC) is trusted is to certify all its design and fabrication process. However, having a trusted supply chain is not economically feasible [3]. In fact, most companies are nowadays fabless and let the process of mask production, design fabrication and even integration be outsourced. These third-party fabs can thus tamper the circuits, which can alter the main functionality of ICs, leak their secure information and perform other malicious activities by some stealthy modifications [3–6]. These modifications are referred to as Hardware Trojans.

Nowadays, most modern companies use commercial-off-the-shelfs (COTS), intellectual property (IP) cores supplied by third-party vendors, and also field programmable gate arrays (FPGAs) to reduce the total cost and time to market [7–9]. The entire system is an integration of these units [1, 3]. Additionally, recent improvements of FPGAs in performance has considerably improved their usage as an alternative for ASICs. Due to increased complexity of modern devices, design-

* Corresponding author.
Email addresses: shekofte@ce.sharif.edu, sbayat@sharif.edu, mosanaei@ce.sharif.edu
ISSN: 2008-2045

ing FPGA IPs and integrating them with other FPGA and ASIC designs are usually performed in different companies. Thus, these designs are very vulnerable to Trojan insertion [10]. As a result, finding methodologies for detecting or preventing Hardware Trojan insertion in FPGA designs seems very necessary.

Hardware Trojan detection has become a challenge in recent years. Many Trojan detection approaches have been presented so far. They can be categorized into two main approaches based on side channel analysis and Trojan activation [11, 12]. Activation of the Trojan circuit can help Trojan detection by observing malfunction results in the functionality of the circuit. It can also accelerate Trojan detection further using power analysis due to consuming more dynamic power [13–15]. Side channel analysis approaches, on the other hand, detect Trojans by measuring some parameters of the circuits under test, such as power and delay. Then, these parameters are compared against the ones achieved from golden circuits [3, 7, 16].

In order to improve the efficiency of traditional Trojan detection methods, design-for-hardware-trust (DFHT) methodologies have been emerged [3, 17]. In [18], a trusted design based on ring oscillator network is proposed. Any switching in the Trojan circuit will cause a voltage drop in power supply. This can affect the delay of each stage in the circuit and in turn the RO (ring oscillator) frequency, which can then be used for Trojan detection. However, existence of a Trojan in one corner of the circuit may not cause the noise effect to be sensed in an RO placed on the other side. Hence, a network of ROs is distributed throughout the chip to improve the accuracy of this methodology.

In [10], a tamper detection method based on error detection codes in FPGA bitstreams is presented, considering that the FPGA itself has no Trojan. In this method, each row/column of CLBs, named as a parity group (PG), is assumed as a row/column of a 2D parity. To verify the bitstream, a test pattern generator (TPG) applies the same test vectors to all PGs. Then, an output response analyzer (ORA) checks whether the outputs are the ones expected. Any difference can indicate a Trojan in the bitstream. Two randomization methods are also offered for preventing adversaries from Trojan masking. In the first method, random CLBs from each row/column is selected as a row/column of the 2D parity. In the second method, the parity applied to each row/column is randomly chosen to be either even or odd.

In [19], spatial correlation of intra-die process variation in FPGA circuits is used to detect Trojans. In [11], a methodology is proposed to increase the transition probability in functional Trojan circuits. For this purpose, dummy scan flip-flops are inserted to increase the transition of nets with transition probability of less than a certain threshold. This method can be useful for improving detection methods based on power side channel analysis by increasing the transition probability (i.e. transient power) of the Trojan circuit and also by improving the chance to fully activate the Trojan, which may result in having wrong value at the primary output. The authors in [20] focus on dummy scan flip flops (DSFF) insertion, as an effective DFHT technique. They mention that DFHT approaches are almost ad-hoc techniques, which are vulnerable to neutralization efforts. However, they show that DSFF can be easily neutralized without recognizable side effects.

The authors of [21] propose a Trojan detection method for ASIC circuits. In this method, circuit under test (CUT) is divided into a number of partitions and generates the test vectors heuristically. Scan chains are being used to control these partitions. The method considers a power pin pad for each region separately to locally measure the subtle changes of the transition current. Such techniques usually impose considerable area overhead to the circuit due to scan chains and power pads. Similarly, [22] proposes a new technique based on scan flip-flops to detect Trojans. This method does not need any golden circuit and uses equal power self-authentication. This method imposes significant area overhead.

In [23], Trimberger addresses various issues in the protection of FPGA designs from tampering. This paper argues that FPGA configuration bitstreams are hard to reverse engineer to be tampered by an adversary. It also argues that the bitstream can be further protected by encryption. However, the circuit design can be required at an integration company that assembles. Moreover, bitstream reverse engineering is not completely impossible [23]. Authors of [24] mention that Triple-DES algorithm is used for bitstream encryption, which can be easily broken by side channel attacks. On the other hand, bitstream encryption imposes certain restrictions on FPGA usage, like disabling partial reconfiguration [23]. According to [25], bitstream encryption is only possible for new FPGA families, such as Virtex, while many old families are still widely used (Spartan family for example).

In [26], authors propose a built-in self authentication (BISA) method, which fills free spaces inside FPGA with dummy circuits. While the work imposes negligible area and power overhead, it is a method to prevent Trojan insertion and can be merged with our Trojan detection method. Similarly, [27, 28] propose Trojan insertion prevention methods in ASIC circuits. While these techniques claim no area overhead, they impose power overhead.

The work presented in [29] argues that functional

Trojan detection methods suffer from trigger circuit activation. In contrast, side-channel based ones can detect Trojans without fully activation of the trigger circuit. However, these methods are sensitive to process variation and environmental noises. Fortunately, our proposed method has the ability to perform both methods and consequently can gain the benefits of both.

In [30], a machine learning-based run-time method is proposed to detect hardware Trojans in microprocessor cores. In this method, a changepoint-based anomaly detection algorithm is used to detect Trojans that introduce abnormal patterns in the data streams obtained from performance counters. The authors in [31] propose an FPGA based Trojan detection and analysis method, which does not need the existence of a golden chip. The proposed method is a combination of a logic testing method, a run-time method, and a side-channel analysis one. Unlike the logic testing and side-channel analysis methods, the proposed run-time method is invasive, in which on-chip digital sensors are used to detect unexpected differentiation in the layout of the IC.

In this work, we have focused on detecting Trojans which are inserted in FPGA bitstreams. Two scenarios, which make bitstreams vulnerable to Trojan insertion, are presented. Both scenarios suffer the untrusted path from designer(s) to the user. One possible approach to authenticate such bitstreams is signature computation. Although the signature computation of the bitstream is possible, it has drawbacks. It should be noted that conditionally triggered Trojans can be activated by an external condition. For example, such condition could be based on the output of a sensor monitoring the temperature, or any kind of environmental condition, such as electromagnetic interference or humidity [3]. Signature computation, on the other hand, is an off-chip checking approach. Therefore, it cannot protect against remote attacks like high-energy ElectroMagnetic Pulses (EMP), malicious IPs, and tampers induced by the device programming unit [10]. The authors in [10] argue another drawback as an aliasing issue. However, we believe that by using cryptographically secure hash functions along with an asymmetric signature scheme, this problem can be avoided.

This paper proposes a trusted platform to detect Trojans in FPGA bitstreams. One of the mentioned threat scenarios and the proposed solution for encountering it is shown in Figure 1. The solution enjoys the increased observability and controllability of a partitioned circuit, which causes an increased transition probability in the nets. This can improve Trojan activation, and detection as a result, by increasing transition probability in the Trojan circuit. To make the partitioning more efficient, we use a *transition-aware partitioning*, which performs the partitioning such that all nets have transition probability greater than a special threshold. On the other hand, we gain partial reconfiguration feature of FPGA circuits to reduce the area overhead of this method. This is because while a partition is in the test mode, the resources of the other partitions are available to be used as control structures, including TPG and ORA.

Experimental results, performed on the mapped version of s38417 ISCAS89 benchmark [32], show that for the transition probability thresholds of 10^{-4} and 2×10^{-5}, this method increases the ratio of the number of transitions in the Trojan circuit (TCTCR) by about 290.93% and 131.48%, respectively, compared to the unpartitioned circuit. Similar experiments on s15850 for the transition probability thresholds of 10^{-4} and 2×10^{-5} show an increase of 290.26% and 203.11% in TCTCR, respectively. Additionally, the ratio of observing wrong results in primary outputs increases considerably, which could be useful for Trojan detection using functional methods.

The motivation of this work can be summarized as the following:

- The proposed platform makes it possible to use different types of detection methods, such as functional, power-based and delay-based methods. Moreover, it is possible to use a combination of these methods to improve Trojan detection further.
- This methodology can localize the partition in which the Trojan exists.
- This method improves Trojan detection because of partitioning and by using partial reconfiguration feature of FPGAs.
    - In the functional method, this improvement is a result of increased controllability and observability in the partitioned circuit, compared to those of the unpartitioned circuit.
    - In the power-based method, the improvement is achieved because of amplifying the dynamic power consumption of the Trojan against the total dynamic power.
- The end user can generate the test vectors (TPGs) and analyze the results (ORAs) herself. Hence, Trojans could not be inserted in them.
- The method introduces a small logic area overhead due to exploiting the partial reconfiguration feature of FPGAs.

The rest of this paper is organized as follows. Section 2 describes some preliminaries behind the proposed work. The proposed trusted design methodology

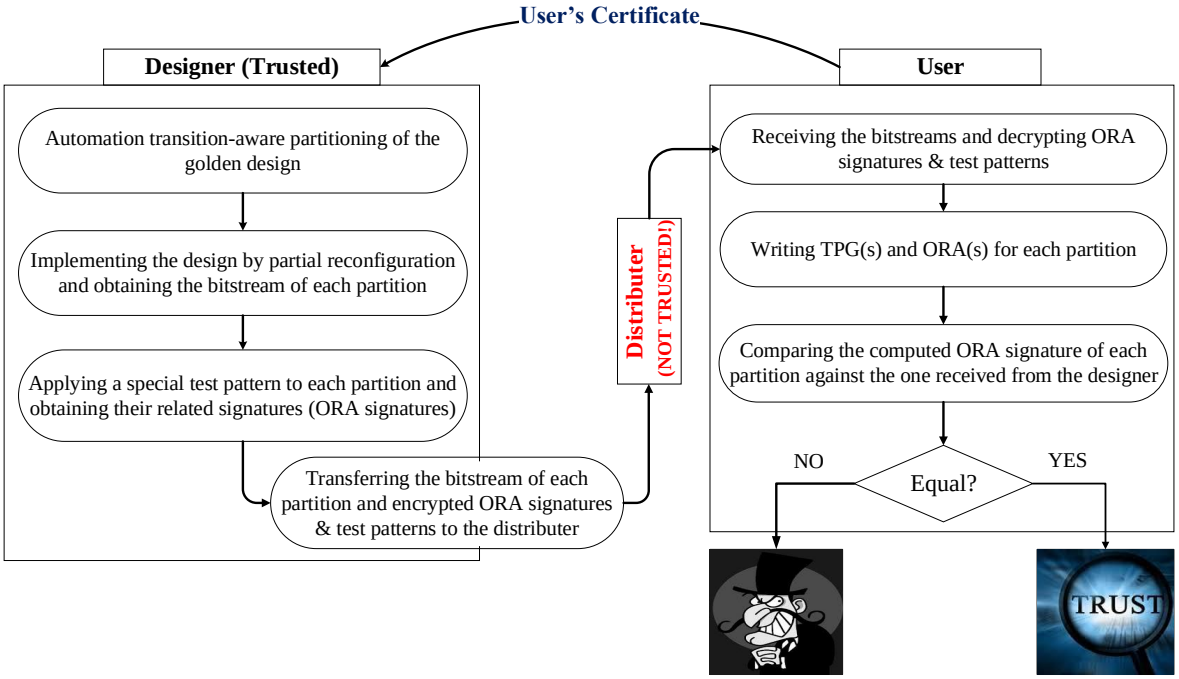


Figure 1. The Trojan detection scenario

is presented in Section 3. In Section 4, an authentication time analysis method along with the transition-aware partitioning are presented. Simulation results and an analysis of overheads are presented in Section 5. Finally, the concluding remarks are given in Section 6.

2 Preliminaries

In this section, the basic model of hardware Trojans and one of the most recent classifications of them are briefly overviewed. Then, a short explanation about partial reconfiguration is presented.

2.1 Hardware Trojan Structure and Taxonomy

The basic model of a hardware Trojan is made of two main components: payload and trigger [33]. Trigger is the part that activates the Trojan, and its effect carryovers to the payload [34, 36]. Trigger logic circuit can be either analog or digital. Digital trigger itself can be of two types: sequential and combinational. Combinationally triggered Trojans are activated by the occurrence of a special condition on the triggered nodes. Sequentially triggered Trojans, on the other hand, are the ones that result in an incorrect value of the payload by the occurrence of a sequence of operations on the trigger nodes.

There are various types of Trojans each of which is detectable by a group of detection methods. It is therefore very important to have a comprehensive classification of them. One of the most recent and detailed taxonomies is proposed by Wang *et al.* [3]. They classify Trojans into three main categories regarding their action, activation, and physical characteristics.

The physical characteristics category is classified into four subcategories: type, distribution, size, and structure. Trojans can be of either parametric or functional type. Parametric Trojans are realized through some changes in parametric characteristics of the circuit such as thickness of wires, while functional Trojans are related to some modifications in the functional characteristics of the circuit such as adding or deleting logic gates. Trojan distribution describes whether Trojans are located loose or tight throughout the layout. Trojan size refers to the amount of modification in gates or number of transistors, where Trojan structure determines whether the Trojan insertion is combined with changes in the layout of the circuit or not.

The activation characteristics category shows how the Trojan becomes activated. Trojans can be either activated internally or externally, e.g. by a sensor. Internally activated Trojans themselves can be either conditional, which are only triggered under a special internal condition, or always on. Action characteristics, on the other hand, refer to the destructive effect caused by the Trojan. Trojans may change the function of the circuit, change its specification or leak its information [35, 36].

2.2 Partial Reconfiguration

While FPGA technology provides the flexibility to be re-configured without any need of re-fabrication, partial reconfiguration takes this flexibility even further. This feature, which has been added to recent FPGA families, makes it possible to reconfigure only a partition of the design while the rest remains unchanged and performs its normal operation [37, 38].

In regular configuration of an FPGA, the device is assumed as an entity and one bitstream is generated. However, in the partial reconfiguration, the device is partitioned into some physical regions. One of the regions, named as the static region, configures at startup and cannot be re-configured any more. On the other hand, regions that can be configured multiple times and with different designs dynamically are known as dynamic regions. Therefore, in partial reconfiguration of an FPGA, there exist at least two bitstreams: one for the static region and one for each single design of each dynamic region [37]. Planahead software [39], which is installed with ISE design suite [39], makes it possible to locate the exact position of each dynamic region.

3 Proposed Trusted Design Platform

In this section, two scenarios with Trojan insertion vulnerabilities are explained. Then, the proposed solution for encountering them is presented.

Two vulnerable scenarios for trojan insertion:

• Scenario 1: According to the high complexity of recent technology products, the process of designing IPs related to FPGA circuits and integrating them with other IP designs are usually performed in separate companies. Therefore, while IP designers are not aware of this integration process, the untrusted integrators must have detailed IP designs. As a result, these products are very vulnerable to Trojan insertion [10].

• Scenario 2: Designs are usually sent to the customer through some distributors. Therefore, these designs are also vulnerable against malicious attacks caused by untrusted distributors.

The proposed solution: The proposed DFHT solution is based on using partial reconfiguration. The solutions for encountering each of the two vulnerable scenarios are explained as follows:

For the first scenario, each designer is supposed to apply a certain test pattern to her design and obtain a signature from the outputs (ORA Signature). The test pattern could be of various types, e.g. parametric or functional. Similarly, the corresponding signature could be of various types, e.g., a SHA-1 digest. On the other hand, the integrator is obligated to partially reconfigure the designs and generate bitstream for each one separately. These bitstreams are then passed to the user and she should apply the test patterns of the designers to the IPs/designs, obtain the ORA signatures and compare them with the ones received from the designers. Any difference can be assumed as a Trojan insertion. Note that test patterns and ORA signatures are transferred to the user authentically and confidentially (see Section 3.1).

It should be noted that designers are assumed to hand out their IP cores in placed and routed XDL format. In fact, there must be a two-way interaction between the integrator and the designers: the integrator should be informed about the size of each design; then she should inform the designers where each design will be placed (this is why there is a two-direction arrow between the integrator and each designer in Figure 2(a)). Finally, the designers can hand out their placed and routed XDL files to the integrator. This way, the possible intra-process variation could be negligible. Therefore, the influence of Trojans can be considered to be more dominant than such process variations. This scenario is illustrated in Figure 2(a).

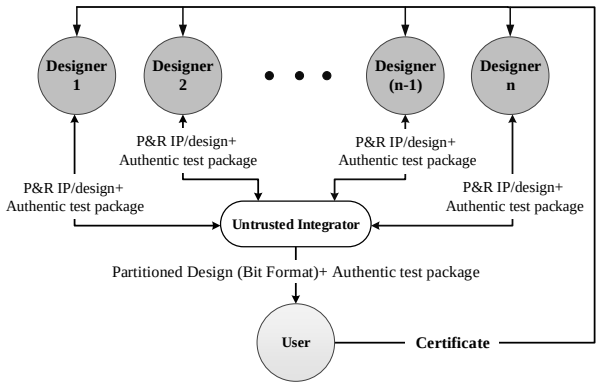
For the second scenario, as shown in Figure 2(b), the designer herself is expected to partition the design into some sub-designs and implement them using partial reconfiguration. In fact, she should generate one bitstream for each partition. Then, similar to the first scenario, the user should obtain the ORA signatures of partitions and compare them with the ones received from the designer. Any difference between the ORA signatures can be assumed as Trojan insertion. This scenario and the proposed solution are illustrated in Figure 1.

3.1 Secure Test Patterns/ORA Signatures Transfer

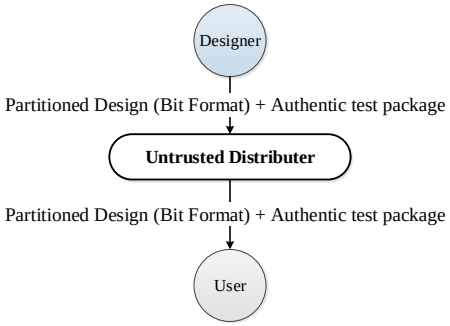
In this section, we investigate how a user can authentically obtain the required test patterns and ORA signatures for each part from its designer.

Suppose M contain test patterns along with ORA signatures. Algorithm 1 presents the actions that the designer has to perform to securely send M to the user. It is worth-mentioning that alternatively the designer can securely post M on her website; however, this approach can impose a considerable communication overhead on the user to obtain them separately.

According to the algorithm, we have used a symmetric key encryption (AES_ENC), a cryptographically secure hash function (SHA), and an asymmetric key encryption (RSA_ENC). It is worth-mentioning that the user's certificate has to be obtained authentically



(a) Scenario 1



(b) Scenario 2

Figure 2. Two vulnerable scenarios and their solutions

Algorithm 1 The Interaction Between the User and Each Designer - Designer Side

1: Authentically obtain user's certificate (containing the user public key, namely UPK)
2: Randomly generate a session key, namely SK
3: Calculate $C_{User} = AES\_ENC_{SK}(M,SHA(M))$
4: Calculate $Encaped\_SK = RSA\_ENC_{UPK}(SK)$
5: Transfer C_{User} and $Encaped\_SK$ to the distributor/integrator or post them on the website

Algorithm 2 The Interaction Between the User and Each Designer - User Side

1: Obtain C_{User} and $Encaped\_SK$ from the integrator/distributor or website
2: Calculate $SK = RSA\_DEC_{PrivateKey}(Encaped\_SK)$
3: Calculate $(M,SHA(M)) = AES\_DEC_{SK}(C_{User})$
4: Verify M against $SHA(M)$

from the user, e.g., directly from her or from her website. Clearly, this certificate contains the user's public key (UPK) required in step 4.

The required actions in user side are presented in Algorithm 2. We note that Figure 2 presents the case that the designer passes the authentic test package ($Encaped\_SK$ and C_{User}) to the integrator/distributor (and it is not the case that she posts the package on her website).

The process mentioned in Algorithm 1 and Algorithm 2 ensures that both confidentiality and data integrity of M are satisfied and integrator/distributor cannot alter or know about M. It should be noted that using this method, the designer of each part can customize the test patterns and their corresponding ORA signatures for each user. Such a customization avoids some attack scenarios such as using identical test patterns and ORA signatures to insert customized Trojans (in such scenarios, the attacker can pose herself as a user and obtain the test patterns and ORA signatures).

It is worth mentioning that Algorithm 1 and Algorithm 2 are actually simple versions of the SSL (TLS) protocol. In other words, TPG and ORA can be sent to the user over HTTPS protocol.

Implementing TPGs and ORAs: According to the previous section, in both scenarios, the TPGs and ORAs can be written by the user herself. For implementing such TPGs and ORAs, it is supposed that the following files are also given to the user by the distributor/integrator.

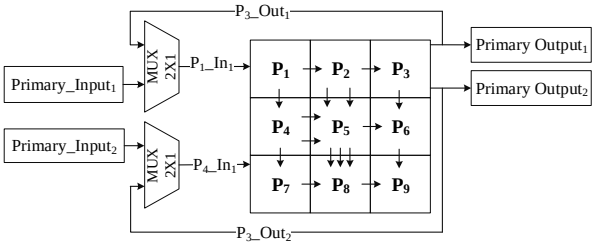
- The binary Xilinx netlist file of the static partition (*.ngc* file),
- The constraint file of the static partition, for locating each dynamic partition in its appropriate place (*.ucf* file),
- One Xconfig file, to locate the pins of each dynamic partition in its appropriate place.

It should be noted that using this method, TPG(s) and ORA(s) are applied to each partition by its adjacent partitions. In other words, it is not needed to have TPGs and ORAs outside the design. This makes the area overhead of the proposed method almost negligible, as is elaborated in Section 5.3. Moreover, one partition might have several TPGs and ORAs. For example, as shown in Figure 3(a) partition P_5 receives its inputs from partitions P_2 and P_4. It also sends its outputs to partitions P_6 and P_8. Then, while P_5 is the partition under test (PUT), P_2 and P_4 act as its TPGs and P_6 and P_8 act as its ORAs.

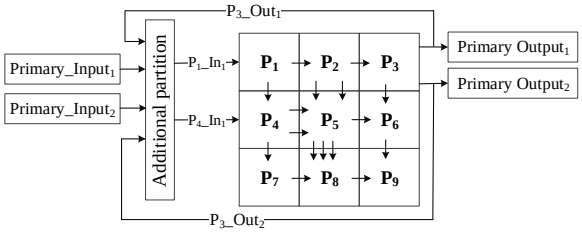
3.2 Platform Based Detection Method

The proposed method is a DFHT platform. Because it can be used for various types of Trojan detection. For example, it can be exploited as a functional platform, using TPGs and ORAs, taking advantage of the increased probability of Trojan activation. Moreover, it can be useful for detecting Trojans using power-based fingerprinting.

The power-based platform benefits from both increased controllability and localizing the measurement of transient power. In other words, as only one of the partitions (PUT) exists (excluding TPG and ORA) in the test mode, the effect of transient power of the



(a) Using multiplexers



(b) Using an extra partition

Figure 3. Analyzing a special case

Trojan circuit can be more observable. In fact, since other partitions do not exist in the test mode, the total number of transitions and in turn the entire dynamic power of the circuit are lower. Then, this effect could be sensed by an RO, which is partly placed in PUT. A simple RO is a device composed of an odd number of negating gates in a ring, whose output oscillates between two voltage levels, representing as true and false. Trojan insertion can affect the delay of the gates inside the ring and hence change the frequency of the RO. This is similar to the idea presented in [40], in which circuit paths of a design are reconfigured to act as an RO. The rest of the required circuit remaining from RO is placed in its adjacent partitions (see Figure 4). In this case, as shown in Figure 4, the adjacent partitions of the PUT act as its controller. The controller consists of an oscillation cycle counter and a comparator, for comparing the signatures (similar to the ORA in the functional platform).

It should be mentioned that the RO circuit must consist of an odd number of inverting logic gates. Therefore, the complementary part must contain an odd number of inverter gates, if the number of inverting logic gates in the PUT is an even number. Additionally, the controller part should apply test patterns in such a way that the RO gets established and works properly; e.g., in Figure 4, there are two gates in PUT that establish the RO. Hence, the second inputs of the NAND and NOR gates have to be "one" and "zero", respectively.

It is worth mentioning that in the case of misusing unused I/Os, some side-channel characteristics will be changed. Regardless a pin is being used or not, we can always verify its value for different test vectors.

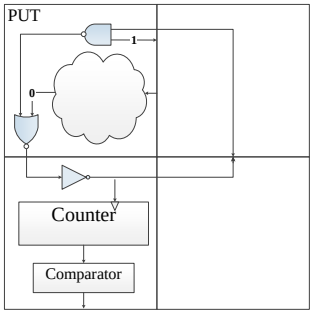


Figure 4. A simple example of using an RO as ORA

Therefore, it could be detected by either functional or side-channel analysis Trojan detection.

3.3 Analyzing a Special Case

For applying test pattern to or receiving response from the partitions with primary inputs/outputs, only the partitions which include primary outputs/inputs could be used. Clearly, a primary input cannot be fed by two nets, simultaneously. Therefore, it should be specified whether it is connected to a primary output, receiving test patterns, or acts as a primary input, receiving its data from outside of the device. To resolve this issue, two approaches are suggested:

- To use multiplexers: In this method, multiplexers are placed on the way of such inputs. The select input of multiplexers is set according to the working mode (normal or test modes). Figure 3(a) illustrates this method in which a 2x1 multiplexer is used to choose between two different input sources. One input source can be the primary output net $P_3\_Out_1$ from partition P_3 (as TPG of P_1) and the other one can be $Primary\_Input_1$ is connected to the input $P_1\_In_1$ of partition P_1 (as PUT). Thus, using this multiplexer, partition P_3 can act as a TPG of P_1 and similarly P_1 can be used as ORA for partition P_3. Likewise, the other multiplexer is supposed to choose whether $P_3\_Out_2$ from partition P_3 (as TPG of P_4) or $Primary\_Input_2$ is connected to the input $P_4\_In_1$ of partition P_4 (as PUT).
- To use an extra partition: In this method, an extra partition is used instead of all multiplexers. Then, according to the working mode (test or normal), a specific bitstream is loaded into this partition. For example, as shown inFigure 3(b), the bitstream loaded into the additional partition is responsible to choose whether $P_3\_Out_1$ or $Primary\_Input_1$ is connected to the input $P_1\_In_1$. It also chooses whether $P_3\_Out_2$ or $Primary\_Input_2$ is connected to the input $P_4\_In_1$.

Comparing the two approaches: Assuming $PICount(P_i)$ is the number of primary inputs in partition number i and $PartitionCount$ is the number of partitions; then, the number of multiplexers needed in the first method can be computed from the following equation:

$$\#Multiplexers = \sum_{i=0}^{PartitionCount} PICount(P_i)$$

In the second method, as it was explained, the additional partition is only responsible to connect some nets to each other. This method thus has no logic area overhead. However, it imposes a slight routing overhead because of using switch matrices for the connections. This is briefly discussed in Section 5.3.

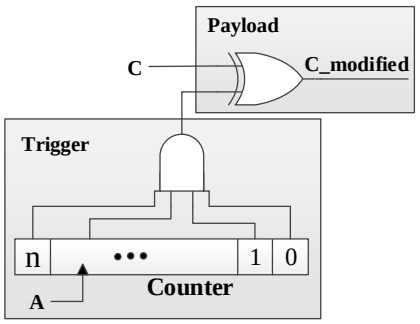
3.4 Automated Partitioning

As partitioning a large circuit with significant number of wires is very difficult to be performed manually (in case of the second scenario), automated partitioning seems to be necessary. For this purpose, we have used *hMetis* [41]. This is a software package capable of partitioning the vertices of a hypergraph into k approximately equal parts in a way that the number of hyperedges, which are the connections between partitions, is minimized [42]. Here a hyperedge is described as an extension of an edge, by which more than two vertices are connected to each other. Thus, for automation, we first generated the netlist of a Verilog code using ISE. Then a *C++* code was written for converting the netlist into the input format of *hMetis* (a hypergraph). Afterward, we used *hMetis* for partitioning the hypergraph. Finally, another *C++* code was written and used for converting these partitions, which are in graph format, into Verilog codes.

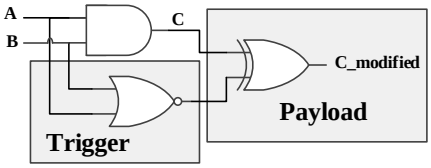
4 Authentication Time Analysis

An adversary inserts the Trojans such that they are being triggered only under very rare conditions, to avoid detecting them in the test mode [43–45]. For this purpose, and for conditionally triggered Trojans, Trojan triggers should either be nets with very low transition probability (sequentially triggered) or a rare combination of nets (combinationally triggered) [11, 45]. Schematics of such Trojan circuits are shown in Figure 5. Analyzing the Trojan activation time of a design is very important, especially when running in the test mode. If a Trojan is sequentially triggered, its transition probability can be a proper estimation of its transition time [11]. Transition probability of a net is defined as the probability of switching from 0 to 1, which is also equal to the probability of switching from 1 to 0. Suppose Pb_i0 and Pb_i1 are the probabilities for net i to be 0 and 1, respectively. Then, transition probability of the net will be $Pb_i0 \times Pb_i1$ as is computed in Equation (1).

$$\begin{aligned} Pb_{i0\to1} &= Pb_i0 \times (Pb_i1 \mid the\ net\ value\ has\ been\ 0) \\ &= Pb_i0 \times Pb_i1 = Pb_i1 \times Pb_i0 \\ &= Pb_i1 \times (Pb_i0 \mid the\ net\ value\ has\ been\ 1) \\ &= Pb_{i1\to0} = Pb_{itransition} \end{aligned} \tag{1}$$



(a) Sequentially internally-triggered Trojan



(b) Combinationally internally-triggered Trojan

Figure 5. Trojan examples [1]

Even in the case of combinationally triggered Trojans, transition probability can be useful for estimating the Trojan trigger time. In this case, the Trojan is triggered when a special combination of a subset of nets (i.e. trigger inputs) occurs. With n number of trigger inputs, the probability of Trojan triggering is as presented in Equation (2) [11, 46]:

$$Pb_{Trojan-Trig} = \prod_{i=1}^{n} \{Pb_i0\ or\ Pb_i1\} \tag{2}$$

According to Equation (1), transition probability of a net i is very low if $Pb_i1 >> Pb_i0$ or $Pb_i0 >> Pb_i1$. Additionally, $Pb_{Trojan-Trig}$ in Equation (2) is expected to be very low if $Pb_i1 >> Pb_i0$ or $Pb_i0 >> Pb_i1$. It should be noted that the result obtained from Equation (2) is non-deterministic. Because either sides of the *or* operation might be selected, according to the structure of the Trojan circuit, which obviously is not determined before being detected. In other words, Equation (2) depends on the Trojan circuit and also the nets that are used as inputs of that circuit. It is noted that the mentioned circuit can be a combination

of gates or one or more LUTs. We would like to have Pb_i0 and Pb_i1 such that selecting either does not result in low value of the $Pb_{Trojan-Trig}$, which clearly means low Trojan detection capability. Therefore, for increasing $Pb_{Trojan-Trig}$, we need to make the values of Pb_i0 and Pb_i1 close to each other. In other words, to improve Trojan detection for net i, we should reduce the Pb_{iTj} value of nets in Equation (3):

$$Pb_{iTj} = |Pb_i0 - Pb_i1| \tag{3}$$

This concept is even more important for the nets with large Pb_{iTj} that are suspicious to act as inputs of the Trojan circuit.

According to Equation (4), we can show that increasing $Pb_{itransition}$ and decreasing Pb_{iTj} are proportional:

$$\begin{aligned} Pb_{itransition} &= Pb_i1 \times Pb_i0 \\ &= \frac{(Pb_i0 + Pb_i1)^2 - (Pb_i0 - Pb_i1)^2}{4} \\ &= \frac{1 - |Pb_i0 - Pb_i1|^2}{4} = \frac{1 - {Pb_{iTj}}^2}{4} \end{aligned} \tag{4}$$

Therefore, it could be concluded that, in both cases (i.e. sequentially and combinationally triggered Trojans), detection probability will improve by increasing $Pb_{itransition}$.

4.1 Calculating Transition Probability

For calculating $Pb0$ and $Pb1$ of every net in an FPGA resource level circuit, we have developed a tool referred to as *FPGA_TPC* (FPGA Transition Probability Calculator). We note that we have inspired from TPC tool in Trust-hub [47] (It is noted that TPC was a tcl script written for calculating transition probability of nets within an ASIC circuit, which was not suitable in our case). The *FPGA_TPC* algorithm is presented in Algorithm 3. According to this algorithm, $Pb0$ and $Pb1$ for inputs of each DFF and primary input are initialize with 0.5. Then the probability of each cell output is computed according to the probabilities of its inputs and the cell type. As it is mentioned in the algorithm, for each cell, the probability of its cone (i.e. the logic circuit connecting to the cell inputs) is calculated earlier.

To clarify, an example of applying *FPGA_TPC* to a simple sequential circuit ($s27$) is shown in Figure 6. In this figure, $TP1$, $TP2$, and $TP3$ show the transition probability of nets after one, two and three clock cycles, respectively. As there may exist a number of DFFs from a primary input to a net, the transition probability of nets will be changed by increasing clock cycle count. Moreover, it is noted that DFFs usually have feedbacks. So, transition probabilities may never become a fixed value. Therefore the numebr of clock cycles in the algorithm should be selected according to the precision which is needed. In our experiments, we have chosen this number to be at least the maximum number of DFFs from primary inputs to primary outputs. Therefore, the required clock cycles for this example is 3.

Algorithm 3 Transition Probability Calculation Algorithm

```
1: procedure FPGA_TPC
2:     Initialize nClock with the number of clocks
3:     Initialize cellsQueue with null
4:     for each cell do
5:         initialize its nInputs with its number of input pins
6:         initialize its nUpdatedInputs (number of updated inputs of cell) with 0
7:     end for
8:     for each primary input do
9:         initialize its Pb0 and Pb1 with 0.5
10:    end for
11:    for each cell connected to primary inputs do
12:        cell.nUpdatedInputs++
13:        if cell.nUpdatedInputs==cell.nInputs then
14:            cellsQueue.Add(cell)
15:        end if
16:    end for
17:    for each input net of flip flops do
18:        initialize its probability with 0.5
19:    end for
20:    for each GND net do
21:        set its Pb1 with 0
22:    end for
23:    for each VCC net do
24:        set its Pb1 with 1
25:    end for
26:    while (nClocks > 0) do
27:        for each flip flop do
28:            set its output net probability with its inputs net probability
29:            for each cell which is connected to flip flop output net do
30:                cell.nUpdatedInputs++
31:                if cell.nUpdatedInputs==cell.nInputs then
32:                    cellsQueue.Add(cell)
33:                endif
34:            end for
35:        end for
36:        for each cell in cellsQueue except flip flops do
37:            set its output net probability with the calculated probability //Calculated with lib.tcl
38:            for each connected-cell which is connected to output net of the cell do
39:                connected-cell.nUpdatedInputs++
40:                if connected-cell.nUpdatedInputs == connected-cell.nInputs then
41:                    cellsQueue.Add(connected-cell)
42:                end if
43:            end for
44:        end for
45:        nClocks−−
46:    end while
47: end procedure
```

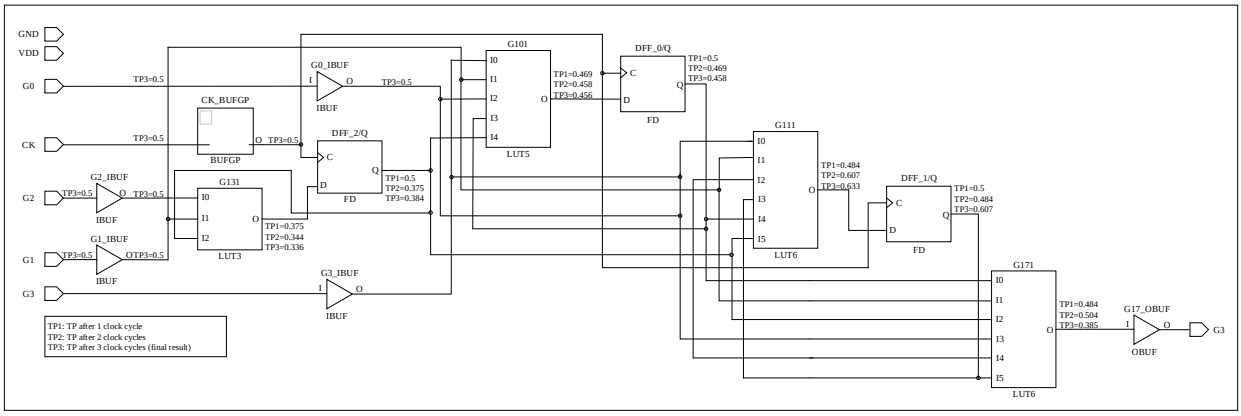


Figure 6. An example of transition probabilities resulted by *FPGA_TPC* program

4.2 Transition-aware Partitioning

As it was mentioned in Section 4, transition probability of a net is a proper estimation for its transition time. Actually, the average number of clock cycles needed to generate a transition in a net could be computed from Equation (5) [11]:

$$Avg\,\#\,clk\_cycles \;=\; Pb_{itransition}^{-1} - 1 \qquad (5)$$

According to Equation (5), increasing $Pb_{itransition}$ in a net will decrease the average time to generate a transition in a net exponentially, which can considerably enhance the probability of Trojan activation in the test mode, as a result. We can define a rare event as the one with a transition probability of less than a special threshold (p_{th}). Then, if all nets of a circuit have transition probability greater than or equal to p_{th}, we can make sure, with a high probability, that the Trojan would be activated in the test mode (if there is any) [11]. Hereafter, we refer to such circuits as ideal ones.

According to the above-mentioned, in an ideal circuit, ${P_{th}}^{-1} - 1$ clock cycles, on average, will be required for each transition. Suppose N_{tr} is the required number of transitions in Trojan circuit for triggering the Trojan. Then, assuming T_{Tester} to be the period of the clock, authentication time of circuit will be computed by Equation (6)[11]:

$$T_{Au} = N_{tr} \times ({P_{th}}^{-1} - 1) \times T_{Tester} \qquad (6)$$

In the proposed method, the input nets of each partition obtain their values directly from TPG(s), in the test mode. Therefore, such nets have $Pb0 = Pb1 = 0.5$ and in turn $Pb_{transition} = 0.25$. Hence, it is expected that the internal nets of each partition have also a $Pb_{transition}$ closer to 0.25 on average, compared to those in an unpartitioned circuit. In other words, partitioning the circuit helps to reduce the number of rare event nets by increasing the overall transition probability.

For having an ideal partitioning in the second scenario, we should ensure that all partitions are ideal. A simple way for this purpose is to start from two partitions and increase the number of partitions until we achieve an ideal partitioning. Clearly, as the number of partitions increases, the number of interface pins (i.e. pins as interface between partitions) also increases. Therefore, the probability of routing congestion increases too. It is thus important to perform partitioning in an efficient way to meet the transition probability threshold constraint, while the number of partitions is minimized. Here we present a transition-aware partitioning, which halves each non-ideal partition of the circuit till it reaches to an ideal partitioning. This partitioning methodology is presented in Algorithm 4 with more details.

We note that, in this algorithm, we have used automated partitioning (hMetis) tools to split a partition into two partitions. hMetis, as described in Section 3.4, performs partitioning such that the number of hyperedges (interface pins) between two partitions becomes minimized. This is important as we use the adjacent partitions of each partition as its TPG/ORAs. Afterward, this will reduce the probability of routing congestion even further.

5 Results and Evaluations

For analyzing the proposed method, we applied it to two ISCAS89 benchmarks: I) s38417, with 28 inputs, 106 outputs, and 1636 D-type flip-flops, and II) s15850 with 77 inputs, 150 outputs, and 534 D-type flip-flops. We performed transition-aware partitioning (described in Section 4.2) using *FPGA_TPC* program and the automated partitioning explained in Section 3.4. We first synthesized and mapped the designs into Virtex6 ML605 FPGA using ISE design suite for the sake of FPGA-level simulation Then we used these

Algorithm 4 Transition Aware Partitioning Algorithm

Input: MyCircuit.v (the unpartitioned circuit)
Output: MyQueue (list of all partitions of MyCircuit.v, making an ideal partitioning).

```
initialize S and MyQueue with empty queues;
initialize MyPartition with a partition;
add MyCircuit.v to S;
while S is not empty do
    MyPartition = S.dequeue();
    if MyPartition is ideal then
        MyQueue.enqueue(MyPartition);
    else
        split MyPartition into partitions MyPartition_1
          and MyPartition_2, using automated partitioning
          described in Section 3.4;
        S.enqueue(MyPartition_1);
        S.enqueue(MyPartition_2);
    end if
end while
return MyQueue
```

mapped designs as the input of $FPGA\_TPC$ script, which was run in Planahead software.

Experiments were performed for two transition probability thresholds: $I)\ P_{th1} = 10^{-4}$ and $II)\ P_{th2} = 2 \times 10^{-5}$. Then, three small Trojans were inserted to both unpartitioned and partitioned circuits. For inserting the Trojans more realistically, nets with the least transition probability were used as the inputs of Trojan trigger circuits.

Trojans can be activated partially or fully. Partial Trojan activation does not cause the payload output to become wrong; however, there are some transitions in the Trojan circuit [11]. Partial activation of Trojan can accelerate Trojan detection using side channel analysis methods, such as delay and power analysis. Full activation of Trojan can help even further by increasing the probability of observing wrong results on primary outputs.

According to the above-mentioned explanations, two parameters are defined for evaluating the results:

- Wrong primary output ratio ($WPOR$): This parameter could be considered as a factor of Trojan detectability using functional analysis methods and is defined in Equation (7). In this equation, "$\#\ of\ test\ vectors\ resulting\ WPO$" and "$\#\ of\ clock\ cycles\ observing\ WPO$" refer to "the number of test vectors resulting in wrong primary outputs" and "the number of clock cycles resulting wrong primary outputs", respectively:

$$WPOR = \frac{\#\ of\ test\ vectors\ resulting\ WPO}{the\ entire\ number\ of\ test\ vectors} = \frac{\#\ of\ clock\ cycles\ observing\ WPO}{\sum\ simulation\ cycles\ of\ all\ partitions} \tag{7}$$

- Trojan circuit TC ratio ($TCTCR$): This parameter, defined in Equation (8), is related to partial Trojan activation. Therefore, it could be noticed as a factor of Trojan detectability using power analysis methods. In Equation (8), $TC\ of\ a\ circuit$ refers to the sum of transition counts of all nets in the circuit during the simulation time. Additionally, Sim_Cycle and $Avg\_TCofCircuit$ refer to the simulation clock cycles, and the average transition count of the circuit per clock cycle, respectively.

$$TCTCR = \frac{TC\ of\ Trojan\ circuit}{TC\ of\ the\ entire\ circuit} \simeq \frac{TC\ of\ Trojan\ circuit}{Sim\_Cycle \times Avg\_TCofCircuit} \tag{8}$$

It should be noted that $Avg\_TCofCircuit$ is assumed as a fixed number for large values of simulation cycles. Therefore, for a long duration of simulation time and for making comparison between the results simpler, we could ignore this parameter. In fact, instead of Equation (8), we can define $TCTCR$ as the following:

$$TCTCR = \frac{TC\ of\ Trojan\ circuit}{Sim\_Cycle} \tag{9}$$

In the following, it is shown that the proposed method increases both partial and full activation of Trojans. Moreover, it is shown that in the case of full Trojan activation, the probability of observing wrong results on primary outputs increases significantly compared to that of the unpartitioned circuit.

5.1 Unpartitioned (Original) Circuit

Simulations were run for 400,000 clock cycles for each benchmark. One random test vector was applied to the circuits per cycle. Table 1 shows the results of each Trojan activity in the unpartitioned circuits. Rows 1 and 2 of the table, referred to as In1 TC and In2 TC, respectively, show the transition count of the Trojan trigger inputs. TC of trigger output is shown in row 3. Row 4 of Table 1 shows *Trojan circuit TC ratio*, described earlier. All these numbers were obtained from the *switching activity interface format (saif) of the design, generated by ISE design suite [39]*. The number of observed wrong results in the payload output and primary outputs of the circuits are referred to as "wrong payload output count" and "wrong primary output count," which are shown in rows 5 and 6 of the table, respectively. In fact, "wrong primary output count" takes "wrong payload output count" one step further by making Trojan effect to be observable even on the primary outputs of the

circuits. Finally, *wrong primary output ratio (WPOR)* is shown in row 7 of the table for making the results more sensible.

5.2 Partitioned Circuit

The given results after transition aware partitioning for two different transition probabilities are explained as follows:

Partitioned circuit with P_{th1}: After running *FPGA_TPC* on the circuits, 24 nets in s38417 and 26 ones in s15850 circuits were found with a transition probability of lower than p_{th1}. Using automated partitioning and according to transition aware partitioning, it was observed that 11 partitions in s38417 and 10 ones in s15850 circuit are needed to have no rare event nets left in the partitions. The partitioning related to s38417 is shown in Figure 7. In this figure, partition number 1 is obtained by splitting the circuit twice. Partitions 2 and 3 are achieved after splitting the circuit for 3 times, while partitions 4 to 11 are obtained by four times splitting the circuit. It is worth mentioning that we stop splitting a partition once the transition probability of all nets in the partition becomes larger than P_{th1}.

Simulations were run for $\frac{400{,}000}{11}$ cycles in s38417, and $\frac{400{,}000}{10}$ ones in s15850 circuit to make the results fairly comparable with the ones obtained before partitioning. In fact, all partitions were assumed to be run for this number of clock cycles. This way, the entire number of clock cycles (i.e. simulation time), for each circuit, would remain to be 400,000 cycles. As the simulation results in Table 2 show, for all of the three Trojans, In1 TC, In2 TC and trigger output TC of each circuit increased significantly, compared to the ones in Table 1. Additionally, $TCTCR$ (row 4) for circuits s38417 and s15850 increased by about 290.93% and 290.26% compared to the unpartitioned circuit, respectively. $TCTCR$ calulcations related to circuit s38417 is discussed below:

- Average of $TCTCR$ for unpartitioned circuit $= \frac{0.236+4.125e-4+0.587}{3} \simeq 0.27447$,
- Average of $TCTCR$ for partitioned circuit (P_{th1}) $= \frac{0.885+0.898+1.436}{3} = 1.073$,
- Increased average of $TCTCR$ (unpartitioned against partitioned circuit with p_{th1}) $\simeq 100 * \frac{1.073-0.27447}{0.27447} = 290.93\%$.

It should be noted that $TCTCR$ in row 4 is computed assuming a fixed value for $Avg\_TCofCircuit$ in Equation (8). However, while testing a partition

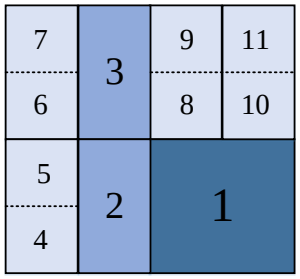


Figure 7. s38417 benchmark after partitioning with p_{th1}

in the partitioned circuit, only PUT, TPG and ORA partitions are running (having activities). Therefore, $Avg\_TCofCircuit$ is expected to be smaller while testing a partition, compared to testing an unpartitioned circuit. In other words, the numbers in row 4 of Table 2 are expected to be even larger, i.e., the power based Trojan detection capability is better than what is reported in Table 2. However, for simplicity, $Avg\_TCofCircuit$ is assumed identical for both cases (before and after partitioning).

The numbers in rows 5 to 7 of Table 2 increased significantly, which shows a considerable improvement in Trojan detection using functional methods.

Partitioned circuit with P_{th2}: After running *FPGA_TPC* on the circuits, it was observed that 13 nets in s38417 and 16 ones in s15850 circuit have a transition probability of less than p_{th2}. It was also observed that 7 partitions in s38417 and 8 ones in s15850 circuit are needed to remove all rare event nets. Therefore, simulations for circuits s38417 and s15850 were run for $\frac{400{,}000}{7}$ and $\frac{400{,}000}{8}$ cycles, respectively. Simulation results in Table 3 show a considerable increase in the results compared to Table 1. Moreover, $TCTCR$ for circuits s38417 and s15850 increased by about 131.48% and 203.11%, respectively. $TCTCR$ calulcations for circuit s38417 is elaborated below:

- Average of $TCTCR$ for partitioned circuit (P_{th2}) $= \frac{0.622+0.751+0.533}{3} \simeq 0.6353333$,
- Increased average of $TCTCR$ (unpartitioned against partitioned circuit with p_{th2}) $\simeq 100 * \frac{0.6353333-0.27447}{0.27447} = 131.48\%$.

Moreover, comparing the results with those in Table 2 shows that considering p_{th1} as the threshold value, Trojan detectability is higher; this is expected and is because $p_{th1} > p_{th2}$ (see Section 5.3 for more details).

Table 1. Trojan Activity in The Unpartitioned Circuit

		s38417			s15850		
		Trojan 1	Trojan 2	Trojan 3	Trojan 1	Trojan 2	Trojan 3
1	In1 TC	1	2	3	3	4	11
2	In2 TC	3	2	1	10	5	4
3	Trigger output TC	0	3	1	3	7	3
4	$TCTCR$	0.236	4.125e-4	0.587	0.254	3.552e-3	0.268
5	Wrong payload output count	9	1	2	11	3	7
6	Wrong primary output count	7	1	0	8	2	8
7	$WPOR$	1.75e-5	2.5e-6	0	2e-5	5e-6	2e-5

Table 2. Trojan Activity After Partitioning with p_{th1}

		s38417			s15850		
		Trojan 1	Trojan 2	Trojan 3	Trojan 1	Trojan 2	Trojan 3
1	In1 TC	301	8503	15081	925	13585	8420
2	In2 TC	16024	4010	204	14551	6224	724
3	Trigger output TC	16057	12322	1005	14671	18293	6773
4	$TCTCR$	0.885	0.898	1.436	0.525	0.743	0.783
5	Wrong payload output count	16241	1184	1026	10026	1535	968
6	Wrong primary output count	16205	295	8912	8231	1386	1554
7	$WPOR$	0.041	7.375e-4	0.022	2.058e-2	3.465e-3	3.885e-3

Table 3. Trojan Activity After Partitioning with p_{th2}

		s38417			s15850		
		Trojan 1	Trojan 2	Trojan 3	Trojan 1	Trojan 2	Trojan 3
1	In1 TC	766	5661	1890	1021	7657	7346
2	In2 TC	2451	6136	613	9462	9529	632
3	Trigger output TC	4102	12215	125	10703	12358	5281
4	$TCTCR$	0.622	0.751	0.533	0.522	0.603	0.468
5	Wrong payload output count	2317	94	1056	2023	843	612
6	Wrong primary output count	2001	3	4981	4022	872	1228
7	$WPOR$	0.005	7.5-6	0.012	1.005e-2	2.18e-3	3.07e-3

5.3 Evaluating the Results

As the results show, the proposed method increases Trojan activation considerably; which improves Trojan detection using both functional and side channel analysis methods. As it was mentioned in Section 3, the methodology can be used as a platform for different Trojan detection methods. It should be noted that different types of Trojans could be detected by different detection approaches. Therefore, a combination of such approaches can be applied together, to improve Trojan detectability even further.

To evaluate the method, Trojans are inserted inside the partitions; clearly, the proposed method can detect a Trojan in the glue logic (i.e. the logic between

partitions) even easier. This is because we are using each partition as TPG/ORA of its adjacent partitions. Therefore, any Trojan inserted in the glue logic will directly change the logic or have side channel effects on the primary inputs/outputs of the partition, which makes Trojan effect more considerable (and its detection easier). It should also be considered that the methodology determines the location of the Trojan to some extent, according to the location of the tampered partition.

We note that, in the studied scenarios, the integrators must have detailed information about IP designs [10]. Therefore, bitstream encryption could not be useful. Furthermore, bitstream encryption methods are mentioned as Trojan prevention method, while our purpose in this paper is to detect Trojans.

Overheads: As mentioned earlier, TPGs and ORAs are applied to each partition by the resources of the other partitions. Therefore, they have theoretically no area overhead. Moreover for applying TPG/ORA to the *primary inputs/outputs*, we could use the second method described in Section 3.3 (an additional partition), which does not even suffer the logic area overhead of multiplexers. Furthermore, as mentioned in Section 4.2, transition-aware partitioning is used to avoid the possible routing congestion. This congestion may be caused by I) extra switch matrices of the additional partitions discussed in Section 3.3, or II) increased number of pins due to partitioning. We note that *hMetis* software performs partitioning such that the number of interface pins becomes minimized (see Section 3.4). Our experiments, performed on a number of circuits from ISCAS89 bechmark [32], confirmed no routing congestion.

We also note that, for the first scenario, small margins among the IPs/designs are needed to assure that they won't have conflict with each other. Thus, some hardware area overhead cannot be denied. It should be mentioned that such overhead does not need to be considered for the second scenario.

It is worth mentioning that increasing p_{th} means a harder constraint on the transition probability of all nets. This makes the probability of Trojan activation, and in turn detection capability, increase. In other words, by increasing p_{th}, time to detect a Trojan decreases. On the other hand, by increasing p_{th}, the number of partitions that are required to satisfy this threshold increases. Therefore, the probability of routing congestion increases too. In fact, increasing p_{th} causes increased routing congestion and decreased Trojan detection time. Clearly, this detection time depends on the time that user can spend on testing the circuit.

6 Conclusion

In this paper, a platform for detecting Trojans in FPGA bitstreams has been presented. This method improves Trojan detection as a result of increased controllability and observability of a transition aware partitioning. Additionally, partial reconfiguration feature of FPGAs has made it possible to use partitions other than the partition under test as controller (TPG and ORA). This has made logic area overhead of the method almost negligible. Experimental studies on the mapped version of s38417 ISCAS89 benchmark show that for the transition probability thresholds of 10^{-4} and 2×10^{-5}, this method increases the ratio of the number of transitions (TCTCR) in the Trojan circuit by about 290.93% and 131.48%, respectively, compared to the unpartitioned circuit. Similar experiments on s15850 for the transition probability thresholds of 10^{-4} and 2×10^{-5} show an increase of 290.26% and 203.11% in TCTCR, respectively. Moreover, the ratio of observing wrong results in primary outputs increases significantly, which could be helpful for Trojan detection using functional methods.

References

[1] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware Trojan: Threats and Emerging Solutions. in *Proceedings of the High Level Design Validation and Test Workshop*, pages 166–171. IEEE, 2009.

[2] Yu Liu, Ke Huang, and Yiorgos Makris. Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting. in *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[3] Mohammad Tehranipoor, Hassan Salmani, and Xuehui Zhang. *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection.* Springer Science & Business Media, 2013.

[4] Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware Trojan taxonomy and detection. *Design and Test of Computers*, 27(1):pages 10–25. IEEE, 2010.

[5] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Electromagnetic circuit fingerprints for hardware Trojan detection. in *Proceedings of the International Symposium on Electromagnetic Compatibility (EMC)*, pages 246–251. IEEE, 2015.

[6] Yu Liu, Yier Jin, Aria Nosratinia, and Yiorgos Makris. Silicon demonstration of hardware Trojan design and detection in wireless cryptographic ICs. *Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4): pages 1506–1519. IEEE, 2017.

[7] Hassan Salmani. COTD: Reference-Free Hard-

ware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist. *Transactions on Information Forensics and Security*, 12(2): pages 338–350. IEEE, 2017.

[8] Abdullah Nazma Nowroz, Kangqiao Hu, Farinaz Koushanfar, and Sherief Reda. Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12): pages 1792–1805. IEEE, 2014.

[9] Chongxi Bao, Domenic Forte, and Ankur Srivastava. On application of one-class SVM to reverse engineering-based hardware Trojan detection. in *Proceedings of the 15th International Symposium on Quality Electronic Design (ISQED)*, pages 47–54. IEEE, 2014.

[10] Shantanu Dutt and Li Li. Trust-Based Design and Check of FPGA Circuits Using Two-Level Randomized ECC Structures. *Transactions on Reconfigurable Technology and Systems*, 2(1): pages 1-6. ACM, 2009.

[11] Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. New Design Strategy for Improving Hardware Trojan Detection and Reducing Trojan Activation Time. in *Hardware-Oriented Security and Trust (HOST)*, pages 66–73. ACM, 2009.

[12] Atieh Amelian and Shahram Etemadi Borujeni. A side-channel analysis for hardware trojan detection based on path delay measurement. *Journal of Circuits, Systems and Computers*, 27(09): pages 1–13, 2018.

[13] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan Detection Using IC Fingerprinting. in *Proceedings of the Symposium on Security and Privacy*, pages 296–310. IEEE, 2007.

[14] Reza M Rad, Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. Power Supply Signal Calibration Techniques for Improving Detection Resolution to Hardware Trojans. in *Proceedings of the International Conference on Computer-Aided Design*, pages 632–639. IEEE/ACM, 2008.

[15] Xiaoxiao Wang, Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis. in *Proceedings of the Defect and Fault Tolerance of VLSI Systems (DFTVS)*, pages 87–95. IEEE, 2008.

[16] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. Scalable test generation for trojan detection using side channel analysis. *Transactions on Information Forensics and Security*, 13(11): pages 2746–2760. IEEE, 2018.

[17] Behnam Khaleghi, Ali Ahari, Hossein Asadi, and Siavash Bayat-Sarmadi. FPGA-Based Protection Scheme against Hardware Trojan Horse Insertion Using Dummy Logic. *Embedded Systems Letters*, 7(2): pages 46–50. IEEE, 2015.

[18] Xuehui Zhang and Mohammad Tehranipoor. RON: An On-Chip Ring Oscillator Network for Hardware Trojan Detection. in *Proceedings of the Conference on Design, Automation and Test in Europe and Exhibition*, pages 1–6. IEEE, 2011.

[19] Youngok Pino, Vinayaka Jyothi, and Matthew French. Intra-die process variation aware anomaly detection in FPGAs. in *Proceedings of the International Test Conference*, pages 1–6. IEEE, 2014.

[20] Seyed Mohammad Hossein Shekarian, Morteza Saheb Zamani, and Shirin Alami. Neutralizing a design-for-hardware-trust technique. in *Proceedings of the 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS)*, pages 73–78. IEEE, 2013.

[21] Xue Mingfu, Hu Aiqun, and Li Guyue. Detecting hardware Trojan through heuristic partition and activity driven test pattern generation. in *Proceedings of the Communications Security Conference (CSC)*, pages 1–6. IET, 2014.

[22] Hossain Fakir Sharif, Mohammed Abdul Kader, and Tomokazu Yoneda. EqSA: A Golden-IC Free Equal Power Self-Authentication for Hardware Trojan Detection. in *Proceedings of the International Conference on Innovations in Science, Engineering and Technology (ICISET)*, pages 86–91. IEEE, 2018.

[23] Steve, Trimberger. Trusted design in FPGAs. in *Proceedings of the 44th annual Design Automation Conference*, pages 5–8. ACM, 2007.

[24] Amir Moradi, Barenghi Alessandro, Kasper Timo, and Paar Christof . On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs. in *Proceedings of the 18th conference on Computer and communications security*, pages 111–124. ACM, 2011.

[25] Rajat Subhra Chakraborty, Saha Indrasish, Palchaudhuri Ayan, and Kumar Naik Gowtham. Hardware Trojan insertion by direct modification of FPGA configuration bitstream. *Design & Test*, 30(2): pages 45–54. IEEE, 2014.

[26] Navneet Kaur Brar, Dhindsa Anaahat, and Agrawal Sunil. Impact of Dummy Logic Insertion on Xilinx Family for Hardware Trojan Prevention. in *Proceedings of the International Conference on Advanced Informatics for Computing Research*, pages 64–74. Springer, 2019.

[27] Kan Xiao, Forte Domenic , and Mark Mohammed Tehranipoor. Efficient and secure split manufacturing via obfuscated built-in self-authentication. in *Proceedings of the International symposium*

on hardware oriented security and trust (HOST), pages 14–19. IEEE, 2015.

[28] Kan Xiao, Forte Domenic , and Mark Mohammed Tehranipoor. A novel built-in self-authentication technique to prevent inserting hardware trojans. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12): pages 1778–1791. IEEE, 2014.

[29] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. Scalable test generation for trojan detection using side channel analysis. *Transactions on Information Forensics and Security*, 13(11): pages 2746–2760. IEEE, 2018.

[30] Rana Elnaggar, Krishnendu Chakrabarty, and Mehdi B Tahoori. Hardware trojan detection using changepoint-based anomaly detection techniques. *Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12): pages 2706–2719. IEEE, 2019.

[31] Apostolos P Fournaris, Lampros Pyrgas, and Paris Kitsos. An efficient multi-parameter approach for FPGA hardware Trojan detection. *Microprocessors and Microsystems*, 71: pages 102863–102878. Elsevier, 2019.

[32] ISCAS89 Benchmarks. http://www.pld.ttu.ee/~maksim/ benchmarks/iscas89/verilog/. [Online].

[33] Francis Wolff, Chris Papachristou, Swarup Bhunia, and Rajat S Chakraborty. Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. in *Proceedings of the conference on Design,Automation and Test in Europe*, pages 1362–1365. ACM, 2008.

[34] Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. in *International Symposium on Circuits and Systems (ISCAS)*, pages 2021–2024. IEEE, 2015.

[35] Faiq Khalid, Syed Rafay Hasan, Osman Hasan, and F Awwad. Behavior Profiling of Power Distribution Networks for Runtime Hardware Trojan Detection. in *Proceedings of the International Midwest Symposium on Circuits and Systems (MWSCAS-2017)*, pages 1316–1319. IEEE, 2017.

[36] Xiaotong Cui, Kun Ma, Liang Shi, and Kaijie Wu. High-level synthesis for run-time hardware Trojan detection and recovery. in *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[37] Richard Neil Pittman. Partial Reconfiguration: A Simple Tutorial. Technical report, Technical Report, 2012.

[38] Partial Reconfiguration User Guide. UG702 (v12.3), Xilinx. *Inc., October*, 5, 2010.

[39] Xilinx. http://www.xilinx.com. [Online].

[40] Jeyavijayan Rajendran, Vinayaka Jyothi, Ozgur Sinanoglu, and Ramesh Karri. Design and analysis of ring oscillator based Design-for-Trust technique. in *Proceedings of the 29th VLSI Test Symposium*, pages 105–110. IEEE, 2011.

[41] hMetis Hypergraph and Circuit Partitioning. http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download. [Online].

[42] George Karypis and Vipin Kumar. A Hypergraph Partitioning Package, 1998.

[43] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Debdeep Mukhopadhyay, et al. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*, pages 577–596. Springer, 2015.

[44] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. MERS: statistical test generation for side-channel analysis based Trojan detection. in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, pages 130–141. ACM, 2016.

[45] Hassan Salmani and Mark M Tehranipoor. Vulnerability analysis of a circuit layout to hardware trojan insertion. *Transactions on Information Forensics and Security*, 11(6): pages 1214–1225, 2016.

[46] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. MERO: A Statistical Approach for Hardware Trojan Detection. in *Proceedings of the Cryptographic Hardware and Embedded Systems (CHES) Conference* , pages 396–410. Springer, 2009.

[47] Trust-Hub Website. https://www.trust-hub.org/. [Online].

Nastaran Shekofte received her B.Sc. degree from Sharif University of Technology, Tehran, Iran, in 2014, in computer engineering (hardware). Currently, she is an M.Sc. student in Sharif University of Technology. Her research interests include hardware security and trust and cryptographic computations.

Siavash Bayat-Sarmadi received the B.Sc. degree from the University of Tehran, Iran, in 2000, the M.Sc. degree from Sharif University of Technology, Tehran, Iran, in 2002, and the Ph.D. degree from the University of Waterloo in 2007, all in computer en-

gineering (hardware). He was with Advanced Micro Devices, Inc. for about 6 years. Since September 2013, he has been a faculty member in the Department of Computer Engineering, Sharif University of Technology. He has served on the executive committees of several conferences. His research interests include hardware security and trust, cryptographic computations, and secure, efficient, and dependable computing and architectures. He is a member of the IEEE.

Hatameh Mosanaei-Boorani received her B.Sc. and M.Sc. degrees in computer engineering from Isfahan University of Technology (IUT), Isfahan, Iran, and Sharif University of Technology (SUT), Tehran, Iran, in 2014 and 2017, respectively. Currently, she is a graduate research assistant at Sharif University of Technology. Her research interests include digital circuit design, VLSI, image processing, cryptographic computations, hardware security, and hardware trust.