6-22-2016

Design and Validation for FPGA Trust under Hardware Trojan Attacks

Sanchita Mal-Sarkar
Cleveland State University, s.malsarkar@csuohio.edu

Robert Karam
University of Florida

Seetharam Narasimhan
Case Western Reserve University

Anandaroop Ghosh
Case Western Reserve University

Aswin Krishna
Case Western Reserve University

Follow this and additional works at: https://engagedscholarship.csuohio.edu/enece_facpub

Part of the Electrical and Computer Engineering Commons

See next page for additional authors

How does access to this work benefit you? Let us know!

Publisher's Statement

Repository Citation

Authors

Sanchita Mal-Sarkar, Robert Karam, Seetharam Narasimhan, Anandaroop Ghosh, Aswin Krishna, and Swarup Bhunia

Design and Validation for FPGA Trust under Hardware Trojan Attacks

Sanchita Mal-Sarkar, *Member, IEEE*, Robert Karam, *Student Member, IEEE*,
Seetharam Narasimhan, *Member, IEEE*, Anandaroop Ghosh,
Aswin Krishna, *Student Member, IEEE*, and Swarup Bhunia, *Senior Member, IEEE*

Abstract—Field programmable gate arrays (FPGAs) are being increasingly used in a wide range of critical applications, including industrial, automotive, medical, and military systems. Since FPGA vendors are typically fabless, it is more economical to outsource device production to off-shore facilities. This introduces many opportunities for the insertion of malicious alterations of FPGA devices in the foundry, referred to as hardware Trojan attacks, that can cause logical and physical malfunctions during field operation. The vulnerability of these devices to hardware attacks raises serious security concerns regarding hardware and design assurance. In this paper, we present a taxonomy of FPGA-specific hardware Trojan attacks based on activation and payload characteristics along with Trojan models that can be inserted by an attacker. We also present an efficient Trojan detection method for FPGA based on a combined approach of logic-testing and side-channel analysis. Finally, we propose a novel design approach, referred to as Adapted Triple Modular Redundancy (ATMR), to reliably protect against Trojan circuits of varying forms in FPGA devices. We compare ATMR with the conventional TMR approach. The results demonstrate the advantages of ATMR over TMR with respect to power overhead, while maintaining the same or higher level of security and performances as TMR. Further improvement in overhead associated with ATMR is achieved by exploiting reconfiguration and time-sharing of resources.

INTRODUCTION

FIELD programmable gate arrays (FPGA) are integrated circuits (IC), consisting of an array of logic blocks and distributed interconnect structure, which can be programmed and *re*programmed several times post-manufacturing to implement logic functions. Early FPGAs were used only as prototypes for implementing ASIC designs on hardware for functional verification. In the past decade, advances in fabrication processes have reduced the performance gap between FPGAs and ASICs, leading to FPGAs being the main solution in a variety of performance-critical applications. Additionally, designs implemented on FPGAs do not suffer the increasing non-recurring engineering (NRE) costs of ASIC production. FPGAs are typically designed as an interleaved array of configurable logic blocks, programmable interconnects, and distributed embedded memory blocks (EMBs). High resource availability and the flexibility of the interconnect network enables designs to perform multiple parallel operations each cycle for increased computational power compared to sequential processors.

Previous research on programmable logic devices has primarily focused on tapping their potential for implementing signal processing algorithms, building reconfigurable systems, and for applications in a wide range of usage domains, including satellite, automotive, and military systems. More recently, FPGAs have been used for encryption and secure processing due to the efficient implementation of cryptographic algorithms. The growing use of FPGAs in diverse and critical applications has motivated designers to consider the security of these devices. In this context, security refers to protecting against Intellectual Property (IP) Piracy, due to the substantial financial investment involved in developing the IP. Little attention has been directed towards security and assurance of the physical system itself. Malicious alterations to the design are possible at several stages of design flow in FPGAs, as shown in Fig. 1. Security in every stage of the design flow is of growing importance; in response, the Defense Advanced Research Projects Agency (DARPA) [1] has initiated the TRUST in Integrated Circuits program for hardware validation, including FPGA device security and protection of third-party IP.

To the best of our knowledge, FPGA system protection has been investigated by relatively few researchers [2],[4], [3]. Hadzic et al. describe various logical and electrical attacks possible to cause malfunction and physical destruction of an FPGA device. These attacks are caused by creating internal conflicts in the device by inserting malicious code in the configuration files of designs [3]. Drimer presents attack models such as replay attacks, cloning, authentication attacks, power analysis attacks, invasive and semi-invasive attacks, and radiation attacks as related to the security of FPGA design [4]. None of these prior works, however, discusses hardware attacks in the foundry as a means to cause malfunction and leak IP.

- S. Mal Sarkar is with the Department of Electrical Engineering and Computer Science, Cleveland State University, Cleveland, OH 44114. E mail: s.malsarkar@csuohio edu.
- R. Karam and S. Bhunia are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611. E mail: robkaram@ufl.edu, swarup@ece.ufl.edu.
- S. Narasimhan, A. Ghosh, and A. Krishna are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106. E mail: {sxn124, axg468, ark70}@case.edu.
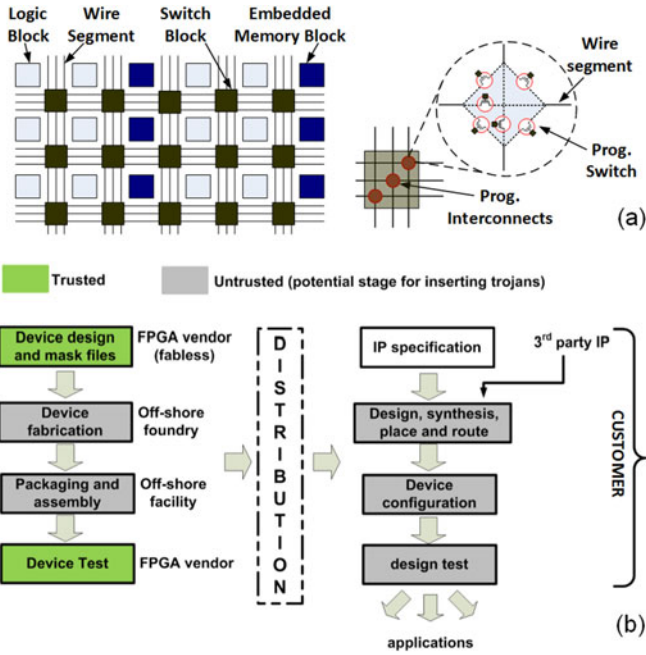
Fig. 1. Simplified diagram of FPGA design flow from device design to deployment showing possible stages for malicious alterations.

Trimberger discusses the issue of foundry trust as related to the production of FPGAs [2]. However, Trimberger claims that attacks on FPGA hardware in the foundry as a means to tamper with the functionality of the final design are unlikely for several reasons: (1) the foundry does not know about the design being mapped or end application of the devices; (2) a large number of reconfigurable resources enables the usage of Triple Modular Redundancy (TMR), so that critical functions are replicated three times, and a majority vote circuit can be used to select the most common output; (3) exhaustively testing the bitstream security features can ensure that any attacks on them are detected during testing; (4) a sample of chips can be destructively tested after fabrication to identify extraneous logic.

However, even with the above-mentioned security features ensuring the integrity of FPGA devices, we can demonstrate that a variety of Trojan attacks are possible in the foundry. First, we show that not all attacks have to depend on the final design and it is possible to insert malicious logic, which is *independent* of the design. Besides causing malfunction, a Trojan in the FPGA can be used to either partially or even completely leak the IP implemented in the device. Moreover, an attacker in the foundry can distribute Trojans dependent on the internal logic values over the chip. Even though such Trojans which depend on the IP have a low probability of being triggered, this is still unacceptable for mission-critical applications.

Second, although bitstream security functions can be fully tested to ensure that no attacks are made on the FPGA's security, an attack can be made to steal a key and *not* cause a malfunction. Thus, thoroughly testing the security functions may not help in protecting the IP from being copied by an attacker. Furthermore, sampling the fabricated devices may not provide complete confidence that the device has not been altered during production; in other words, though invasive testing may not reveal any Trojans,

their absence in the tested devices does not necessarily imply absence on other untested devices. Since most FPGA vendors are fabless and rely on off-shore foundries for production, hardware Trojans inserted in the foundry pose a practical threat to the functionality and reliability of the device.

Third, the TMR technique suggested in [2] suffers from high area, power, and performance overhead. Given the relatively high power consumption and poor performance of FPGAs relative to application specific ICs (ASICs), this technique can be used only for a few logic functions, leaving other functions vulnerable to attack. Moreover, the technique assumes that an attack will affect only one of the identical functions-it will not be useful if the payload of the Trojan is the output of the voting circuit.

Therefore, in this paper, we make the following key contributions:

1) We provide a detailed analysis of hardware Trojan attacks in FPGA devices, as well as a taxonomy of hardware Trojans in FPGA.
2) We present a trust validation approach for FPGA devices, based on both functional and side-channel validation, to counter diverse FPGA Trojan attacks. These approaches enable detection of arbitrary Trojan circuits in FPGA hardware.
3) We introduce a modified version of TMR, which we call Adapted Triple Modular Redundancy (ATMR), to enable robust protection against Trojan attacks with considerably less power overhead than TMR.

The remainder of the paper is organized as follows. Section 2 provides a brief overview of hardware Trojan and how it differs from faults. Section 3 provides an analysis of hardware Trojans that can be inserted into FPGA devices during production. In Section 4 we discuss the methods that can be used for detecting the Trojan attacks. We describe an approach for run-time Trojan tolerance in Section 5. We discuss the simulation results for the Trojan tolerance scheme and important design/test considerations in Section 7. Finally, we conclude in Section 8.

BACKGROUND

Hardware Trojan Attacks

Malicious modifications of integrated circuits, referred to as Hardware Trojans, have emerged as a major security threat due to widespread outsourcing of IC manufacturing to untrusted foundries. An adversary can potentially tamper with a design in these fabrication facilities by inserting malicious circuitry, leading to potentially catastrophic malfunctions in security-critical application domains, such as the military, government, communications, space, and medicine. Conventional post-manufacturing testing, test generation algorithms, and test coverage metrics often fail to detect Hardware Trojans due to their diversity, complexity, and rare triggering conditions.

An intelligent adversary can design a Trojan to only trigger under very rare conditions on an internal node, which is unlikely to arise during post-manufacturing test, but can be triggered during long hours of in-field operation [24]. The detection of Trojans by employing side-channel parameters, such as power trace or delay overhead, is
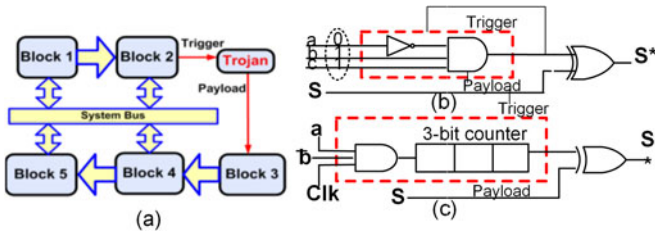
Fig. 2. (a) General model of a hardware Trojan circuit realized through malicious modification of a hardware. (b) An example of combinational Trojan. (c) An example of sequential Trojan.

limited due to the large process variations in nanoscale IC technologies, detection sensitivities of small Trojans, and measurement noise [25]. Often these issues mask the effect of Trojan circuits, especially for ultra small Trojans. From an adversary's perspective, the desired features for a successful Trojan are as follows: rarely activated to evade logic based testing, low overhead to evade side-channel based detection approach, and low side-channel signature to evade Design for Security (DfS) hardening mechanisms.

The condition of Trojan activation is referred to as the *trigger*, and the node affected by the Trojan is referred to as its *payload*. Trojans can be classified based on their triggering conditions or payload mechanisms. The trigger mechanism can be either digital or analog. Digitally triggered Trojans can be classified into combinational and sequential Trojans. Trojan can also be classified into digital and analog based on the payload mechanisms. Digital Trojans invert the logic values at internal nodes or modify the contents of memory locations, while the analog payload Trojans may affect circuit parameters, such as performance, power, and noise margin.

A combinational Trojan is activated on the simultaneous occurrences of a particular condition at certain internal nodes, while a sequential Trojan acts as a time-bomb, exhibiting its malicious effect due to a sequence of rare events after a long period of operation. Fig. 2a illustrates the general scenario of a Trojan attack in a design, where a Trojan is realized through the malicious modification of the circuit with a trigger condition and payload. Fig. 2b shows an example of combinational Trojan which does not contain any sequential elements, and depends only on the simultaneous occurrence of a set of rare node conditions. Conversely, the sequential Trojans shown in Fig. 2c undergo a sequence of state transitions before triggering a malfunction. The 3-bit counter causes a malfunction at the node S on reaching a particular count, and the count is increased only when the condition $a = 1$, $b = 0$ is satisfied at the positive clock-edge.

Protection against hardware Trojan has been widely explored [25], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], by researchers. These approaches are based on the following three approaches: (1) specialized functional testing [14] that rely on triggering an unknown Trojan and observing its effect in output ports of a design; (2) side-channel analysis that rely on observing a Trojan effect in physical parameters, such as supply current or path delay [25], [40] and (3) design/integration approaches [31], [32], [33], [34], [35] that either prevent a Trojan insertion or facilitate detection during production test.

Fault Versus Trojan

Conventional fault models can properly *represent* hardware Trojans; however, standard functional/structural testing methods, as well as fault tolerance schemes cannot adequately *protect* against diverse Trojan attacks. The major differences are that unanticipated behavior is not included in the fault list [24], and hardware Trojans usually have a covert trigger condition, which is expected to rarely occur during normal field operation. On the contrary, traditional faults, run-time failures, and soft-errors typically occur at random locations and are sensitized through arbitrary, typically non-rare—and hence easily detectable—condition. Due to the random occurrence of faults, in many cases they may turn out to be ineffective or benign [26]. The payload of a Trojan, however, is likely to be carefully selected by an adversary to cause critical system failure or information leakage.

With respect to post-silicon validation or run-time tolerance, the difference between faults and Trojans are twofold. First, faults induced by manufacturing defects are static, and run-time failures are one-cycle transient failures. However, hardware Trojans are dynamic, causing malfunction for one or more cycles after triggering, and can also go back to a benign state after causing the malicious effect. In a spatial sense, manufacturing faults and soft errors can occur randomly in any part of a design. However, hardware Trojans are inserted by intelligent adversaries and hence are likely placed at strategic locations (e.g., in the key-dependent logic of a crypto-chip, enabling key leakage) while being hard-to-detect.

HARDWARE TROJAN ATTACKS IN FPGA

Before we describe the taxonomy of hardware Trojans in reconfigurable hardware, it is necessary to understand why such Trojans can be inserted in the foundry. Reconfigurable hardware consists of a regular array of identical reprogrammable cells and other modules connected through a distributed programmable interconnect structure. Since most of the chip is occupied by the regular structure of logic blocks and interconnect, it is relatively easy (compared to ASICs) for an attacker to reverse engineer the device and identify the regular structures as well as additional modules. For example, a DSP core or clock manager can be easily identified from the layout of the FPGA and can be a potential target for hardware attacks as described in the next section [27] While the proposed Trojan models and detection methods may be applicable to many programmable logic devices, we focus on hardware Trojans in the widely-used SRAM-based FPGAs

Programmability in FPGAs can be used to change the logic and electrical properties of a system [3]. Although this programmability provides flexibility to designers to quickly implement their designs according to their requirements, it can be exploited by an adversary to mount attacks to cause malfunction, leak sensitive information, and even cause physical damage [3],[5]. This also differentiates FPGA hardware Trojans from ASIC Trojans where the former alter the state of the system through malicious reprogramming *after* configuration.

Wang, Tehranipoor, and Plusquellic proposed a taxonomy of hardware Trojans in ICs [6]. In this paper, we present a taxonomy of FPGA-specific hardware Trojans that alter the programmed state of logic and I/O blocks. We
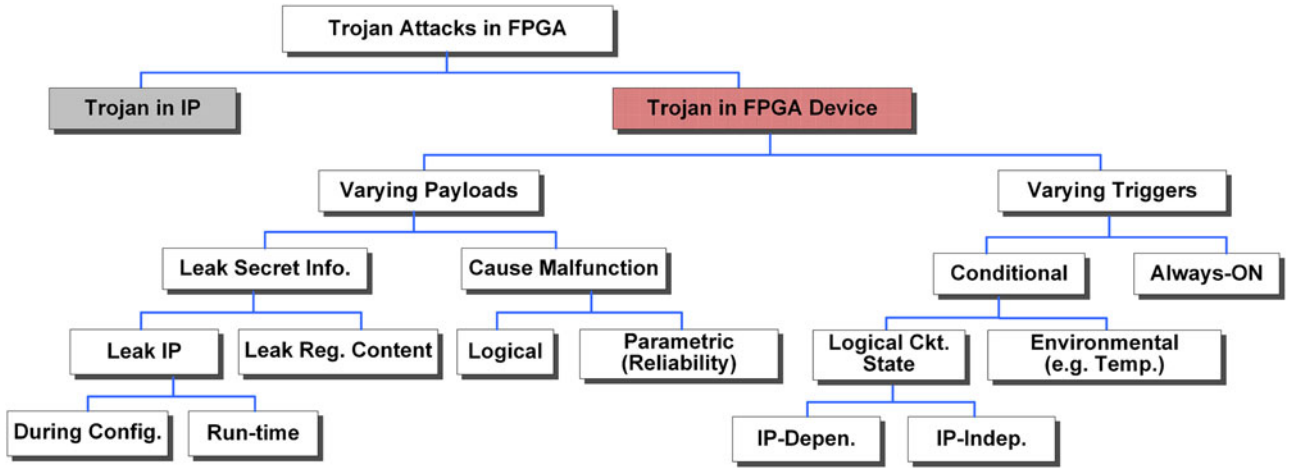
Fig. 3. Taxonomy of hardware Trojans in FPGA.

classify the variety of FPGA hardware Trojans into two main categories as shown in Fig. 3 according to their activation and payload characteristics. While it may be possible to classify FPGA Trojans based on other characteristics such as size or distribution, we believe that the proposed classification covers most FPGA Trojans and is adequate to evaluate the capabilities and limitations of detection methods.

Activation Characteristics

Based on the activation characteristics, Trojans can fall into two subcategories marked as *condition-based* and *always-on* in Fig. 3. Always-on Trojans are always active and perform their defined purpose of malfunction or leaking sensitive information. An example in this subcategory would be a Trojan that inverts all the bits of the configuration bitstream as the device is being programmed. This class of Trojans may not be inserted by an intelligent adversary since they can be easily detected during conventional testing.

Condition-based FPGA Trojans, on the other hand, wait until a particular condition is met before they become active and cause malfunction. At this level, Trojans can be further classified as *logic-based* and *sensor-based* (e.g., temperature, delay). At the lowest level, logic-based FPGA Trojans can be further divided into *IP dependent* and *IP independent* subcategories which we will discuss in detail with examples.

IP-Dependent Trojans. IP-dependent Trojans represent a subclass of Trojans whose trigger signals depend on the design implemented in the device. Fig. 4 shows a simplified architecture of FPGAs consisting of a regular array of programmable logic blocks and interconnects. As shown in Fig. 4, an adversary can insert a malicious circuit which monitors the logic values of several nodes such as configuration logic, outputs of logic modules, or look-up table (LUT) values. When triggered, such a Trojan can cause malfunction in many different ways, e.g., by altering the values stored in LUTs or configuration cells in the interconnect network to cause incorrect routing between logic blocks, or writing random values into the embedded memory.

Since the final IP design is not available to the foundry during device fabrication, an attacker who plans to insert design-dependent hardware Trojans must do so without assuming anything about the IP. Even though the probability of such a Trojan becoming active is very low, an attacker may distribute many such Trojans over the entire chip to increase the likelihood of causing malfunction. Given the growing scope of the FPGA domain, IP-dependent Trojans are a practical threat that must be considered for hardware assurance.

IP-Independent Trojans. An intelligent attacker can also insert Trojans whose activation conditions do not depend on the final design. Such Trojans can be inserted to alter the functionality of critical modules of the device. For example, Xilinx Spartan-3, Virtex-II, Virtex-II Pro FPGAs contain a separate module for clock management known as the digital clock manager (DCM) as shown in Fig. 5. This module contains a delay locked loop (DLL) for reconditioning clock signals. Additionally, it contains a frequency synthesizer for producing multiples or divisions of the input clock. Configuration parameters for the DCM are stored in its local SRAM. A simple Trojan design could simply increment an n-bit counter each clock edge until a particular number is reached, and then modify the configuration to produce a faster clock. This in turn can cause the critical path logic to fail in a sequential circuit.

As another example of IP-independent Trojans, consider a simplified programmable I/O block shown in Fig. 6 which contains buffers, enable logic, slew rate control, and level
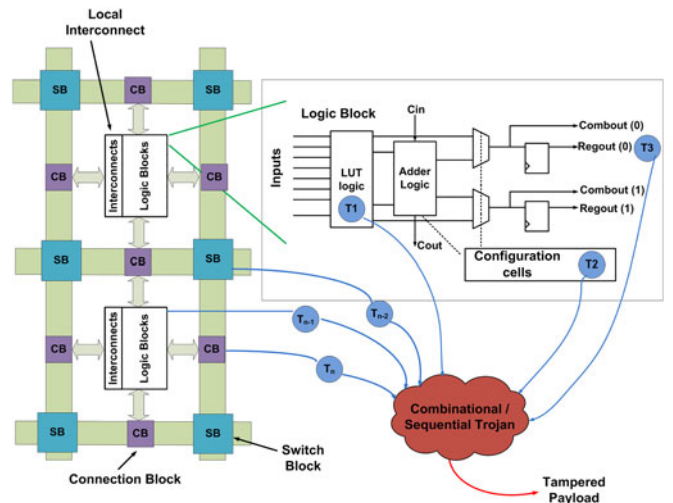

Fig. 4. Simplified architecture of an FPGA showing the trigger points that a Trojan may use.
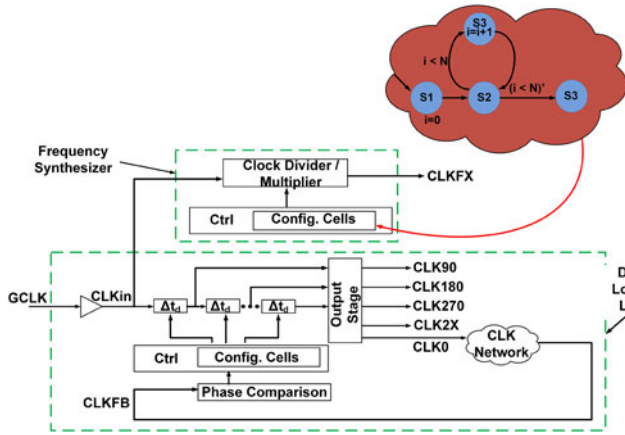
Fig. 5. Simplified schematic of digital clock manager (DCM) in Xilinx's Virtex-5 FPGA device.

Fig. 7. Diagram showing examples of payloads that can be altered by an implanted Trojan circuit.

shifters for logic level compatibility. Similar to the DCM, the configuration of the I/O block is stored in local SRAM. A counter-based Trojan could be inserted in the device; when activated, this could modify the output-slew control, disable the output logic, or cause physical damage with logic-level incompatibility at the I/O port. Again, these Trojans could be distributed in many I/O blocks to improve the chances of causing malfunction.

Payload Characteristics

Hardware Trojans can also be classified based on their intended behavior. Trojans can be inserted for causing malfunction or for leaking sensitive information. In the former case, Trojans alter the functionality of the design in some way, while Trojans designed for leaking sensitive information may do so without modifying the logic functionality of the design.

Trojans for Malfunction. Trojans in this category can be further classified into two subcategories based on whether they cause *logical* malfunction or *physical* malfunction. Trojans presented in the previous sections cause logic malfunction by modifying the values in the LUTs, causing undesired routing between two logic modules, etc. Fig. 7 shows additional examples of payloads affected by Trojans.
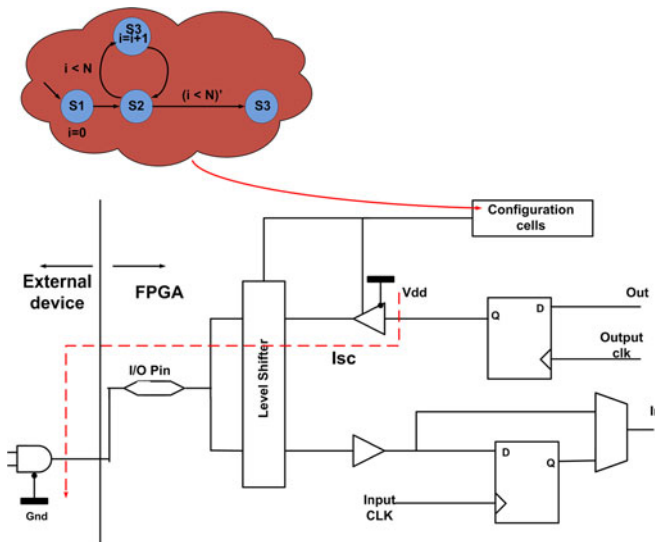

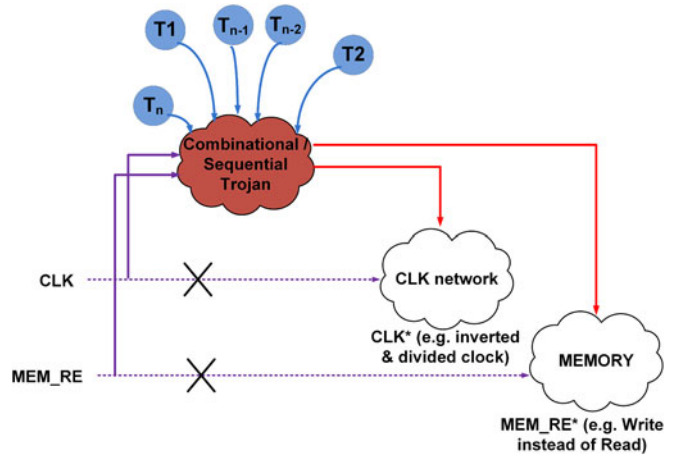Fig. 6. Programmable I/O block containing hardware Trojans to cause logical and electrical malfunction.

Trojans intended to cause physical damage can create electrical conflicts at the I/O ports or at the programmable interconnects. Consider the programmable I/O block in Fig. 6. When an I/O port is configured to be an input by a design, the configuration cells in the I/O block should disable the output block to prevent internal conflicts. A counter-based Trojan can be inserted in the foundry which detects the state of the I/O port and begins counting. When the counter counts to the final value, the Trojan may enable the output logic when the port is configured as an input. This would cause a high short-circuit current to flow between the FPGA and the external device, possibly damaging the system. These Trojans are similar to the *MELT* viruses described in [3] except that Trojans causing physical destruction may also be inserted in the foundry.

IP-Leak Trojans. Since IP designs involve a high development cost and contain sensitive information, security is of utmost importance Many high-end FPGAs such as Xilinx's Virtex4 and Virtex5, and Altera's StratixII and StratixIII offer bitstream encryption to prevent unauthorized cloning of the bitstream. Fig. 8 shows the security features in a generic FPGA device that contains the programmable logic array (bottom right in the figure), configuration logic which controls the programming of the SRAM cells in the logic array, interconnect network, and additional modules in the device [7],[8]. The device also contains a decryptor module for decrypting the bitstream using a key stored in a non-volatile memory. Security measures in the device (1) prevent the key from being read and sent to a port by clearing the configuration data and keys when a read attempt is made, (2) prevent readback of the configuration data, and (3)
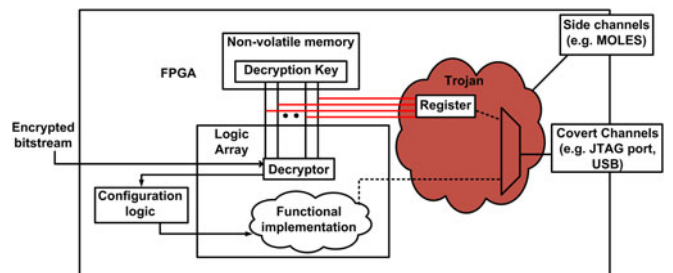

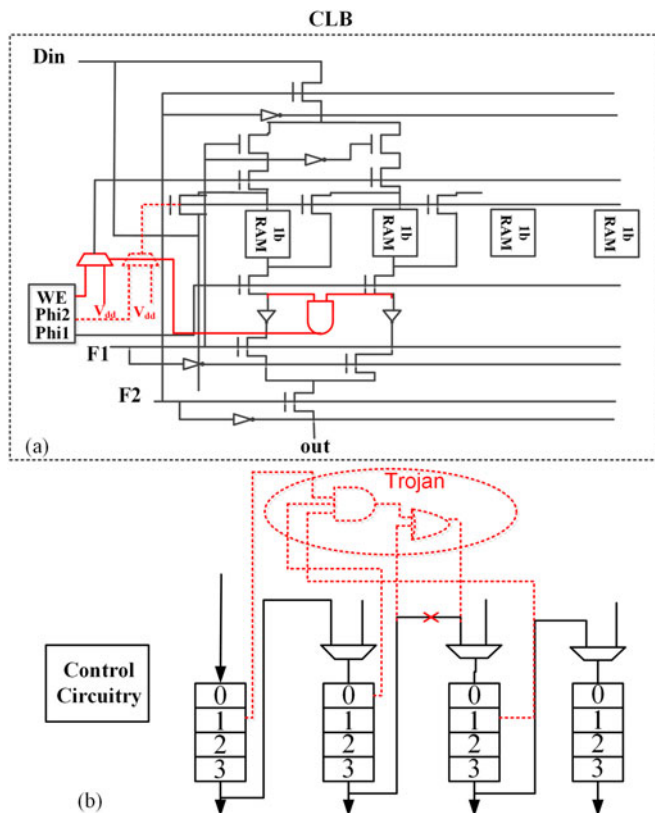Fig. 8. FPGA device with security features for bitstream decryption.

Fig. 9. Hardware Trojan inside: (a) a configurable logic block (CLB) and (b) an embedded memory block (EMB) of FPGA.

restrict decryptor access after configuration [2]. However, all of these measures only prevent malicious code in an IP from accessing the key or configuration data.

Hardware Trojans can leak the IP in two ways: by leaking the decryption key, or by leaking the design itself. An attacker in the foundry can insert an extraneous circuit (Fig. 8) to tap the wires connecting the non-volatile memory and decryptor module. Even if the decryptor module is implemented in the logic array by using a decryptor bitstream as mentioned in [7], such an instantiated module must have access to the non-volatile key for decryption. A copy of the key can be stored in the Trojan, which may then leak it through side-channels or covert-channels. Using side-channels, a Trojan can hide the key in the power traces or by emitting electromagnetic radiation containing the information and an attacker can observe these signals to steal the key. For example, the MOLES Trojan presented in [9] uses a spread-spectrum technique to leak the key in the power traces over several clock cycles. Alternatively, a Trojan may also multiplex the JTAG port, USB port, or any other programming port to leak the key through covert channels when the ports are not being used.

Since SRAM-based FPGAs are volatile, an external device must be used to store the encrypted design. If an adversary is in possession of the FPGA device loaded with the design, the encrypted bitstream can be stolen by eavesdropping the connection between an FPGA's programming ports and the external device storing the encrypted bitstream. In other cases, a Trojan may fake a request to the external device to send the programming data to the FPGA. This time, however, the Trojan muxes the bitstream and

rather than sending it to the the decryptor, it may store blocks of the bitstream at any given time and leak them through side-channels or covert-channels.

Trojans in CLB/EMB

FPGA configurable logic blocks (CLBs) and embedded memory blocks are highly flexible, but require significant configuration to implement the desired functions. This severely harms the memory or logic integration density in FPGA which makes it more amenable for Trojan insertion. Fig. 9 shows a FPGA CLB, which can act as a 2-input look up table, a 4-bit Random Access Memory (RAM), or a 4-bit shift register [29]. In Fig. 9a, the inserted Trojan has been shown in red: the trigger condition is derived from the memory content of two consecutive RAM locations, and can harm the shift register functionality or the write enable functionality of the memory block at run-time. The trigger condition can also be generated from the output of other CLBs, or alternatively can be derived from the output of other functional units.

Fig. 9b shows a Trojan instance inserted inside an embedded memory block in a commercial FPGA device [30]. Similar to a CLB, an EMB is also capable of executing functionalities like shift register, FIFO etc. in addition to acting as Random Access Memory. The control circuitry shown in Fig. 9b decides between normal read operation and shift register operation inside the EMBs. The inserted logic or Trojan conditionally affects the shift operation inside a EMB by using adjacent memory contents and so can be triggered at run-time. It can be noted that the similar trigger conditions can also be effectively used to leak the contents of the memory to the output port. Such malfunctions can be achieved by changing the address bits of the memory blocks in different clock cycles and reading out the immediate next location of the memory in each and every cycle so as to obtain the complete memory contents stored in a particular EMB or a set of EMBs.

TROJAN DETECTION METHODS FOR FPGA

In this section, we discuss some detection methods for detecting FPGA hardware Trojans. These methods must be used by the FPGA vendors when they receive the chips from the off-shore foundry. We assume that the testing facility used by a FPGA vendor is secure, eliminating the possibility that an adversary in the testing facility could intentionally not detect malicious alterations. Detection methods can be classified into three categories: *Visual detection techniques*, *logic testing*, and *side-channel analysis*.

Visual Detection Methods. This class of detection methods uses imaging to identify any malicious insertions in the chip. These techniques include using X-ray imaging [10], scanning optical microscopy (SOM), scanning electron microscopy (SEM) [11], and picosecond imaging circuit analysis (PICA), among others. These methods, however, can be expensive in cost and analysis time. Moreover, these techniques suffer from lack of resolution to decipher logic/transistor/interconnect level information, primarily due to the obstruction by the stack of metal layers in modern FPGAs [12]. With increasing device density due to technology scaling, effectiveness of the imaging techniques is
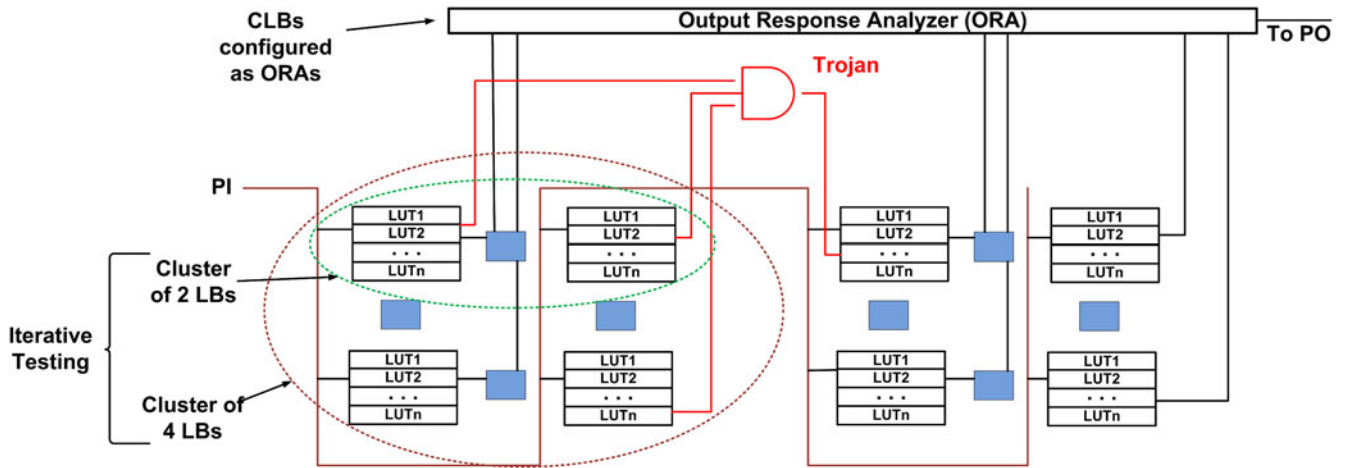
Fig. 10. Iterative logic-based self-checking test.

expected to reduce significantly. Partial de-layering of ICs appears more effective [13]; however, it may in turn render an FPGA non-functional. Due to the above limitations, imaging analysis may not be a viable Trojan detection technique.

Logic-Based Testing. Standard logic testing of FPGAs by automatic test pattern generation (ATPG) tools is used for detecting faults. Using input vectors, all the programmable logic blocks can be tested to function correctly without faults. For example, a stuck-at-0 fault in the programmable logic blocks can be detected by mapping an *AND* function in the blocks and applying all-1 inputs. However, since Trojan models are very different from fault models, a better approach is required to detect Trojans. For example, an attacker can insert a Trojan which uses many values of the LUT SRAM cells or configuration cells as trigger nodes, and such a Trojan will not be detected using testing based on fault models.

Due to the availability of a large number of programmable blocks containing countless nodes, exhaustive testing of all combinations of nodes is infeasible. For a k-input LUT, having $L = 2^k$ cells in the logic block, $F = 2^{2^k}$ distinct functions are possible. For an n-input Trojan, the inputs can be chosen in $^L C_2$ ways from the L cells. Since each combination can in turn be one out of 2^n values, the total number of functions that need to be mapped to exhaustively test a logic block becomes $F = 2^{2^k} \times 2^n$. For example, for a 2-input LUT having four cells, a 2-input Trojan can be chosen from the four cells in $4C_2 = 6$ ways. If the chosen two cells are designated a and b, then the trigger values for the Trojan can be $(ab, \overline{a}b, a\overline{b}, ab)$, requiring 24 functions to be mapped. However, since entire functions are mapped onto the LUTs, mapping one function with values a, b, c, d can detect several Trojans such as ab, bc, cd, etc., thus requiring fewer functions to be mapped.

Still, if the trigger nodes are distributed among logic blocks, the sheer number of logic blocks, LUTs, and configuration cells makes it impossible for exhaustive testing to be used for Trojan detection. Due to this restriction, we propose a statistical approach of iterative self-checking based on the MERO test approach [14] as shown in Fig. 10. Since the probability of activating a Trojan using the logic testing approach decreases with the number of trigger inputs, we assume that the number of trigger nodes n is small (two to four) and only distributed within nearby CLBs.

Algorithm 1 shows the major steps in the proposed statistical test approach. We begin with a cluster of logic blocks of size S, the number of nodes B, random configuration set C, and input pattern set I. For each configuration mapped to the logic blocks, we activate the nodes to logic-0 and logic-1 several times using different input patterns. This procedure is done concurrently with the other remaining clusters of logic blocks, such that for each configuration all clusters are tested simultaneously. The outputs of the clusters can be tested by a few logic blocks configured as output response analyzers (ORA). These ORAs can be exhaustively tested at the beginning of the test procedure. Any Trojan that is activated will be observed at the primary outputs. Then, the cluster size is iteratively increased (e.g., to include two neighboring logic blocks) and the process is repeated. Such a statistical approach of testing can be effective since an attacker does not know the exact test procedure to cleverly insert malicious circuits. Moreover, for larger combinational and sequential Trojans, this approach can be useful to cause partial activation of Trojans for detection using side-channel techniques.

Algorithm 1. The Statistical FPGA Testing Approach

Inputs: Cluster of logic blocks of size S, the number of nodes B, random configuration set C and input pattern set I
Outputs: Trojan-free ICs
1: **for all** $S = 2^Y$ such that $0 \leq Y \leq K$ **do**
2:   **for all** nodes in S **do**
3:     **for all** configurations in C **do**
4:       **for all** vector v_i in I **do**
5:         propagate values to ORAs (simultaneously test other clusters of same size)
6:         observe outputs of ORAs
7:       **end for**
8:     **end for**
9:   **end for**
10: **end for**

Redundancy and reconfigurability are two key features of FPGA devices that can be that helpful for Trojan

detection. Just as these features are used to counter run-time failures in FPGAs, so can they be used to counter against FPGA hardware *and* design Trojans. In the case of FPGA hardware, reconfigurability allows the activation of several nodes in the logic blocks through different logic values. Redundancy can be used during testing, for example, by using N-modular redundancy to ensure that the trigger nodes present in the ORAs can also be detected by comparing the outputs of many ORAs. This is under the assumption that Trojans in FPGA hardware (localized or distributed) do not affect all the resources in the same way. This can be coupled with dynamic run-time reconfigurability to improve the level of security.

Side-Channel Analysis. Logic-based testing may not be effective for activating large combinational or sequential Trojans due to the extremely large number of possible trigger nodes. Side-channel analysis involves the measurement and analysis of information obtained from an IC's side-channels. The information could be based on timing, power consumption, or electromagnetic radiation. Side-channel analysis has been proposed previously as a powerful technique to detect malicious insertions in an IC [14]. In this section, we specifically concentrate on side-channel information obtained from power consumption in the device.

Static power contains comprehensive information regarding all the gates in the chip (including malicious gates/transistors). Trojans causing physical damage by creating electrical conflicts can also be detected using side-channel analysis since these Trojans result in a large current flow through the power supply. A simple design file can be loaded that configures I/O ports as inputs and then measures the supply current. If these Trojans simultaneously try to configure the port as an output, then a very large current can be detected by current sensors in the device, indicating a malicious modification. Since on-chip current sensors may also be tampered in the foundry during production, they must be tested thoroughly to identify any tampering. An alternative and secure strategy would be to use an on-board current sensor to detect short-circuit conditions.

Trojans which do not cause physical damage and only cause logical malfunction may be extremely difficult to detect by analyzing static power. This is due to the difficulty in isolating the contribution of the malicious insertions from the measured power traces in ICs containing many millions of transistors. On the other hand, transient or dynamic power can be controlled by applying input vectors to reveal information about a few gates which are switching at any given time. The advantage of this type of analysis is that, unlike logic-based testing, a Trojan does not have to become active for detection; it merely needs to cause switching in the Trojan to consume dynamic power. For the IP-independent Trojans presented in Section 3, transient power analysis can be an effective detection method. For example, a counter-based Trojan inserted in the clock manager module can be detected by applying a clock signal to the FPGA and applying constant inputs to prevent logic blocks from switching. An extraneous counter or any sequential circuit will consume transient power as it transitions from one state to another. This contribution to dynamic power can be identified and associated with malicious insertions after accounting for process noise and clock coupling power.

HARDWARE TROJAN TOLERANCE IN FPGA

In this section, we propose a novel approach to protect against Trojan circuits of varying forms in FPGA devices (as discussed in Section 3). The protection approach is based on tolerating Trojan effects during operation by either containing the Trojan effect (through activation detection) or bypassing the effect of Trojan activation using redundant logic. The proposed Trojan tolerance approach takes its inspiration from existing run-time fault tolerance approaches[15], [16]. The focus of the Trojan tolerance approach is to achieve maximum confidence with respect to trusted operation in the presence of Trojans in FPGA hardware, while minimizing the area, power, and performance overhead.

It is a challenge to maintain a satisfactory level of survivability with minimum cost. Each user requires different survivability level, optimizing some attributes at the expense of others. A single Trojan mitigation scheme may not be satisfactory to all FPGA users; therefore various schemes can be combined in a coherent way to satisfy different users' requirements. We propose a hybrid scheme of hardware Trojan tolerance that combines adapted Triple Modular redundancy and dynamic reconfiguration.

Dynamic reconfiguration in FPGA works around failures while allowing time sharing of hardware components, and therefore reduces area overhead while achieving high reliability. However, increasing device failures can significantly affect the yield, run-time reliability, and the number of mappable functions. In addition, hardware Trojans are diverse, requiring different levels of resource usage, power consumption, and response time [17], [18]. Although dynamic reconfiguration allows us to reduce resource requirements, it must be combined with a technique that is capable of detecting Trojans and restoring the original functionality.

Trojan Tolerance through Adapted TMR (ATMR)

Triple modular redundancy is a well-known fault mitigation technique that masks circuit faults by using redundant hardware [15], [16], [19]. A TMR circuit has three copies of the original circuit and a majority voter. A single fault in one of the redundant hardware modules will not produce an error at the output as the majority voter selects the result from the two working modules. Several experiments have demonstrated significant improvements in reliability when using TMR through fault injection and radiation testing [19]. Use of TMR has been explored earlier in FPGA in the context of tolerating run-time failures, such as functional failures induced by soft errors [20], [21]. However, the use of TMR in FPGA results in very high (>3x) area overhead, as well as a performance overhead incurred from additional routing delay. Reducing these effects by selectively applying TMR makes all non-covered functions vulnerable to attack. Moreover, the output of the voting circuit itself may be the target of a Trojan. TMR also cannot protect the system against attacks made to steal a key without causing malfunction.

Minor changes to TMR can not only reduce the overhead associated with redundant hardware, but also enable it to detect *when* an error has occurred, and *which* replica is responsible for the error. This is accomplished by using only two instances at a time instead of three. Using a comparator circuit, the outputs of the two instances are
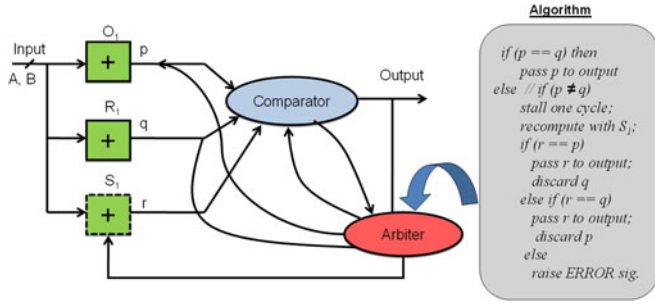
Fig. 11. Proposed Trojan-tolerant application mapping scheme using redundant computing blocks.

Fig. 12. Flowchart showing a variant generation approach from a gate-level design along with an example.

compared; if a mismatch occurs, only then will the third replica, along with an arbiter circuit, be enabled. This enables the system to detect which of the two original replicas was in error, output the correct result, and discard the erroneous replica. During operation, the circuit may halt for a short time interval during the involvement of the third replica and arbiter.

Fig. 11 shows three adders (O_1, R_1, and S_1) that are mapped to the Trojan infected region of an FPGA, and their corresponding outputs (p, q, and r). The comparator will compare the outputs (p, q) of the first two adders (O_1, R_1). The third adder (S_1) will not be used unless there is a mismatch between the two outputs p and q. In case of a mismatch, the comparator, with the help of the arbiter, continues comparing the output r with the outputs (p, q) until it finds a match and it determines which adder is in error. Then the comparator outputs the correct result and prevents the propagation of erroneous data.

Although the third replica is required to determine which replica is in error, only two replicas are enough to flag that at least one of them is infected with a Trojan, prevent the propagation of the Trojan's payload in the system, and stop leakage of potentially sensitive data. Therefore the scheme we have proposed, even without the third replica, will be of particular interest to military and government applications, as well as to commercial entities concerned with guarding their highly sensitive data.

Improving Trojan Coverage with Design of Variants

The Trojan tolerance scheme proposed in Section 5.1 can provide protection against a single Trojan attack in either the original or replica module. However, it cannot protect against simultaneous activation of identical Trojan instances in O_i and R_i. An adversary can incorporate the same Trojans in two or more clusters or CLBs in an FPGA, so that both O_i and R_i can be similarly affected. For example, this can happen if O_i and R_i are identically mapped in two different clusters, both of which have a combinational Trojan triggered by a specific combination of LUT content. In such a scenario, two Trojans in O_i and R_i would trigger at the same cycle with identical malicious effect and the proposed tolerance approach would fail to detect it. To address this scenario, we present an addition to the proposed Trojan tolerance approach. McIntyre et al. [22] have proposed the use of variants previously on multiple processing elements (PEs) or cores of a chip to discover Trojans but they did not explore it for other environments (e.g., FPGAs or ASICs).
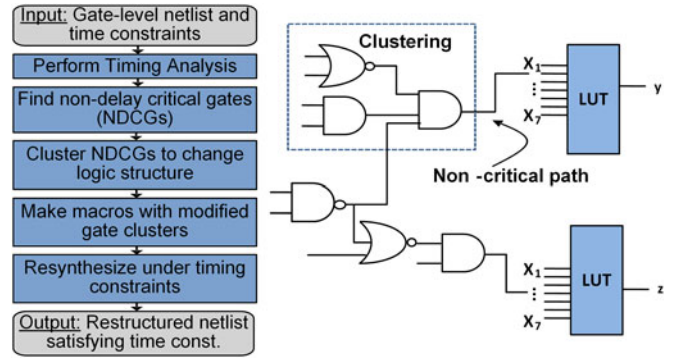
In FPGA, the proposed ATMR scheme can further be improved by implementing variants of adders for O_i and R_i to ensure that O_i and R_i functionally compute the same result, but the adders are implemented in a structurally different manner. This can be achieved by applying different synthesis constraints like area and latency to different instances of the same module. It is highly unlikely that both adders (O_i and R_i) will simultaneously trigger the same Trojan because different implementations involve different logic blocks, storage elements, interconnects, and memory locations. An intelligent attacker is likely to design sophisticated Trojans so that they are triggered by a sequence of rare events or conditions [17]. Thus the probability of triggering the same Trojan by its variants is extremely rare, and we can assume that both O_i and R_i cannot simultaneously trigger the same Trojan.

A judicious design of structural variants can tolerate simultaneous activation of Trojans in both original and replica modules. The dissimilarity between the variants can be exploited to defeat/avoid Trojans and design Trojan tolerant FPGAs. The variants would differ in both LUT and programmable interconnect structures while maintaining the functional behavior and parametric specifications (e.g., critical path delay). One possible approach to making variants is finding non-delay critical gates and regrouping them. It ensures that the content of the LUT and the layout of the design will be changed while the delay remains the same. We regroup the logic gates to change both LUT content and their interconnection possibly changing the required number of LUT resources. Next, we convert the regrouped LUTs as "hard macros" (so that they are not optimized again by the synthesis tool). The last step is to resynthesize the circuit including the macros with the original time constraints. Thus even when both replicas are infected with a Trojan, the Trojan is very unlikely to be activated in both the original and replica modules simultaneously. Thus, the proposed scheme can overcome the limitation of TMR in that it can protect the circuit from multiple Trojan effects by employing variants of a processing unit. Fig. 12 shows the flow chart for implementing the variants.

Exploiting Reconfigurability to Reduce Overhead

Many real-time applications do not require the highest level of security and spare resources can be time-shared among the faulty modules. For example, instead of using a dedicated third redundant component, n number of redundant
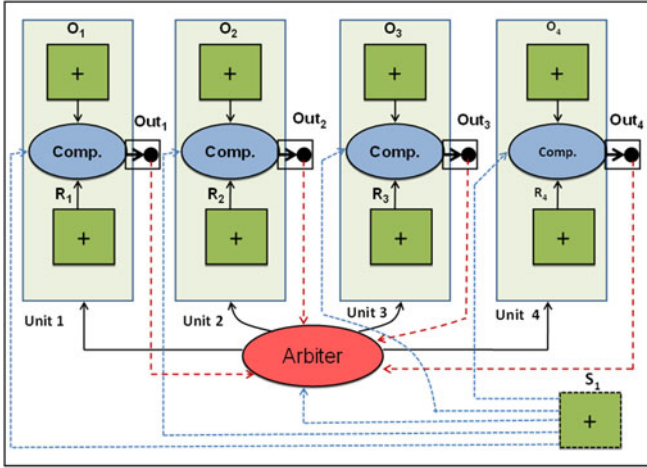
Fig. 13. Sharing a spare resource by a set of homogeneous resources to reduce the hardware overhead.

components can be shared among m $(n < m)$ number of faulty modules, depending on the required level of security. Then the arbiter can select the component on demand allowing efficient and flexible use of resources. Reconfiguration also allows time-sharing of hardware components. Fig. 13 shows several modules and their time sharing of spare resources.

Different Trojan triggering scenarios and their Trojan tolerance are discussed below.

Case 1. If only one adder triggers Trojans, then the Trojan infected adder can be detected by using the spare adder S_1 as explained in Fig. 11.

Case 2. If two adders trigger Trojans, then the following cases can occur.

Case 2a: When two adders are from the same module, the set of adders that are affected by similar Trojan instances = $\{O_i, R_i\}$, where $i = 1, 2, 3, or 4$.

Case 2b: When two adders are from different modules, the set of adders that trigger Trojans = $\{O_i, R_j\}$, where $i, j = 1, 2, 3, or 4 and i \neq j$.

Case 2a can be mitigated by using the variants of adders for O_i and R_i. It is highly unlikely that both adders (O_i and R_i) will simultaneously trigger the same Trojan because different implementations involve different logic blocks, storage elements, logic structures, and memory locations. Thus a judicious design of structural variants can avoid simultaneous activation.

Case 2b cannot be handled by using only one spare adder with the highest level of security. Two spare adders S_1 and S_2 may be required in order to achieve the highest level of trust in the case of two simultaneous failures as in Case 2b (in general, to tolerate n number of simultaneous failures

with $n \leq max \# of modules$, we need n spares). Each spare resource is used to determine and replace a faulty adder in each module.

Worst-Case Scenario. If multiple adders from multiple modules trigger Trojans simultaneously, the set of adders that trigger Trojans = $\{X_1, X_2, X_3, X_4\}$, where $X = O or R$.

In the worst-case scenario, the highest level of security can be achieved at the expense of four spare adders because different implementations for O_i and R_i ensure that both adders will not trigger the same Trojans at the same time.

Thus the number of spare resources (S_i) varies from zero to four to achieve the highest level of security, depending on the number and the location of Trojan affected adders (Fig. 13). However, many real-time applications do not require the highest level of security and S_i can be time-shared among the faulty modules. The proposed hybrid scheme that combines adapted TMR with reconfiguration has the potential to reduce the overhead/power consumption associated with TMR by decreasing the usage of redundancies while allowing reconfiguration. The proposed scheme can also overcome the limitation of TMR in that it can protect the circuit from multiple single event upsets (SEUs) by employing variants of adders.

SIMULATION RESULTS

In this section we present case studies with a commercial FPGA device and a benchmark circuit to illustrate the effectiveness of the proposed Trojan tolerance approach.

We consider the Altera Cyclone IV GX FPGA family of devices and a 32-bit pipelined DLX processor benchmark to analyze the effectiveness of proposed approach. First, we present the mapping results for the execution unit of the processor. We consider several instances of the execution unit. Next, we present the mapping results for the entire DLX processor. Area, delay and power overhead due to the proposed tolerance mechanism are compared with those for conventional TMR.

Table 1 compares the overhead, in terms of power, resources, and performance of conventional TMR and the proposed hybrid tolerance scheme without variants, with variants, and with time-shared resources. For cases both with and without variants, ATMR consumed $1.5\times$ less power than TMR to achieve the same level of security while maintaining equivalent resource usage and performance as TMR. This result is expected given that ATMR, unlike TMR, uses the third spare resources/replicas only when Trojans are activated and a mismatch occurs in the initial outputs. Similar latencies for TMR and ATMR indicates that ATMR does not incur a significant performance overhead. In the case of time-shared resources, ATMR requires $1.4\times$ fewer

TABLE 1
Comparison of Design Overhead between TMR and ATMR without Variants, with Variants, and with Time-Shared Resources

	Without Variants			With Variants			Time-shared Res.		
	TMR	ATMR	$\times$ Impr.	TMR	ATMR	$\times$ Impr.	TMR	ATMR	$\times$ Impr.
Power (mW)	4.70	3.15	$1.5\times$	4.95	3.26	$1.5\times$	16.0	12.42	$1.3\times$
Area (LE*)	850	856	$1.0\times$	860	872	$1.0\times$	3,166	2,184	$1.4\times$
Latency (ns)	6.7	6.7	$1.0\times$	6.4	6.4	$1.0\times$	7.9	7.9	$1.0\times$

*Logic Elements, as reported by the Altera Quartus II FPGA mapping tool

TABLE 2
Comparison of Design Overhead for TMR and ATMR
at the Component (C) and ALU (A) Levels of Granularity

	Area (LEs)	Lat. (ns)	Pow. (mW)	× Impr. (Area)	× Impr. (Lat.)	× Impr. (Pow.)
ALU	225	6.4	1.4	-	-	-
TMR-C	1,928	10	7.05	8.6×	1.6×	7.9×
ATMR-C	1,940	10	5.1	8.6×	1.6×	5.7×
TMR-A	860	6.4	4.95	3.8×	1.0×	3.5×
ATMR-A	872	6.4	3.26	3.9×	1.0×	2.3×

resources than TMR, while consuming $1.3\times$ less power, and maintaining comparable performance.

DISCUSSION

In this section, we discuss several relevant issues related to FPGA security and the proposed Trojan tolerance approach.

Trojans in the Control/Arbiter or Comparator Logic

Most research on TMRs assumes that the voter circuits are hardened structures, and therefore not vulnerable to hardware Trojans. This assumption is not always true, leaving the TMR system itself unprotected. Kshirsagar and Patrikar [23] proposed a novel fault-tolerant voter circuit for TMR that improved the reliability of the digital systems. Ban and Lirida [28] further enhanced the reliability of a system by providing an alternative architecture for a majority voter in TMR. However, their architectures are robust to single fault but cannot handle multiple failures.

The proposed scheme can overcome the limitation of TMR, since it can protect the circuits from multiple faults by employing variants. We suggest two techniques to determine if the arbiter or the comparator in the proposed architecture is compromised: (1) majority voting, or (2) exhaustively testing the comparator and arbiter circuits for the presence of Trojans. The first approach would improve the reliability of the system majority

voting on the voters/arbiters will result in high overhead. The second approach, i.e. exhaustive testing for Trojans, is feasible in this case because the voter/arbiter circuits are relatively small with limited input space, making this the preferred method.

Trojan in the Design (FPGA IP)

Hardware Trojans can also be inserted into the RTL or netlist of the design IP. With the increasing complexity of hardware designs, it is becoming increasingly common for companies to purchase IP blocks, saving on the design and verification effort. The prevalence of IP vendors makes IP Trojan insertion a very real concern for real-world applications. The proposed approach, which aims to tolerate Trojans in the physical hardware, can be combined with Trojan verification and/or tolerance approaches [20] for the IP. Additional, redundant hardware, such as that used in hot-swapping [31] or hot-spares [31] can provide additional fault tolerance against some Trojan attacks and increase reliability for mission-critical systems. to provide comprehensive protection against Trojan attacks.

Granularity for ATMR

While selecting the granularity for designing ATMR, there is a trade-off among the controller/comparator overhead, the number of Trojans, reconfiguration complexity, and the delay in Trojan detection. The advantages to finer-granularity modules include fewer Trojans, simplicity in reconfiguration, and accuracy in Trojan detection. However, fine-grained detection will result in high controller/comparator overhead, increased simulation time, and increased latency of the entire system, which makes it hard to track the system-level propagation of Trojans. On the other hand, coarse-grained modules could result in multiple Trojans affecting the same circuit and increased complexity in reconfiguration, but less controller/comparator overhead and delay. Table 2 shows the area/delay/power overheads for component level and ALU level TMR and ATMR approaches. While component level TMR/ATMR provides more security, it increases reconfiguration

TABLE 3
Comparison of Design Overhead for TMR and ATMR Correction Techniques on a DLX Processor

	Area (LEs)	Flipflops	RegFile (bits)	Delay (ns)	Power (mW)	× Impr. (Area)	× Impr. (Lat.)	× Impr. (Pow.)
IF	199	128	0	5.6	6.89	-	-	-
TMR IF	813	384	0	5.7	20.1	4.1×	1×	2.9×
ATMR IF	835	384	0	5.7	11.4	4.2×	1×	1.7×
ID	222	128	2,048	4.9	11.96	-	-	-
TMR ID	988	384	6,144	4.9	33.96	4.5×	1×	2.8×
ATMR ID	1,011	384	6,144	4.9	21.4	4.6×	1×	1.8×
EX	1,512	133	0	18	23.0	-	-	-
TMR EX	4,747	399	0	19.7	71.98	3.1×	1.1×	3.1×
ATMR EX	4,770	399	0	19.7	38.03	3.2×	1.1×	1.7×
MEM	105	101	0	5	2.03	-	-	-
TMR MEM	487	303	0	5	11.05	4.6×	1×	5.4×
ATMR MEM	510	303	0	5	5.5	4.9×	1×	2.7×
WB	44	0	0	4	1.24	-	-	-
TMR WB	186	0	0	4	3.09	4.2×	1×	2.5×
ATMR WB	209	0	0	4	2.05	4.8×	1×	1.7×
DLX	2,082	490	2,048	18	29.04	-	-	-
TMR DLX	7,221	1,470	6,144	19.7	89.7	3.5×	1.1×	3.1×
ATMR DLX	7,244	1,470	6,144	19.7	48.5	3.5×	1.1×	1.7×

latency and overall system latency ($1.6\times$), while incurring greater area ($8.6\times$) and power ($5.7\times$) overhead. Conversely, ALU-level TMR and ATMR provide less security, but incur smaller area ($3.9\times$) and power ($3.5\times$) overhead, while delay remains unchanged; again, the power consumption for ATMR is further reduced ($2.3\times$) because of the conditionally-enabled third instance. These tradeoffs enable designers to fine-tune the system security while considering relevant overheads in the resultant hardware.

CONCLUSION

Malicious alterations to a design are possible at various stages of the design flow. In this paper, we focused on malicious changes that can be inserted into FPGA devices during production. We presented a taxonomy of hardware Trojan models specific to FPGAs, including those that cause logical malfunctions and/or physical damage, and can be inserted by an attacker in the foundry without knowledge of the final design. We explained multiple detection strategies that could be used to non-invasively test FPGAs to identify the presence of Trojans. As FPGAs are being increasingly used in a wide field of applications, the hardware Trojan models must be fully understood before reliable detection strategies can be developed to provide hardware assurance. In addition, we proposed a novel Trojan tolerance scheme, namely ATMR, that protects FPGA devices against Trojans of varying sizes and functionalities. We compared the proposed scheme with the conventional TMR approach and demonstrated that ATMR requires less power overhead, while maintaining the same or higher level of security and performances as TMR. Further improvements in overhead have been achieved by exploiting reconfiguration and reuse of the resources in ATMR without compromising security.

ACKNOWLEDGMENTS

REFERENCES

[1] *DARPA, TRUST in Integrated Circuits (TRUST)*, 2008. [Online]. Available: http://www.darpa.mil/mto/programs/trust/index.html

[2] S. Trimberger, "Trusted design in FPGAs," in *Proc. 44th Des. Autom. Conf.*, 2007, pp. 5 8.

[3] I. Hadzic, S. Udani, and J. Smith, "FPGA viruses," in *Proc. 9th Int. Workshop Field Programmable Logic Appl.*, 1999, pp. 291 300.

[4] S. Drimer, "Volatile FPGA Design Security: A Survey," Cambridge Univ., Cambridge, U.K., Tech. Rep. 763, 2008.

[5] T. Huffmire, et al., "Handbook of FPGA design security," Springer, 2010.

[6] X. Wang, M. Tehranipoor, J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Proc. IEEE Int. Workshop Hardware Oriented Security Trust*, 2008, pp. 15 19.

[7] S. Trimberger, "Method and Apparatus for Protecting Proprietary Decryption Keys for Programmable Logic Devices," U.S. Patent 6 654 889, Nov. 25, 2003.

[8] *Aletar: Military Anti Tampering Solutions Using Programmable Logic*. [Online]. Available: http://www.altera.com/literature/cp/CP 01007.pdf

[9] L. Lin, W. Burleson, C. Paar, "MOLES: Malicious off chip leakage enabled by side channels," in *Proc. Intl. Conf. Comput. Aided Des.*, 2009, pp. 117 122.

[10] A. Schropp "Non destructive and quantitative imaging of a nano structured microchip by ptychographic hard X ray scanning microscopy," *J. Micros.*, vol. 241, pp. 9 12, 2010.

[11] J. Soden, R. Anderson and C. Henderson, "IC failure analysis tools and techniques magic, mystery, and science," in *Proc. Int. Test Conf.*, pp. 1 11, 1996.

[12] V. Nagabudi, "Extraction Based Verification Method for off the Shelf Integrated Circuits," M.S. thesis, Dept. Elect. Eng. Comput. Sci., Case Western Reserve Univ., Cleveland, OH, 2010.

[13] K. K. Yu and C. N. Berglan, "Automated System for Extracting Design and Layout Information from an Integrated Circuit," U. S. Patent 5 086 477, Feb. 4, 1992.

[14] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojandetection," in *Proc. 11th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2009, pp. 396 410.

[15] H. Kubatova and P. Kubalik, "Fault tolerant and fail safe design based on reconfiguration," *Chapter 8, Design and Test Technology for Dependable System on Chip*, IGI Global, 1st Edition, pp. 175 194, 2011.

[16] N. Gaitanis, "The design of totally self checking TMR fault toler ant systems," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1450 1454, Nov. 1988.

[17] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Tro jan: Treats and emerging solutions," in *Proc. IEEE Int. High Level Des. Validation Test Workshop*, 2009, pp. 166 171.

[18] M. Tehranipoor and F. Koushanfar, "A survey of hardware Tro jantaxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10 25, Jan./Feb. 2010.

[19] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of vertex FPGA TMR design method ology," in *Proc. 6th Eur. Conf. Radiat. Effects Compon. Syst.*, 2001, pp. 275 282.

[20] M. Abramovici and P. Bradley, "Integrated Circuit Security New Threats and Solutions," in *Proc. 5th Annu. Workshop Cyber Security Inf. Intell. Res.*, 2009, pp. 1 3.

[21] S. Srinivasan, A. Gayasen, N. Vijaykrishnan, M. Kandemir, Y. Xie, and M. J. Irwin, "Improving soft error tolerance of FPGA configu ration bits," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2004, pp. 107 110.

[22] D. McIntyre, F. Wolff, C. Papachristou, and S. Bhunia, "Trustworthy computing in a multi core system using distributed scheduling," in *Proc. IEEE 16th Int. On Line Test. Symp.*, 2010, 211 213.

[23] R.V. Kshirsagar and R.M. Patrikar, "Design of a novel fault tolerant voter circuit for TMR implementation to improve reliability in digi tal circuits," *Microelectron. Reliab.*, vol. 49, pp. 1573 1577, Dec. 2009.

[24] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan free trusted ICs: Problem analysis and detection scheme," in *Proc. Conf. Des. Autom. Test Eur.*, 2008, pp. 1362 1365.

[25] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojandetection using IC fingerprinting," in *Proc. IEEE Symp. Security Priv.*, 2007, pp. 296 310.

[26] M. Rebaudengo "Analysis of SEU effects in a pipelined processor," in *Proc. 8th IEEE Int. On Line Test. Workshop*, 2002, pp. 112 116.

[27] R. Torrance and D. James, "The state of the art in IC reverse engi neering," in *Proc. 11th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2009, pp. 363 381.

[28] T. Ban and L. A. B. Naviner, "A simple fault tolerant digital voter circuit in TMR nanoarchitectures," in *Proc. 8th IEEE Int. NEWCAS Conf.*, 2010, pp. 269 272.

[29] T.J. Bauer and S.P. Young, "FPGA Architecture with Deep Look up Table RAMs," U.S. Patent 6 288 568 B1, Sep. 11, 2001.

[30] Y. Lin, C. Zhang, D. Jefferson, and S. Reddy, "Memory Array Operating as Shift Registers," U.S. Patent 6 288 568 B1, 2006.

[31] J. M. Bearfield, Introduction to Hot Swap, Nov. 2011. [Online]. Available: http://eetimes.com/design/analog design/4018093/ Introduction to Hot Swap

[32] K. Xiao and M. Tehranipoor, "BISA: Built in self Authentication for preventing hardware Trojaninsertion", in *Proc. IEEE Int. Work shop Hardware Oriented Trust Security*, 2013, pp. 45 50.

[33] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. 22nd USENIX Conf. Security*, 2013, pp. 495 510.

[34] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhu nia, "Improving IC security against Trojanattacks through integra tion of security monitors," *IEEE Des. Test Comput.*, vol. 29, no. 5, pp. 37 46, Oct. 2012.

[35] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, "Design and analysis of ring oscillator based design for trust technique," in *Proc. 29th VLSI Test Symp.*, 2011, pp. 105 110.

[36] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel tech nique for improving hardware Trojandetection and reducing Trojanactivation time," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 20, no. 1, pp. 112 125, Jan. 2011.

[37] M. Banga and M. Hsiao, "ODETTE: A non scan design for test methodology for Trojandetection in ICs," in *Proc. IEEE Int. Work shop Hardware Oriented Trust Security*, 2011, pp. 18 23.

[38] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant level hardware Trojans: Extended version," *J. Crypto graphic Eng.*,vol. 4, pp. 19 31, 2014.

[39] S. Wei and M. Potkonjak, "Scalable hardware trojandiagnosis," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 20, no. 6, pp. 1049 1057, Jun. 2012.

[40] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identi fication of stealthy malicious logic using Boolean functional analy sis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 697 708.

[41] N. Yoshimizu, "Hardware Trojandetection by symmetry breaking in path delays," in *Proc. IEEE Int. Symp. Hardware Oriented Security Trust*, 2014, pp. 107 111.