Enhancing Trust in Reconfigurable Based Hardware Systems with Tags and Monitors

Devu Manikantan Shila, Vivek Venugopalan
United Technologies Research Center
411 Silver Lane, E Hartford, CT USA
Email:{manikad, venugov}@utrc.utc.com

Cameron D Patterson
Bradley Department of ECE, Virginia Tech
Blacksburg, VA, USA
Email: cdp@vt.edu

Abstract

Extensive use of third party IP cores (e.g., HDL, netlist) and open source tools in the FPGA application design and development process in conjunction with the inadequate bitstream protection measures have raised crucial security concerns in the past for reconfigurable hardware systems. Designing high fidelity and secure methodologies for FPGAs are still infancy and in particular, there are almost no concrete methods/techniques that can ensure trust in FPGA applications not entirely designed and/or developed in a trusted environment. This work strongly suggests the need for an anomaly detection capability within the FPGAs that can continuously monitor the behavior of the underlying FPGA IP cores and the communication activities of IP cores with other IP cores or peripherals for any abnormalities. To capture this need, we propose a technique called FIDelity Enhancing Security (FIDES) methodology for FPGAs that uses a combination of access control policies and behavior learning techniques for anomaly detection.

FIDES essentially comprises of two components: (i) *Trusted Wrappers*, a layer of monitors with sensing capabilities distributed across the FPGA fabric; these wrappers embed the output of each IP core i with a tag τ_i according to the pre-defined security policy Π and also verifies the embeddings of each input to the IP core to detect any violation of policies. The use of tagging and tracking enables us to capture the generalized interactions of each IP core with its environment (e.g., other IP cores, memory, OS or I/O ports). *Trusted Wrappers* also monitors the statistical properties exhibited by each IP core functions on execution such as power consumption, number of clock cycles and timing variations to detect any anomalous operations; (ii) a *Trusted Anchor* that monitors the communication between the IP cores and the peripherals with regard to the centralized security policies Ψ and the statistical properties produced by the peripherals. We target FIDES architecture on a Xilinx Zynq 7020 device for a red-black system comprising of sensitive and non-sensitive IP cores. Our FIDES implementation leads to only 1-2% overhead in terms of the logic resources per wrapper but 4-5X increase in latency (worst case scenario), measured in terms of clock cycles, as compared to the baseline implementation.

Index Terms

Design; Security and Trust; Hardware Trojans; FPGAs

I. Introduction

Due to the size, weight and power (SWaP) advantages, FPGA (Field Programmable Gate Array) devices are the primary source for both computation and communication tasks in various mission and safety critical embedded systems such as smart grid, embedded and network encryption, avionics, tactical radios, satellite communications, finance and banking, and homeland security. While the benefits of FPGAs are great, hardware security is now emerging as a serious concern for FPGAs due to the extensive use of third party IP cores/functions (e.g., HDL, netlist), open source tools (e.g., CAD) and unauthenticated configuration bitstreams in the application design and development process. For instance, efforts [1], [2], [3] show how one can exploit various vulnerable stages of the hardware lifecycle to inject stealthy malwares ('Trojans') into the ASIC and FPGA devices. Evidence of the existence of Trojans in the forms of backdoors in military grade and wireless communication devices are also shown in [4]. Typically, an FPGA application encompass a collection of intellectual property (IP) cores and soft-core processors which are in turn glued together to implement specific functionalities within a system. In many cases, the IP cores (available in the form of HDL, netlist or bitstream) are procured from multiple third-party vendors with varying trust levels which provides ample opportunities for an adversary to inject hardware Trojans. A successful triggering of one such Trojan could create chaos in civilian infrastructure (aerospace, transportation or energy domain), sabotage critical military applications and missions, disable missile and weapon systems, leak sensitive information or provide backdoor access to highly secure systems. Enforcing hardware security at the device level and system level is very critical which in turn has motivated defense agencies such as DARPA and IARPA to launch innovative programs such as Supply Chain Hardware Integrity for Electronics Defense (SHIELD) [5] and Trusted Integrated Chips (TIC) [6].

Besides these programs, a handful of research efforts along the tangents of detecting and identifying the Trojans in hardware have been proposed. The techniques presented in works such as UCI [7], VeriTrust [8], FANCI [9] and HaTCh [10] essentially focus on the idea of identifying and flagging unused or suspicious Trojan wires within a design. In HaTCh [10], for instance, authors use functional testing on the IP cores to identify all the list of unused wire combinations and augments additional tagging circuitry to the IP cores so as to keep track of the suspicious wires; as soon as a malicious wire is activated, the tagging circuitry will raise an exception to prevent the Trojan from depicting malicious behavior. A major drawback with all these approaches is the assumption that the source code of the IP core design is available to the end consumers in HDL/netlist format and hence, often neglect the scenarios where the consumers have access only to the configuration bitstreams ('hard IP cores'). Proof-Carrying Codes [11] are also proposed where the IP vendor will construct a formal proof of the design adhering to certain security properties, which will be later verified by IP consumer to ensure that design is free from modifications. Besides the overhead involved in creating huge proofs for even smaller codes, loop holes can also occur while defining many security properties.

In contrary to above efforts, the concepts presented in [12], [13], and [14] assumes a 'blackbox' methodology where no assumption is made on the availability of the source code and are perhaps closest to our design. In [12], authors propose an application-dependent security infrastructure monitoring datapath signals for illegal behaviors. This approach adds reconfigurable Design-for-Enabling-Security (DEFENSE) logic to the functional design to implement a centralized run-time security monitor. The hardware-based monitors are configurable finite-state machines (FSMs) that check the current set of signals for the properties specified by the designer. Huffmire *et al.* [13] propose a memory protection mechanism by designing and implementing policy-driven memory protection techniques using reconfigurable hardware. Their work develops an access policy language that precisely describes the fine-grained memory separation of modules on an FPGA and a policy compiler that converts the specified memory access policies into enforcement hardware modules. This memory access policy mechanism is then integrated into the hardware-based centralized reference monitor to detect malicious memory accesses at run-time. The use of centralized monitors not only affects the scalability of the security mechanisms in [15], [16], [17] but also makes it vulnerable to classic communication attacks such as spoofing, Man-in-the-Middle (MITM) that enables an attacker to spoof or tamper the signals/messages to the centralized monitor, as indicated by [18], [19]. Another drawback of these approaches is the inability to identify the malicious IP cores that conforms to the access control policies but deviates from the normal execution behavior; for e.g., a malicious encryption IP core instead of encrypting the data when triggered leaks the secret key. In such scenarios, monitoring the normal execution behavior of IP cores and peripherals such as number of clock cycles, total power consumption in real-time can be used to detect any abnormalities.

Motivation and Contribution. The addition of third party IP cores and the use of open source tools in the design and development process renders an FPGA application vulnerable to hardware Trojan based attacks. There are currently no concrete methods/techniques that can be easily instrumented into the FPGAs and provide trust guarantees in FPGA applications which are not completely developed in a trusted environment. The security methods currently available have the following shortcomings: (a) requires low level knowledge of the IP core design; (b) vulnerable to traditional communication attacks; (c) centralized approach; and (d) inability to detect functional behavior deviation. Moreover most existing solutions overlook the communication activities and functional behavior of IP core which in turn plays an important role in actually carrying out the attack. This paper addresses such a significant loophole by proposing a secure communication framework and behavior monitoring module that can reliably thwart hardware Trojan based attacks.

Alternatively speaking, motivated by the shortcomings in the existing works, we strongly argue the need for an anomaly detection capability within the FPGAs that can be easily instrumented with the application and continuously monitor the underlying IP core behavior and also the access activities of IP cores with other IP cores or peripherals for any signs of abnormalities. To capture this need, we propose a technique called FIDelity Enhancing Security (FIDES) methodology for FPGAs that uses a combination of access control policies and behavior learning techniques for anomaly detection. FIDES essentially comprises of two components: (i) *Trusted Wrappers*, a layer of monitors with sensing capabilities distributed across the FPGA fabric; these wrappers stamp the output of each IP core i with a tag τ_i according to the pre-defined security policy Π_i and also verifies the embeddings of each input to the IP core to detect any violation of policies. The use of tagging and tracking enables us to capture the generalized interactions of each IP core with its environment (e.g., other IP cores, memory, OS or I/O ports). *Trusted Wrappers* also monitors the statistical properties exhibited by each IP core functions on execution such as power consumption, number of clock cycles and timing variations to detect any anomalous operations; (ii)a *Trusted Anchor* that monitors the communication between the IP cores and the peripherals with regard to the centralized security policies Ψ and the statistical properties produced by the peripherals.

We prototyped FIDES architecture on a Xilinx Zynq 7020 device, that consists of a hard fused ARM core and an Artix FPGA. The $\mathcal{TA}$ is mapped to the ARM core that controls the communication of the IP cores with the external peripherals (enabled to thwart leakage of sensitive information) and provides behavioral monitoring of the peripherals (enabled to thwart covert attacks). Our FIDES implementation, in worst case, yields a 1-2% increase in logic resource utilization per trusted wrapper and 4-5 times increase in latency (worst case), measured in terms of clock cycles, as compared to the baseline implementation.

To the best of our knowledge, we first design and implement a distributed and secure anomaly detection framework for FPGAs that can identify abnormal FPGA behaviors (hardware Trojans) in real-time.

Roadmap: Section II provides the detailed description of the FIDES methodology along with the description of the adversary/threat model and the system design of the distributed information model. The hardware implementation of the FIDES architecture is described in Section III with the results presented in Section IV. Section V concludes this work.

II. Proposed Security Framework

This paper proposes a novel, distributed and secure methodology for enhancing trust in FPGA applications designed or developed under untrusted environments called FIDES (FIDelity Enhancing Security methodology) named after the Greek God of Trust. FIDES essentially relies on the use of two major approaches: (a) *decentralized information flow control* (DIFC) model that enables safe and distributed information flows between various elements of FPGA such as IP cores, physical memory and registers by annotating/tagging each data item with its sensitivity level and the identity of the participating entities; and (b) *statistical learning techniques* that learns the normal functional behavior of IP cores, peripherals defined as *conformant core behavior* ($\mathcal{CCB}$) during the FPGA application integration/testing stage and leverages the learned $\mathcal{CCB}$ model to detect any anomalous behavior deviations in run-time.

In the sequel, we will initially present the adversary and threat model and then discuss the detailed FIDES system design in section II-B. Table I presents the summary of various notations used for describing the FIDES design.

Table I: Table of Notations

Notation	Meaning
$<Core>_i^s$	Sensitive IP core i
$<Core>_i^{ns}$	Non-sensitive IP core i
W_i	Trusted Wrapper assigned to $<Core>_i$
$\mathcal{TA}$	Trusted Anchor
Π_i	User-defined policies for $<Core>_i$
Ψ	User-defined policies at $\mathcal{TA}$
τ_i	Tags assigned to $<Core>_i$
σ_i	$<Core>_i$ functional behavior signature
$\mathcal{M}^{\mathcal{S}}$,$\mathcal{M}^{\mathcal{N}}$	Sensitive and Non-sensitive memory regions
α	Characteristics of IP core ($\mathcal{S}, \mathcal{N}, \mathcal{E}$)

A. Adversary and Threat Model

Majority of the efforts show that various layers of the FPGA lifecycle ranging from design to final deployment stage are vulnerable to Trojan intrusions. This work assumes trusted FPGA application integration and testing stage however untrusted design, development and deployment stage due to the immense involvement of the third party IP cores, CAD tools and also, easily accessible FPGA configuration bit streams. With this assumption, a more realistic adversary model is considered: (i) at design and development stage, the Trojan might be present in the third party IP core or CAD tool, or might be inserted by a rogue designer during the design process; (ii) at deployment stage, the Trojan might be injected into the FPGA configuration bitstream. A handful of works along the lines of cryptographic defenses (e.g., use of bitstream signatures [20], design obfuscation [21]) have been proposed to thwart tampering attacks on FPGA bitstream. FIDES architecture will leverage those existing defenses based on asymmetric cryptography.

Embedded systems often contains a mixture of critical and non-critical applications residing on the same platform. Any intentional or accidental fault on the operation of the critical application could have an impact on safety, or could cause large financial or social loss. It is therefore necessary to thwart the non-critical components from interrupting or compromising the critical operations. This paper considers a similar scenario called the red-black system implemented in FPGA that will encompass multiple IP cores procured from various third party vendors with different sensitivity levels. In such a system, two distinct kinds of IP core ('Core') sensitivity levels are considered: (a) IP core that accesses and processes critical and sensitive information (e.g., network encryption) called *sensitive core* ($<Core>_i^s$) and (b) IP core that accesses and processes less critical and sensitive information called *non-sensitive core* ($<Core>_i^{ns}$).

Proper segregation between the sensitive and non-sensitive operations should be realized so that an untrusted IP core will be thwarted from accessing the shared resources (e.g., memory, peripherals) and transmit the information to the outside world via a covert channel (e.g., secret unnoticeable channels) or a normal channel (e.g., send sensitive cryptographic information by mixing with the noise levels or encrypting with the attacker-known keys). With regard to the red-black system design, we derive the attack scenarios given by ([A1] - [A4]) and propose the security policy requirements addressed by ([R1] - [R4]).

Attack Scenarios.

A1. $<Core>_i^s$ accesses the critical and sensitive information and export the information to the outside world

A2. $<Core>_i^{ns}$ accesses the critical and sensitive information and export the information to the outside world

A3. $<Core>_i^{ns}$ uses trusted $<Core>_j^s$ as a conduit (a Man-in-the-middle) to access sensitive regions and export the information to the outside world

A4. Malicious IP core spoof as a legitimate core and access application resources

Security Requirements.

R1. $<Core>_i^{ns}$ shall be prevented from accessing and processing the sensitive information

R2. Cores shall be restricted from accessing the peripherals directly to limit export of data.

R3. The information flow between $<Core>_i^{ns}$ and $<Core>_i^{s}$ shall be restricted or performed in the presence of a security officer.

R4. The statistical properties of each $<Core>_i$ shall be measured and monitored to detect any anomalous functional behavior deviations

B. System Design

Initialization phase. During the FPGA application integration/testing stage, the following components will be generated by the end consumer: (a) *Trusted Wrapper* (W_i) - a thin layer of hardware logic that interface with each IP core $<Core>_i$ and manages/monitors the communication activities of $<Core>_i$ with other cores ($<Core>_j$, where $j \neq i$); (b) *Trusted Anchor* ($\mathcal{TA}$) - a centralized security officer, implemented in ARM, that manages/monitors the communication of the IP cores with the external world (I/O Peripherals) according to the pre-defined security policy Ψ; (c) *Tagged Physical Memory* - a partitioned memory with *tags* annotated to the data denoting their sensitivity level; and (d) *Secure Bitstream Blob* - a cryptographically signed bitstream blob containing W_i; $\forall i \in \mathbb{S}$, where $\mathbb{S}$ represents the set of IP cores and $\mathcal{TA}$ modules with the private key (K-) of the end consumer; the signed blob together with the public key (K+) will be loaded into the memory. A hash of the (K+) will be also loaded into the trusted region within the processor to thwart any attempts to modify the bitstream signatures. The use of cryptographic signatures will prevent an adversary from subverting the protection schemes included within the configuration bitstream. The complete architecture of FIDES is presented in Figure 1.

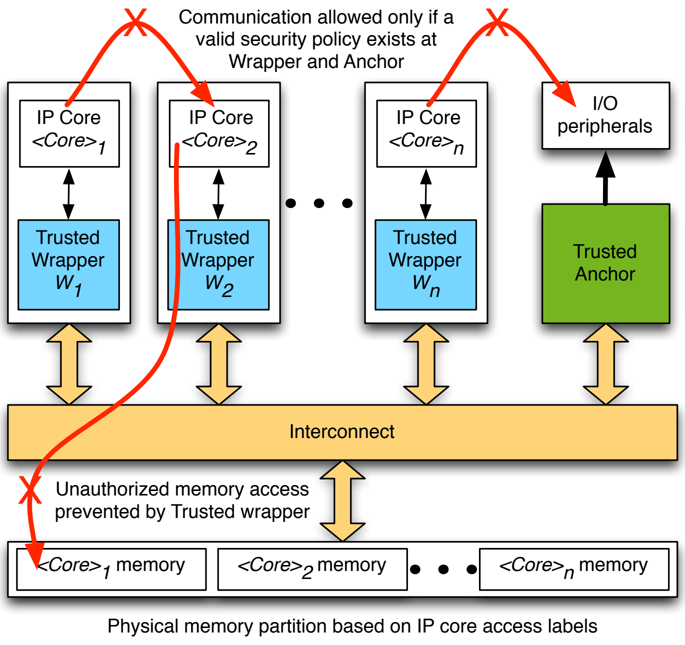
Communication allowed only if a valid security policy exists at Wrapper and Anchor
IP Core <Core>1
IP Core <Core>2
IP Core <Core>n
I/O peripherals
Trusted Wrapper W1
Trusted Wrapper W2
Trusted Wrapper Wn
Trusted Anchor
Interconnect
Unauthorized memory access prevented by Trusted wrapper
<Core>1 memory
<Core>2 memory
<Core>n memory
Physical memory partition based on IP core access labels

Figure 1: FIDES Architecture

1) Distributed Information flow model: FIDES uses tag-based information flow control methodology to achieve decentralized and safe flow of information between various elements in the system [22]. In order to do so, during the initialization phase, each trusted wrapper W_i will be assigned a *tag* (τ_i) that reflects the characteristics (α) of the information handled by the $<Core>_i$ and also the identity (i) of the participating entity (e.g., IP Core identity).

Initialization phase: $W_i \Leftarrow \tau_i;\ \tau_i =< \alpha, i >$

Where $\alpha = (\mathcal{S}, \mathcal{N}, \mathcal{E})$. Here $\mathcal{S}, \mathcal{N}$ and $\mathcal{E}$ denotes sensitive, non-sensitive and empty tokens respectively. The value of α will be assigned based on the following policies:

- $<Core>_j^{s}|W_j \Leftarrow <\mathcal{S}$, j>, $<Core>_j^{ns}|W_j \Leftarrow <\mathcal{N}$, j>: The IP cores $<Core>_i^{s}$ and $<Core>_i^{ns}$ will be labeled with $\mathcal{S}$ and $\mathcal{N}$ tokens, respectively, depending on the characteristics of the information handled. In cases where an IP Core processes both sensitive and non sensitive information, the wrapper will be assigned a $\mathcal{S}$ token.

- Tagged physical memory: The physical memory regions $\mathcal{M}$ will be partitioned into sensitive and non-sensitive regions by tagging with $\mathcal{S}$ and $\mathcal{N}$ tokens denoted by $\mathcal{M}^{\mathcal{S}}$ and $\mathcal{M}^{\mathcal{N}}$, respectively. The memory regions can be also tagged with the identity i of the owner to enable horizontal access control protection, if needed. Alternatively speaking, under horizontal access control protection $<\mathit{Core}>_i^s$ will be prevented from accessing the memory space belonging to $<\mathit{Core}>_j^s$, though both cores are tagged with $\mathcal{S}$.
- $\mathbb{P} \Leftarrow <\mathcal{E}>$: The I/O peripherals (external world), denoted by $\mathbb{P}$, will be labeled with empty tokens ($\mathcal{E}$). This approach of assigning empty tokens will prevent the IP cores from releasing the sensitive information to the external world as tagging based information flow rule requires an IP core to hold an $\mathcal{E}$ token to release the information to the peripherals.

Since the system is distributed and each entity is not aware of the tag of others, the messages exchanged between the entities is labeled appropriately by the trusted wrappers. Then the receiving wrappers will use the tagging based information flow rules to determine whether to accept or reject the messages. Enforcing tags on the messages also ensures that the information will be always prefixed with a tag when stored in temporary or permanent memory locations and only components that follows the tagging-based information flow rules can access the information.

2) Tagging-based information flow rule aka 'DIFC': The information labeled with the tag of $\tau_j = <\alpha_j, j>$ can flow to $\tau_k = <\alpha_k, k>$, only if the tags of τ_j is included in τ_k. We define the partial order $\preceq$ (pronounced can flow to) for two tags τ_j and τ_k as

$$\tau_j \preceq \tau_k; \text{ if } \alpha_j = \alpha_k \text{ and } (j,k) \in \Pi_k$$

Where Π_k is the security policy defined at the IP core k. The security policy Π_i essentially captures the authorized set of communications for each IP core i based on the design specifications. The above equation mainly states that core k can accept the data from core j only if the security policy defined at k (Π_k) allows the communication from core j and the characteristics token α at both j and k matches. Under this rule the following communication will be prevented.

$$\begin{aligned}
\tau_j = (<\mathcal{S}, j>) \npreceq \tau_k = (<\mathcal{N}, k>) \\
\tau_j = (<\mathcal{N}, j>) \npreceq \tau_k = (<\mathcal{S}, k>) \\
\tau_j = (<\mathcal{S}, j>) \npreceq \mathbb{P} = <\mathcal{E}> \\
\tau_j = (<\mathcal{N}, j>) \npreceq \mathbb{P} = <\mathcal{E}>
\end{aligned}$$

While the above rules prevent the contaminated cores from communicating with other cores or releasing the information, it would also make it implausible for any legal communication activities or any sensitive data to get out of the system. Therefore, DIFC supports *tag declassification* capability (denoted by $\downarrow$) which enables the higher tags to declassify to lower tag to receive information; especially, we consider the following two scenarios:
(a) *IP cores that process both sensitive and non-sensitive information labeled with tag $\mathcal{S}$* - We provide an ability for $W_{i;\forall i \in \mathbb{S}}$ to declassify its tag from $\mathcal{S}$ to $\mathcal{N}$ such that $\mathcal{S} \downarrow \mathcal{N}$. To up the privilege back to $\mathcal{S}$, the $<\mathit{Core}>_i^s|W_i$ will communicate with $\mathcal{TA}$ initially and $\mathcal{TA}$ will set the privilege of W_i back to $\mathcal{S}$, only if security policy at $\mathcal{TA}$ allows. If W_i is allowed to up its privilege without contacting $\mathcal{TA}$, it can be leveraged by a non-sensitive compromised IP core to defeat the system. As an example, consider the attack where a $<\mathit{Core}>_i^{ns}$ uses $<\mathit{Core}>_j^s$ as a conduit (a Man-in-the-middle) to access sensitive memory areas and export the information to the outside world. If in the absence of $\mathcal{TA}$, the $<\mathit{Core}>_j^s$ will declassify to $\mathcal{N}$ to receive data from the $<\mathit{Core}>_j^{ns}$ and up its privilege back to $\mathcal{S}$ token to access the sensitive regions; once accessed, the IP core will declassify its privilege to $\mathcal{N}$ to export the information to $<\mathit{Core}>_j^{ns}$. To avoid such attack scenarios, we impose the following constraint: a W_i can declassify only after receiving an explicit request from $\mathcal{TA}$;
(b) *IP core that exports the data to the external world (I/O peripherals)* - To export the data, declassification privileges are assigned to $\mathcal{TA}$ (declassify the label from $\mathcal{S}/\mathcal{N}$ to $\mathcal{E}$) based on the pre-defined security policies, Ψ at $\mathcal{TA}$. The Ψ essentially captures the authorized set of peripherals for each IP core based on the design specifications.
Remark #1: This paper does not consider the presence of Trojans on interconnect. Existing efforts can be leveraged to build secure and trustworthy interconnects such as the time based techniques in [23] to detect malicious interconnect operations or the use of secure checksums, e.g., Message Authentication Codes (MACs), to prevent the Trojans within the interconnect modifying or spoofing the messages.
Remark #2: The security of the proposed FIDES design depends on the policies defined by the user, Π and Ψ. This paper assumes that the policies are generated by the end consumer based on the design specifications and requirements.

3) Behavior monitoring: Besides labeling and monitoring the communication activities, to boost the trust in an FPGA application, the characteristics of the underlying tagged IP cores and peripherals ($\mathbb{P}$) will be also observed and used by the trusted wrappers for identifying malicious behaviors and decision making (e.g., forward/drop the message, report to $\mathcal{TA}$). This technique allows to identify the malicious IP cores that conforms to the access control policies but deviates from the normal execution behavior; for e.g., a malicious IP core that performs erroneous computations on the input data. As each IP core exhibit different statistical properties on execution (e.g., power consumption, number of clock cycles, and timing variations), trusted

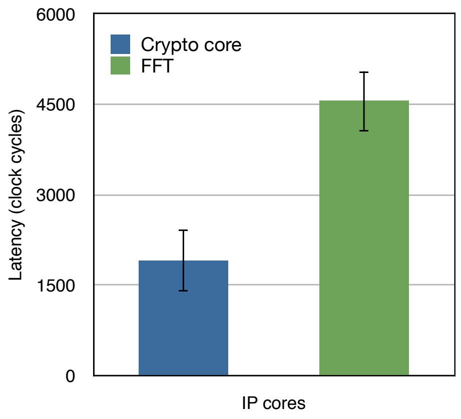

Figure 2: IP cores with distinct latency profiles

wrappers will use these properties to determine the normal functional execution behavior of IP cores defined as *conformant core behavior* ($\mathcal{CCB}$). In Figure 2, we show that each IP core exhibits distinct behavior such as *number of clock cycles* during its execution. Therefore in the FPGA application integration and testing stage, for each <*Core*>$_i$, we compute a signature, denoted by σ_i, that captures these statistical properties. Together with the *tag* (τ_i), during the initialization phase each wrapper W_i will be assigned σ_i as follows:

$$\textbf{Initialization phase: } W_i \Leftarrow \sigma_i$$

In the run-time phase, W_i will monitor the statistical properties produced by <*Core*>$_i$ and compares with the already learned signature σ_i. If the difference between the expected and observed behavior exceeds a user-defined threshold, W_i flags the observation as "anomalous" and informs $\mathcal{TA}$ for further actions. Similar to trusted wrapper, $\mathcal{TA}$ will also monitor the behavior of external peripherals to detect any anomalous leaking of information. The intuition behind using statistical properties for malicious behavior detection comes from [18], [19] where a combination of several statistical properties such as power consumption, clock cycles, called the Hardware Trojan Detection Metric (HDM), can be used to detect malicious IP core operations.

Remark #3: In this paper, we do not address heat or power-based Trojans. However, we can easily extend our work with metrics that can detect these Trojans.

4) Security Proofs: We here show with proofs on how FIDES can detect and prevent the attack scenarios presented in [A1]-[A4].

A1. <*Core*>$_i^s$ accesses the critical and sensitive information and export the information to the outside world using either of the two approaches: (a) covert channel (secret and unnoticeable) and (b) obfuscate (mixing the data with random noise) the information and transport via normal channels:

Proof. (a) <*Core*>$_i^s$ accesses the critical and sensitive information from the memory($\mathcal{M}^{\mathcal{S}}$) and forward the information to the $\mathcal{TA}$. Since the covert channel is not an authorized peripheral for <*Core*>$_i^s$, $\mathcal{TA}$ will reject the information according to the pre-defined policies Ψ. (b) As obfuscation leads to noticeable changes from the learned signature σ_i, wrapper W_i will flag <*Core*>$_i^s$ as anomalous and informs $\mathcal{TA}$ about the observation. □

A2. <*Core*>$_i^{ns}$ accesses the critical and sensitive information and export the information to the outside world

Proof. This attack will fail as tagging based information flow rule restricts <*Core*>$_i^{ns}$ tagged with $\mathcal{N}$ tag from accessing the sensitive memory regions such that $\mathcal{M}^{\mathcal{S}} \npreceq \tau_i =< \mathcal{N}, i >$ □

A3. <*Core*>$_i^{ns}$ uses trusted <*Core*>$_j^s$ as a conduit (a Man-in-the-middle) to access sensitive regions and export the information to the outside world

Proof. As tagging based information flow rule either prevents <*Core*>$_i^{ns}$ tagged with $\mathcal{N}$ tag from accessing the sensitive IP cores ($\tau_i = (< \mathcal{N}, i >) \npreceq \tau_j = (< \mathcal{S}, j >)$) or forces <*Core*>$_j^s$ to declassify its tag from $\mathcal{S}$ to $\mathcal{N}$, this attack fails. □

A4. Malicious IP core spoof as a legitimate core and access application resources

Proof. As each IP core <*Core*>$_i$ is integrated with W_i, all the communications originating from <*Core*>$_i$ will be appropriately labeled with the tag $\tau_i =<\alpha, i>$ which will prevent a malicious IP core from impersonating as a valid IP core. □

III. Hardware Implementation of FIDES

The communication framework in FPGA-based systems plays a critical role in supporting the hardware Trojan attacks and hence it is necessary to understand the existing bus-based protocols in the FPGA platform. FPGA-based systems with soft core processors consisted of interconnects such as Avalon in case of Altera and the processor local bus (PLB) in case of Xilinx. With the introduction of hard fused ARM cores in FPGA devices, AXI interconnect serves as the primary protocol for communication between the ARM core and the different IP cores [24]. The AXI protocol is a subset of the ARM Advanced Microcontroller Bus Architecture (AMBA) and the different AXI4 interfaces are compared in Table II. The selection of AXI protocol allows a designer to focus only on the IP behavior and not the IP interconnect, when targeting different FPGA-ARM platforms. The AXI4-Lite interface is the most area efficient and best suited for control logic data transfer.

Table II: AXI4 interfaces

Features	**AXI4**	**AXI4-Lite**	**AXI4-Stream**
Type	High-performance and memory mapped interface	Register-style interface	Non-address based high speed streaming interface
Data transfer	256 cycles	1 cycle	unlimited cycles
Data width	32-1024 bits	32-64 bits	unlimited bytes
Application	Embedded, memory	Control logic	DSP, video

In case of Xilinx systems, the ARM core is referred to as the processing system (PS), while the FPGA fabric consisting of reconfigurable gates is called the programmable logic (PL). The IP cores are bundled with the AXI4-Lite interface by the IP Packager tool in Xilinx's Vivado Design suite. Each IP core is wrapped with a AXI wrapper, which configures the data transfer into the IP core using custom register interfaces. These memory mapped registers are configured as write registers for input ports and read registers for the output ports of the IP core. The AXI wrapper initializes these registers to a default size of 32 bits. By default, the AXI wrapper is configured to initialize write registers for the IP core's inputs and the IP core's outputs are routed to external I/O peripherals.

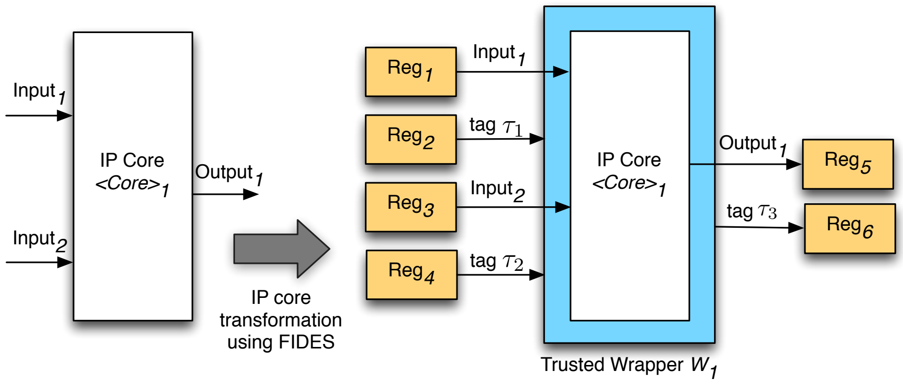

Figure 3: Trusted wrapper mechanism for an IP Core

The trusted wrapper W_1 shown in Figure 3 is the top-level interface to the IP core <*Core*>$_1$. Each of the IP core's I/O port is assigned a tag τ_i, which is mapped to a register from the AXI4 wrapper. Since the IP core's output is also monitored by W_1, the AXI wrapper is modified to incorporate read registers for the output ports. If the output port is directly connected to an external peripheral, the additional read registers for the output port preserves a copy of the data and its tag. Thus the $\mathcal{TA}$ supervises the data being transferred through the external I/O peripherals and is able to provide behavioral monitoring. In summary, the trusted wrapper W_1 consists of the following register configurations:

- `Write` registers for the input ports of the IP core,
- `Write` registers for tagging the input ports,
- `Read` registers for monitoring the IP core's output ports,
- `Read` registers for tagging the IP core's output ports,

- `Read` register for declassification of data transfer depending on the security policy framed for the IP core,
- `Write` register for providing the modified output label based on the temporary downgrade of the security policy access of the IP core.

The trusted wrapper is embedded into the IP core during the Xilinx Vivado tool's IP Packaging and Configuration step. This ensures that the IP core's tagging functionality is available when instantiating the IP core in other designs.

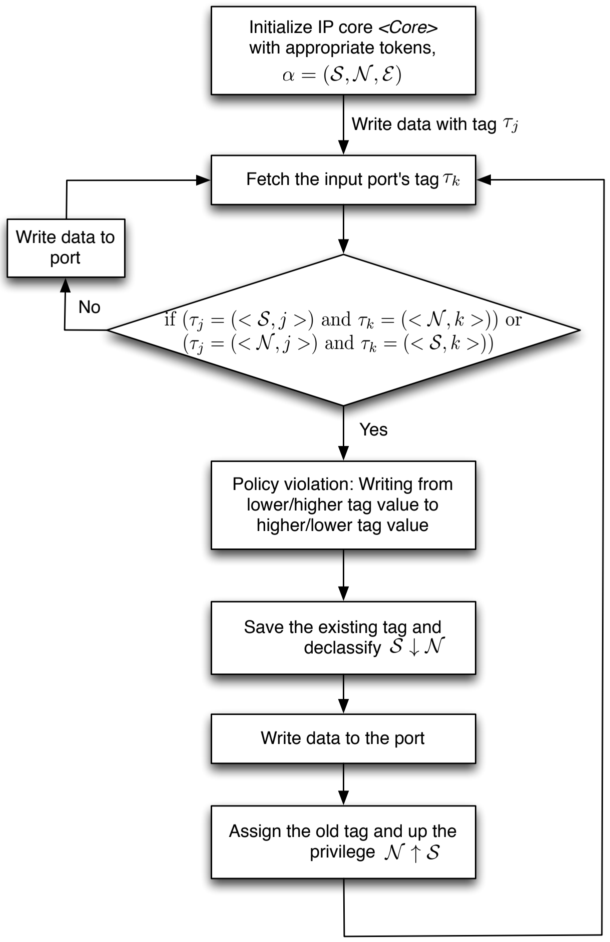
Initialize IP core *<Core>* with appropriate tokens, $\alpha = (\mathcal{S}, \mathcal{N}, \mathcal{E})$
Write data with tag τ_j
Fetch the input port's tag τ_k
Write data to port
No
if $(\tau_j = (<\mathcal{S}, j>)$ and $\tau_k = (<\mathcal{N}, k>))$ or $(\tau_j = (<\mathcal{N}, j>)$ and $\tau_k = (<\mathcal{S}, k>))$
Yes
Policy violation: Writing from lower/higher tag value to higher/lower tag value
Save the existing tag and declassify $\mathcal{S} \downarrow \mathcal{N}$
Write data to the port
Assign the old tag and up the privilege $\mathcal{N} \uparrow \mathcal{S}$

Figure 4: Flowchart of the FIDES algorithm implementation

Figure 4 shows the flowchart of the FIDES algorithm implementation. The first step consists of initializing the IP core with appropriate tokens, $\alpha = (\mathcal{S}, \mathcal{N}, \mathcal{E})$. Each IP core's ports and tags are memory mapped and is accessed by the trusted wrapper W_k to identify the IP core's privilege and the different sensitivities assigned to each of its ports. When data with tag τ_j is being written to the IP core's input port with its initialized tag τ_k, W_k checks the tokens embedded in the tag. These tokens are fetched from the initialization list corresponding to the IP core and its ports. Then W_k checks the following conditions:

$$\tau_j = (<\mathcal{S}, j>) \wedge \tau_k = (<\mathcal{N}, k>)$$
$$\tau_j = (<\mathcal{N}, j>) \wedge \tau_k = (<\mathcal{S}, k>)$$

If any of these conditions are true, the trusted wrapper W_k issues a policy violation message. W_k saves the tag of the sensitive port or data and declassifies it from $\mathcal{S} \downarrow \mathcal{N}$. In case of IP core *<Core>*$_i^s$, W_k checks the register corresponding to the declassification value for the IP core. Once this value is verified as $\mathcal{N}$, W_k writes the data to the port. After the data is written, W_k fetches the previously saved tag and writes it back to the IP core or to the data value, thus changing the privilege from $\mathcal{N} \uparrow \mathcal{S}$. All the declassification and classification of privileges is done in the presence of the $\mathcal{TA}$.

IV. Results

The FIDES algorithm is prototyped on the MicroZed Xilinx Zynq-based Z7020 FPGA development board. The Zynq 7020 consists of a dual core ARM processor and a Xilinx Artix FPGA with 85K logic cells. Both the baseline and the FIDES implementations are synthesized and implemented using Xilinx Vivado tool. The Vivado tool provides the logic resource utilization of the complete design and also provides a definition file. The definition file is mapped using Xilinx SDK to execute the software algorithm implementation. Figure 5 shows the resource utilization obtained from Xilinx Vivado after embedding the IP core in the trusted wrapper. The trusted wrapper only increased the resource utilization by about 1-2% as compared to the baseline implementation.

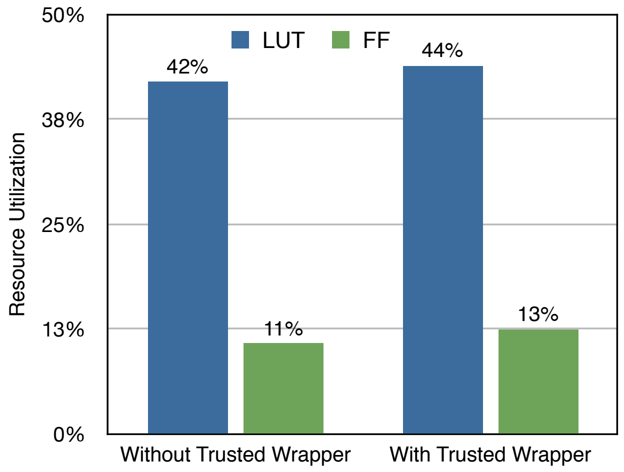

Figure 5: Resource utilization for an IP core with and without a trusted wrapper

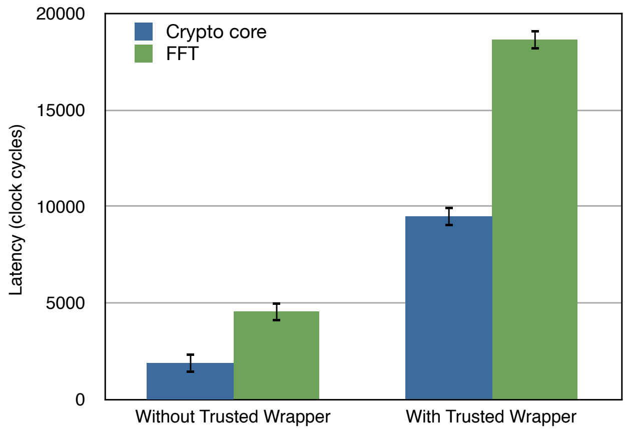

Figure 6: Latency measurement with the trusted wrapper

The trusted anchor $\mathcal{TA}$ is implemented on the ARM core, as it can easily monitor the data transfer through the AXI interconnect to the I/O peripherals, thus enabling behavioral monitoring of peripherals. Although the trusted wrapper does not increase the resource utilization, the latency increases. Figure 6 shows the latency measurement taken for two different IP cores: (a) crypto core, and (b) (64-point) Fast Fourier Transform (FFT) core. The latency measurement for the IP cores is for the worst case scenario when all the inputs have mismatched tags, which requires communication with the $\mathcal{TA}$ for declassification and classification purpose. The latency for the IP cores increased by 4-5 times as compared to the baseline implementation. The latency measurement is from the *Trusted Wrapper* $\rightarrow$ *AXI* $\rightarrow$ $\mathcal{TA}$. Since the data goes through the AXI interconnect, we observe a variation in the clock cycles for the IP cores shown in Figure 6.

V. Conclusions

In this paper, we present the FIDelity Enhancing Security (FIDES) methodology for FPGAs to enhance trust in FPGA operations that are not completely designed or developed in a trusted environment. The security measures currently available for FPGAs have a number of shortcomings such as (a) require low level knowledge of the IP core design; (b) vulnerable to bus communication attacks; (c) centralized design; and (d) failure to detect hardware Trojans that deviate from normal functional behavior. We addressed a significant loophole present in these existing solutions by proposing a secure communication framework and behavior monitoring module that can reliably detect and prevent hardware Trojan based attacks. We implemented FIDES on the Xilinx Zynq 7020 device for a red-black system consisting of critical and non-critical IP cores. Our results yielded a 1-2% increase in hardware resources and 4-5X increase in latency (worst case scenario where all the communications happen between the trusted wrapper and Trusted Anchor) as compared to the baseline implementation. We observe that the resource utilization and latency results can be optimized by clustering the IP cores into groups that share same characteristics, which will be addressed in future work.

VI. Acknowledgments

We would like to thank Prof. Marten van Dijk and his team for fruitful discussions and their helpful feedback.

References

[1] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *Design Test of Computers, IEEE*, vol. 27, no. 1, pp. 10–25, 2010.

[2] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware." *LEET*, vol. 8, pp. 1–8, 2008.

[3] S. Adee, "The Hunt For The Kill Switch," *Spectrum, IEEE*, vol. 45, no. 5, pp. 34–39, 2008.

[4] S. Skorobogatov and C. Woods, *Breakthrough silicon scanning discovers backdoor in military chip*. Springer, 2012.

[5] *Supply chain hardware integrity for electronics defense (SHIELD)*. Defense Advanced Research Projects Agency (DARPA), Microsystems Technology Office/MTO Broad Agency Announcement,, 2014.

[6] *Trusted Integrated Chips (TIC) Program Broad Agency Announcement 11–09*. Intelligence Advanced Research Projects Activity (IARPA), 2011.

[7] M. Hicks, M. Finnicum, S. T. King, M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 159–172.

[8] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 61.

[9] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer &amp; communications security*. ACM, 2013, pp. 697–708.

[10] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, "HaTCh: Hardware Trojan Catcher," *Cryptology ePrint Archive, Report 2014/943*, 2014. [Online]. Available: http://eprint.iacr.org

[11] Y. Jin and Y. Makris, "Hardware Trojans in Wireless Cryptographic ICs," *Design Test of Computers, IEEE*, vol. 27, no. 1, pp. 26–35, 2010.

[12] M. Abramovici and P. Bradley, "Integrated circuit security: new threats and solutions," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM, 2009, p. 55.

[13] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, May 2007, pp. 281–295.

[14] M. Bilzor, T. Huffmire, C. Irvine, and T. Levin, "Security checkers: Detecting processor malicious inclusions at runtime," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 34–39.

[15] T. Huffmire, T. Levin, T. Nguyen, C. Irvine, B. Brotherton, G. Wang, T. Sherwood, and R. Kastner, "Security primitives for reconfigurable hardware-based systems," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 3, no. 2, p. 10, 2010.

[16] T. Huffmire, S. Prasad, T. Sherwood, and R. Kastner, "Policy-driven memory protection for reconfigurable hardware," in *Computer Security–ESORICS 2006*. Springer, 2006, pp. 461–478.

[17] T. Huffmire, T. Sherwood, R. Kastner, and T. Levin, "Enforcing memory policy specifications in reconfigurable hardware," *computers &amp; security*, vol. 27, no. 5, pp. 197–215, 2008.

[18] D. M. Shila and V. Venugopal, "Design, implementation and security analysis of Hardware Trojan Threats in FPGA," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 719–724.

[19] J. Lamberti, D. Manikantan Shila, and V. Venugopal, "xdefense: An extended defense for mitigating next generation intrusions (abstract only)," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 253–253. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554714

[20] R. Chakraborty, I. Saha, A. Palchaudhuri, and G. Naik, "Hardware trojan insertion by direct modification of fpga configuration bitstream," *Design Test, IEEE*, vol. 30, no. 2, pp. 45–54, April 2013.

[21] R. Chakraborty and S. Bhunia, "Security against hardware trojan through a novel application of design obfuscation," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, 2009, pp. 113–116.

[22] A. C. Myers and B. Liskov, *A decentralized model for information flow control*. ACM, 1997, vol. 31, no. 5.

[23] M. M. Farag, L. W. Lerner, and C. D. Patterson, "Interacting with hardware Trojans over a network," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 69–74.

[24] Xilinx Inc. (2014) LogiCORE IP AXI Interconnect v2.1 Product Guide.