Security Assurance for System-on-Chip Designs With Untrusted IPs

Abhishek Basak, *Member, IEEE*, Swarup Bhunia, *Senior Member, IEEE*, Thomas Tkacik, and Sandip Ray, *Senior Member, IEEE*

***Abstract*— Modern system-on-chip (SoC) designs involve integration of a large number of intellectual property (IP) blocks, many of which are acquired from untrusted third-party vendors. An IP containing a security vulnerability—whether inadvertent or malicious—may compromise the trustworthiness of the entire SoC, *e.g.*, by leaking sensitive information or causing execution failures at key points. Existing functional validation approaches, post-manufacturing tests, and IP trust verification techniques are inadequate to accomplish comprehensive system-level security assurance in the presence of untrusted IPs. In this paper, we analyze security issues at the SoC level caused by untrusted IPs. We also propose a novel, resilient SoC security architecture to ensure trusted SoC operation with untrusted IPs. Our architecture realizes fine-grained IP-trust aware security policies in an efficient security policy checker that enables run-time monitoring of security issues arising from untrusted IPs. It also exploits on-chip design-for-debug architecture to ensure trusted information flow from IP blocks to the security policy checker. Unlike existing solutions to the untrusted IP problem, which rely on verification of IP trust before they are integrated into an SoC, the proposed approach follows a fundamentally different architecture-level solution based on run-time resilience. We demonstrate the effectiveness of this framework for system protection using several illustrative practical use cases. We also provide experimental results to show that the overhead of the proposed architecture is modest on representative SoC designs.**

***Index Terms*— Hardware Trojan, untrusted IPs, security policy, design-for-debug, security wrapper, SoC security, resilient architecture, Trusted SoC.**

I. Introduction

With increasing globalization of the design and fabrication processes, the development of a modern computing device involves a large number of participants, — often dispersed geographically — coordinating to create a complex supply-chain pipeline. Most computing devices are developed as System-on-Chip (SoC) designs, which are architected by integrating pre-designed and pre-verified hardware blocks, often referred to as "intellectual properties" or "IPs". Among the participants in the SoC design supply chain are the IP vendors and suppliers, the SoC integration house, the foundry, Original Equipment Manufacturers (OEMs), and product suppliers. Any player in this complex ecosystem can introduce a security vulnerability in the design, *e.g.*, malicious design alterations, subversion of critical access control requirements, etc. The source of a specific security vulnerability may be a rogue employee, a malicious foundry, or even a hacked or malicious design automation tool chain. Even in cases where there is no intended malice, the aggressive time-to-market requirements and high optimization needs may result in errors and vulnerabilities inadvertently left into the design, which can be exploited by a malicious adversary on-field to subvert system security and trustworthiness. Given this situation, it is both critical and challenging to ensure that the product created by this complex ecosystem is trustworthy and free from security vulnerabilities.

In this paper, we focus on system-level security vulnerabilities in the SoC designs created by integration of untrusted IPs. These IPs constitute one of the largest vulnerabilities in the SoC design cycle. In particular, a modern SoC design typically integrates several hundreds of IPs, most of which are procured by the SoC integration house from third-party vendors. It is possible for an IP to contain additional design logic (often referred to as a hardware or firmware Trojan) that can make the system fail during critical system operation or leak sensitive information from the system to an unauthorized agent. Unfortunately, malicious logic in IP design is difficult to identify by standard functional validation [1], [2]. In particular, Trojans are typically designed to be exercised by rare events under very specific execution corner-cases that are difficult to excite in a functional validation environment.

Current research on detection and mitigation of hardware Trojans has primarily focused on trust validation for identification of malicious logic by exploiting their structural and activation characteristics, *e.g.*, location of unused circuits and nets, design points triggered under rare conditions, etc. [3]–[5]. However, such techniques have typically targeted standalone behavior of an IP block and are difficult to adapt or scale to Trojans affecting system-level behaviors, influencing other IPs in the SoC design.

In this paper, we develop a novel *architectural* solution for addressing the problem of untrusted IP blocks. Rather than depending on Trojan characteristics for static verification, our solution enables development of trustworthy SoC designs even if some IP components are untrusted. Our

Manuscript received August 14, 2016; revised December 2, 2016; accepted January 3, 2017. Date of publication January 25, 2017; date of current version April 13, 2017. The work was supported in part by the National Science Foundation under Grant 1603483 and Grant 1603475 and in part by the Semiconductor Research Corporation under Grant 2649.001 and Grant 2651.001. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jean-Luc Danger.

A. Basak is with Intel Corporation, Hillsboro, OR 97124 USA (e-mail: abhishek.basak@intel.com).

S. Bhunia is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: swarup@ece.ufl.edu).

T. Tkacik is with NXP Semiconductors, Chandler, AZ 85226 USA (email: tom.tkacik@nxp.com).

S. Ray is with NXP Semiconductors, Austin, TX 78735 USA (email: sandip.ray@nxp.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2658544

work targets "system-level Trojans", *i.e.*, those affecting the behavior of other IP cores or the interconnect fabric in the system. We provide a detailed overview of these Trojans and discuss the inadequacy of existing functional validation and IP trust assurance approaches for them. Our work thus provides a mechanism for security assurance that is complementary to existing approaches that rely on trust validation. Furthermore, the proposed approach enables protection against Trojans or other malicious design vulnerabilities detected on-field, that may have been missed during functional or security validation. To our knowledge, our work is the first comprehensive architecture-level solution for providing protection against system-level Trojan attacks in SoC designs with untrusted components.

Our architecture is based on a centralized, fine-grained controller for SoC security policies, together with standardized IP-trust aware security wrappers for the IPs to collect and monitor necessary events at run time [6], [7]. We show how to design and configure the controller and associated wrappers to provide run-time threat mitigation from untrusted IPs. We also discuss techniques for mitigating and protecting against errors or malicious logic inserted in the hardware blocks implementing the architecture itself. Finally, experiments are performed to evaluate the overhead introduced by the architecture. Our results show that the overhead is minimal for realistic use-case scenarios. More importantly, the results identify the trade-offs that an SoC integration architect needs to consider in order to adapt our framework in the context of an industrial SoC design.

The remainder of the paper is organized as follows. Section II provides a general overview of SoC security policies, thereby setting up the context of our work. In Section III, we briefly summarize an architecture called E-IIPS for implementing security policies, which we use as the foundation for the work in this paper. Section IV provides a characterization of system-level Trojan behaviors and their impact in SoC designs. Section V describes our proposed security architecture, Section VI shows its application on illustrative use cases, and Section VII provides overhead results. We conclude in Section IX.

II. Security Policies

Our work makes use of fine-grained system-level security policies to protect against Trojans or vulnerabilities in untrusted IPs. In this section, we start with a general overview of security policies. In Section V, we will discuss how to implement fine-grained policies to protect against untrusted IP behavior.

Traditionally, security policies have been designed to define control of sensitive data or *assets* in SoC designs. Such assets include cryptographic and Digital Right Management (DRM) keys, premium content, de-featuring bits, configuration fuses as well as personal end user information, etc., and are sprinkled across different IP blocks. Following are two representative examples for a typical SoC.

- *Example 1:* During boot, data transmitted by the crypto engine cannot be observed by any IP in the SoC fabric other than its intended target.
- *Example 2:* A secure key container can be updated for silicon validation but not after production.

Example 1 is a confidentiality requirement while Example 2 is an integrity constraint. The policies provide definitions of (computable) conditions to be satisfied by the design for accessing a security asset. Furthermore, these may vary depending on the state of execution (e.g., boot time, normal execution, etc.), or position in the development life-cycle (e.g., manufacturing, production, etc.). Below we summarize some typical policy classes. It is beyond the scope of this paper to provide a comprehensive overview of different policies, or even to discuss any of them in detail. The description below merely provides a flavor of some existing policies, and the interested reader is referred to existing extensive literature [8]–[11] on security policies for more detail.

A. Access Control [12]–[14]

This is the most common class of policies, and specifies how different agents in an SoC can access an asset at different points of the execution. Here an "agent" can be a hardware or software component in any IP. Examples 1 and 2 above are examples of such policy. Furthermore, access control forms the basis of many other policies, including information flow, integrity, and secure boot.

B. Information Flow [15], [16]

Values of secure assets can sometimes be inferred without direct access, through indirect observation or "snooping" of intermediate computation or communications of IPs. Information flow policies restrict such indirect inference. Following is an example:

- *Key Obliviousness:* A low-security IP cannot infer cryptographic keys by snooping only the data from crypto engine on a low-security NoC.

Information flow policies require highly sophisticated protection mechanisms and advanced mathematical arguments for correctness, typically involving hardness or complexity results from information security. Consequently they are employed only on critical assets with very high confidentiality requirements.

C. Liveness [17]

These policies ensure that the system performs its functionality without "stagnation" throughout its execution. A typical liveness policy is that a request for a resource by an IP is followed by an event response or grant. Deviation from such a policy can result in system deadlock or livelock, consequently compromising system availability requirements.

D. Time-of-Check vs. Time of Use (TOCTOU) [10], [18]

This refers to the requirement that any agent accessing a resource requiring authorization is indeed the agent that has been authorized. A critical example of TOCTOU is in firmware update, where the policy requires that firmware eventually installed on an update is the same one that has been authenticated.

Typically, most policies relate to *integration* characteristics of SoC designs, not individual IPs i.e., the underlying

threat model includes external attacks through software and/or SoC interface with the system, but not malicious internal hardware/proprietary firmware introduced in the IPs. This threat model is reasonable for SoC designs involving primarily in-house rather than third-party IPs. However, the latter is becoming more widespread, allowing SoC design houses a simpler plug-and-play approach towards designing SoCs and thereby helping them realize stricter time-to-market requirements with existing resources. With this trend, along with static trust verification of IP designs, the root of trust and assumptions based on which security policies are implemented currently in modern SoC designs need to be revisited and analyzed carefully.

III. E-IIPS: A Policy Implementation Architecture

The architecture we consider in this paper is based on a centralized, policy implementation architecture called E-IIPS, which we developed in previous work [6], [19], [20]. Note that the previous work was only concerned with system-level SoC security policy implementation and did not account for untrusted IPs. Below we provide a brief description of E-IIPS architecture, and also point out its limitations in the context of untrusted IPs. The following sections will explain approaches to extend E-IIPS for untrusted IPs.

The E-IIPS architecture includes a programmable central security policy controller (SPC) that keeps track of the system security state and enforces the restrictions imposed by the policies, together with security wrappers for individual IP blocks that detect security-critical events of interest from IP operations and communicate them to the SPC. The security wrappers provide a standardized way for SPC to obtain IP-specific collaterals while abstracting out the details of internal implementation of individual IPs. They extend the existing IP debug/test wrappers for SoC security and can be inserted by IP providers.

As an example of policy implementation through E-IIPS, consider a representative policy that prohibits access of first 16 (address-wise) internal registers of IP A by IP B when A is in the middle of a specific security-critical computation. To enforce the policy, E-IIPS must know when B attempts to access particular local registers of A as well as the security state of the computation being performed by A at that instant. Considering no intrinsic untrustworthy logic in IPs A and B, the operation flow in the E-IIPS architecture implementing the necessary security requirement is shown in Fig. 1. When IP A starts the particular security-critical computation as indicated by a status flag, the corresponding security wrapper detects the event and communicates it with the SPC via a frame. SPC updates the security state of the SoC and disables accesses to all registers of A by B through control logic in B's security wrapper, as part of this particular policy. If at this point B attempts to access a register bit (*e.g.*, register 7) in A's address space to read a configuration value, the security wrapper of B detects this event of interest and informs the SPC. The SPC, determining that the request of B as a violation of the policy, denies corresponding access and maintains the disabling as it is, in B's security wrapper.

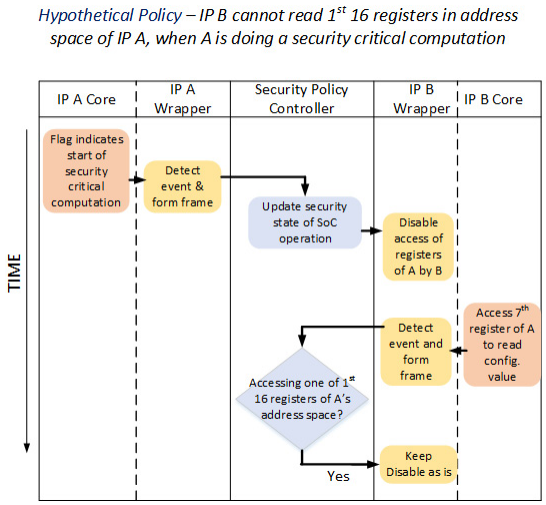

Fig. 1. Flow of implementation of representative SoC security policy in E-IIPS architecture, considering internal designs of constituent IP blocks as trustworthy.

The above example did not assume that any of the IPs A and B to be intrinsically malicious. However, suppose that there is indeed a Trojan in B that is triggered only under certain rare conditions to masquerade a request of the 7^{th} register of A and substitute it with a fabricated request of 28^{th} register in A's address space. Note that 7 shifted two bits left yields 28 in a 5 or higher bit representation, and hence the required malicious payload logic is minimal. The fabricated request line (which is equal to actual request line in most scenarios and hence undetectable) goes through the security wrapper logic whereas the line with the actual request is interfacing with IP A. Observing the requested register address as higher than 16, SPC grants access to B by lifting the restrictions. IP B can easily snoop on potential system secrets stored in 7^{th} register of A.

Of course, the Trojan may not be in the core of B but instead in the wrapper control/detection logic and be activated under rare system conditions in field to trigger a similar attack. Even in a scenario when the same IP vendor provides both the IPs A and B (e.g. two processor cores), a Trojan could be specifically inserted inside A to leak a secret to a system component through IP B's interface under a certain rare trigger condition. Unfortunately, E-IIPS is not resistant to such intrinsic untrusted IP based system attacks. This paper shows how to augment the foundation provided by E-IIPS to address such attacks.

IV. System-Level Security Issues Caused by Untrusted IPs

To develop protection mechanisms for SoC designs with untrusted IPs potentially containing system-level Trojans, we need to first understand the operation and effect of such Trojans. We start with an overview of system-level security issues caused by untrusted IPs to highlight the distinction between our work and related work on IP-level hardware Trojans. Table I lists the key distinctions. For a SoC design, at an event granularity or behavioral level, Trojans or malicious logic in an IP can affect the overall system function rather

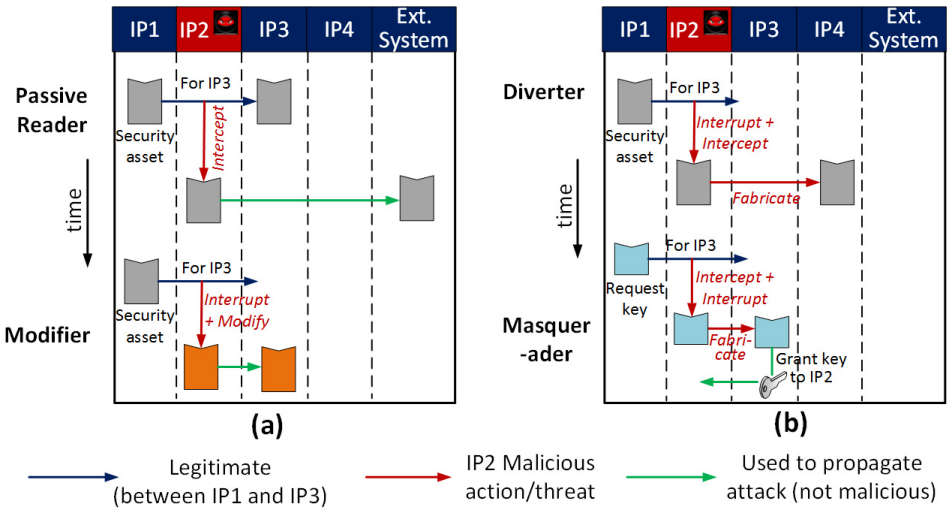

Fig. 2. Message diagram level representation of untrustworthy IP being a (a) passive reader and modifier, (b) diverter and masquerader, with associated security threats.

TABLE I
CURRENT TRENDS IN TROJAN RESEARCH AND SCOPE OF THIS WORK

Existing Research on Untrusted IP	*Scope of the Proposed Work*
Analyze feasibility and effect of Trojans in ASICs and processor cores	Analyze effect of IP level Trojans at the SoC or system level
Explore static IP trust verification techniques for Trojan detection	Run time detection of potentially suspicious IP activity affecting system
Run-time methods that do not address error correction or recovery	IP-Trust aware security policies analyze & assert appropriate security controls

than the IP core itself. For example, a malicious IP may send spurious communications to other IPs resulting in leakage of sensitive information, data corruption, or denial-of-service of the entire system. A critical problem with such system-level Trojans in an IP $\mathcal{A}$ is that their effect can only be observed in an overall system context — typically as a direct malicious effect on another IP $\mathcal{B}$ — and may remain undiscovered in standalone functional or IP-trust validation of $\mathcal{A}$. On the other hand, due to scalability reasons, system-level validation (of both functionality and security assurance) with real use-cases can be exercised only on executions using fabricated chips [21]. However, aggressive time-to-market requirements imply a limited window for post-silicon validation before the product goes on-field, resulting in potential escapes to shipped systems. The objective of this work is to provide architectural support for on-field detection (and mitigation) of system-level Trojan threats.

A. System-Level Trojans vs. IP-Level Trojans

While there is no standard, universally-agreed taxonomy, industrial SoC integration teams have developed a categorization based on analysis of system-usage scenarios and protection requirements at different phases of system execution. In particular, an untrustworthy IP can typically affect system-level behavior through message communications, resource sharing, and control of operation and data flow. We can classify malicious behavior either in terms of the kind of threat introduced, or in terms of the system-level impact of the adversary. We can classify the rogue IP *threats* into four categories [22]: (1) *Interception*; (2) *Interruption*; (3) *Modification*; and (4) *Fabrication*. Correspondingly, in terms of system-level impact, malicious IPs can be characterized with the following taxonomy.

- *Passive Reader*: An IP that illegally reads/collects secret information meant for other IPs.
- *Modifier*: An IP that maliciously changes communication/message content between two IPs.
- *Diverter*: An IP that diverts a message/information between two IPs to a third IP.
- *Masquerader*: An IP that poses or disguises itself as some other component, in order to request service from or control the operation of other IPs.

Unlike IP-level Trojans, the taxonomies above characterize system-level Trojans by *impact* rather than their design, implementation, or triggering characteristics. An upshot is that the taxonomy does not directly translate to a scheme of statically checking a design for system-level Trojans, and one must resort to validation of the run-time system behavior either through dynamic or formal analysis. Also, the categories are not mutually exclusive, e.g., a passive reader may also act as a masquerader. Furthermore, as illustrated in Fig. 2, any adversarial behavior can result in multiple threats, e.g., a masquerader can cause any of interception, interruption, or modification.

Below we provide some examples of system-level Trojans arising from a sample of potentially malicious IPs. While the descriptions themselves are simplified for pedagogical reasons, they are inspired by realistic protection/mitigation strategies developed by security architects in typical industrial SoC design flows. Note however that they are only meant to be illustrative samples; an exhaustive overview of the spectrum of vulnerabilities caused by untrusted IPs is beyond the scope of this paper.

B. Untrusted Processors

A typical example of a malicious logic in a processor involves exploiting reserved opcodes. Such a malicious

TABLE II
ASSUMPTIONS REGARDING TRUSTWORTHINESS OF COMPONENTS IN PROPOSED METHODOLOGY FOR AN UNTRUSTED IP

	IP Core	IP Security Wrapper(ISW)	Local DfD	SPC-DfD prog. link	ISW to SPC comm.	SPC	Interacting IP
Trust Assumptions	Except standard test,debug wrapper, all untrusted (eg. datapath,cntrl.)	Except standard interface to SPC, DfD& frame gen., all untrusted	Tru-sted	Tru-sted	Tru-sted	Tru-sted	Tru-sted

behavior avoids detection by IP-level validation techniques unless there are (potentially expensive) checks to ensure functional completeness of the instruction fetch logic to return a fixed, golden set of instructions. Another example is the generation of shadow load or stores by malicious decode or memory access logic. Note that the malicious logic can encompass hardware, firmware, and microcode. Such Trojans can cause system-wide effect by using unauthorized DMA requests at a high privilege level, potentially enabling writes of system secrets by the memory subsystem. With memory-mapped input-output, such writes can be performed on external devices, thereby leaking system secrets.

C. Untrusted Memory Controller

A memory controller IP governs access to the system memory, which is shared by different IPs and system components of the SoC design. With memory-mapped input-output and DMA, the memory controller governs access to external devices as well. An illustrative system-level Trojan in the memory controller can tamper with stored values of a specific IP, e.g., an IP governing an external temperature sensor. Tampering may affect the temperature feedback control, causing the system to overheat and fail. Such a scenario is very difficult to validate in a standalone IP-trust paradigm, and the exact conditions for exciting the malicious behavior may be difficult to exercise during a post-silicon debug through typical validation use-cases or spec-guided directed tests.

D. Untrusted Network-on-Chip

Another shared system resource which is heavily used by the different IPs to communicate with each other to perform system-wide functions is the Network-on-Chip (NoC). NoCs are composed of a hierarchy of routers or switches which direct the packets of information from the source to the intended destination. A malicious router can potentially misdirect a highly secure packet (e.g., a security key from the cryptographic module) to a device controller, compromising the system's digital signature, fuse or debug configuration, or private end-user information.

E. Untrusted Device Controller

Device controllers control devices like USB, bluetooth, ethernet, modem and other I/O components. A malicious device controller may modify bits of device data input under certain trigger events such as a particular authentication request (for login) in a program on the processor, that would require an user to input credentials. This intentional tampering may lead to denial-of-service attacks.

V. SoC SECURITY ARCHITECTURE RESILIENT TO UNTRUSTED IP

Our approach to ensure security assurance in the presence of untrusted IPs is to develop fine-grained, IP-trust aware security policies. We show how to implement these fine-grained policies on top of a standard security architecture for access control over security assets. In particular, we extend E-IIPS, a microcontrolled, centralized architecture for security policy implementation developed in previous work [6]. E-IIPS permits implementation of diverse system-level security policies. The previous work primarily used it for protection of system assets against attacks through high-level software stacks and SoC to system interface. The key contribution of this paper is to show how to adapt and extend such a policy implementation architecture to provide a systematic, disciplined and scalable approach for addressing the threats of the aforementioned SoC level Trojans.

A. Trust Assumptions

Developing any security assurance requires an assumption of which design components (and supply-chain entities) can be trusted. Our framework assumes a trustworthy SoC design integration house, collecting IPs from potentially untrusted vendors. The key assumptions regarding integrity of components interacting with a particular untrustworthy IP are listed in Table II. In the IP, we assume that apart from the standard (highly validated) test and debug wrappers, a Trojan might be inserted anywhere within the internal control logic, data-path and/or temporary storage. Furthermore, the E-IIPS architecture [6] involves smart security wrappers on IPs to communicate with the SPC. These wrappers are typically inserted by the corresponding IP providers, like most test/debug wrappers. We assume that apart from the relatively standard (in terms of high-level functional specification and microarchitecture) frame generation logic and wrapper interfaces with SPC and local design-for-debug (DfD), a Trojan might be present anywhere in the wrapper sub-units. However, we assume no collusion among different (even potentially malicious) IPs or components in the SoC. In other words, interacting components coming from different IP design houses may be untrustworthy, but they do not express their malicious behaviors (if applicable) under the same system contexts. For example, the debug macrocell local to the corresponding untrusted IP is part of the SoC DfD framework and comes from the provider of the debug infrastructure IP. Hence, even if we consider the macrocell to be untrusted, as the IP design houses operate independently in the normal SoC ecosystem, the contexts under which they are malicious are completely

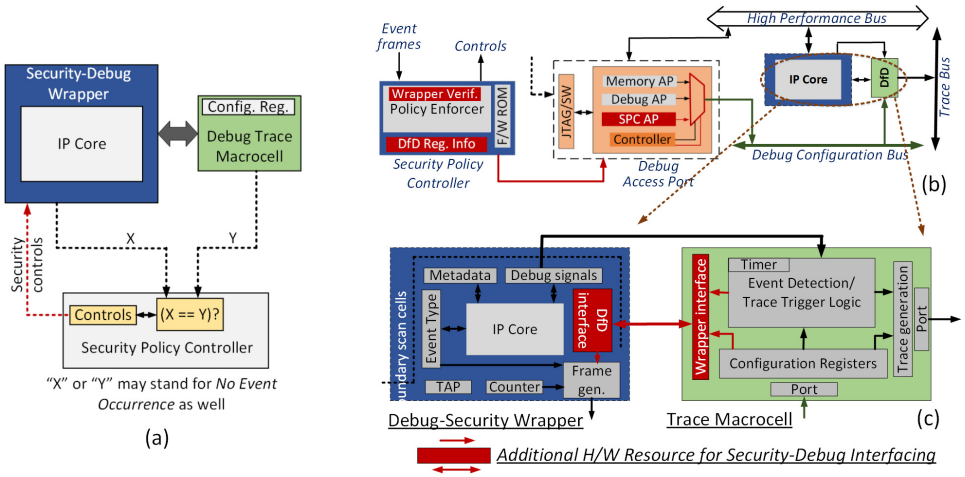

Fig. 3. (a) Cross-verification based proposed methodology to detect untrusted wrappers; (b) Modifications required for re-purposing DfD for security policies in SoC; (c) Zoomed view of the additions in IP security wrapper and corresponding DfD.

different. Hence, in the context of this untrustworthy IP, the local DfD unit and/or the interacting IP block is trusted as listed in Table II.

B. Untrustworthy Security Wrappers

We first consider the case where the malicious logic is enclosed within the security wrapper. The case of a Trojan in the IP core functionality will be discussed in Section V-C. The solutions for the two can be effectively combined to provide the desired security. To understand the issue of malicious wrapper, note that the event monitoring/extraction logic in a smart security wrapper is highly susceptible to Trojan insertion, since the events detected vary according to IP types (thus making it difficult to catch malicious logic through validation). A malicious event detection logic may cause the security wrapper to not report certain security-critical events. SPC would then be oblivious to any behavior/activity (rogue or trusted) originating inside the IP and potentially propagating to the system level. A Trojan may also communicate to SPC that an event has occurred inside the IP when it has not, leading to either system failure or denial-of-service.

1) Solution Methodology: Our approach involves detection of an untrustworthy security wrapper action by verifying it with an on-die trusted monitor at run time as shown in Fig. 3(a). The trusted monitor is implemented using the DfD trace macrocell local to the IP. Note that DfD instrumentation exists for virtually all IPs in a SoC design and incorporates logic for detecting assertions, coverage events, and checkers for post-silicon validation. For example, a processor trace macrocell in a SoC design includes standard instruction and data value/address/range comparators, condition code/status flag match check, performance counters, event sequencers, logical event combinations etc. for validation/debug. Utilizing this to re-purpose debug instruments for security, SPC would configure the DfD module for the untrustworthy IP of interest at boot/power-up to detect some or all of the security-critical events at run time [20]. For an event communicated by the security wrapper (or DfD), SPC compares the other's response (same event, wrong event, or no event) and asserts the necessary security controls. The security architect can configure the appropriate mitigation strategy as a security policy in SPC. As an example, SPC may obtain all policy-critical events (of IP) from the trusted DfD module in the event of one or multiple mismatches detected from the corresponding wrapper. However, re-purposing DfD for security requires addressing the following trade-offs.

- Post-silicon debug and validation are themselves critical activities performed under highly aggressive schedules. Re-purposing the DfD should not interfere or "compete" with debug usages of the same hardware.
- On-chip instrumentation, and in particular the debug communication fabric, is optimized for energy and performance in debug usages. While re-purposing DfD, one must ensure that the system power/energy profile is not significantly disrupted.

2) Implementation Note: The SPC requires an interface with the debug framework to configure the local DfD. Fig. 3(b) shows how to extend a typical debug access port [23] to achieve this. The configuration register values of the associated DfD macrocells are stored as policy arguments in SPC (Fig. 3(b)). Standard trace macrocells typically incorporate logic resources to detect all of the required IP events. Hence, most of them can be configured at system boot to extract required events at run time. To ensure noninterference with debug usage, we transmit security data from DfD to SPC via a separate port instead of re-purposing the debug trace port and the trace communication fabric. This port is interfaced with the corresponding IP security wrapper to send the event information. The corresponding architecture is shown in Fig. 3(c).

C. Untrustworthy IP Cores

To address the issue of untrusted IPs with potentially embedded system-level Trojans, our approach is to facilitate systematic development of a run-time monitoring and protection scheme for direct or indirect effects of system-level Trojans using fine-grained security policies implemented through SPC. We will now extend the centralized policy framework to facilitate this usage.

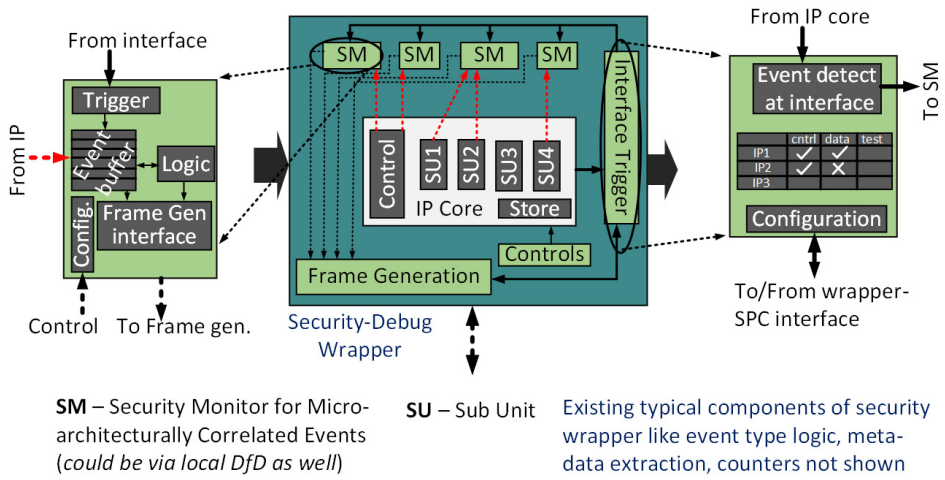

Fig. 4. Architecture of enhanced IP-Trust aware security wrapper incorporating additional security monitors and interface trigger logic modules.

Event Detection: Our overall approach is to identify events that can detect anomalies in the communication between an untrusted IP and the rest of the SoC design. Note that there is typically no golden RTL model for third-party IPs; nevertheless, the SoC design integration team has access to the high-level behavior and architectural features of the IP. For example, in case of a processor, key parameters including number of pipeline stages and cache levels, virtual memory parameters, etc. are available. A key observation is that these high-level design features, coupled with common, architecture-level rationale can be used to verify correlations between specific temporal events across different IP sub-components, and thereby detect potentially un-trustworthy behavior that might originate inside the IP and affect the SoC operations. In particular, a functionally relevant operation, meaningful and visible to SoC components external to the IP, incorporates specific correlated, internal events occurring across multiple sub-units in the IP. These corresponding events are referred to here as "Micro-architecturally Correlated Events (MCE)". System-level Trojans disrupt/affect the correlation between these spatio-temporal events. Some examples are provided below.

Example 1: For a typical in-order processor core with 5 pipeline stages, a memory sub-system request (or simply a load/store in a RISC core) would involve a typical MCE sequence, i.e., the decode stage deciphering the instruction to be a load/store (LD/ST), the execution unit calculating the address and consequently the memory access stage generating the appropriate memory sub-system request. An example processor level Trojan is one inside the memory access logic to conditionally generate a shadow LD/ST in addition to the normal one. Here the correlation disruption is in the form of a single active memory instruction (after decode) in instruction window generating two data memory requests, which does not satisfy common architecture-level assertions for a simple RISC processor.

Example 2: A hardware Trojan in the instruction fetch stage of a processor, triggering a branch or jump (to potentially a malicious source) without any previous active branch/jump instruction or corresponding activity in the program counter (PC) select logic is an example of not satisfying correlation.

Example 3: In a typical memory controller architecture, a Trojan inside the request buffer causing the earliest active request not being served under a FIFO based scheduling policy or a random change to row buffer based policy for a Trojan in the arbiter/scheduler (for just a normal request; no condition flags etc.) are examples of uncorrelated spatio-temporal events.

Example 4: A rogue router conditionally generating an additional destination address and sending the packet in addition to the address in its active buffer also does not satisfy the MCE typical for a router from the point of view of high-level specification and common architecture level rationale.

The micro-architectural components required to be added, to target this untrustworthy IP core problem are described below.

1) IP-Trust Aware Security Monitors: For targeting confidentiality (C) and integrity (I) attacks at the system level, the payload of the Trojan in the IP would be designed by an adversary to propagate the malicious action out of the IP. This would invariably involve output transactions with interacting IPs or SoC components. Detecting availability attacks is more challenging and is dealt with later. For C/I attacks through these untrusted IPs, we monitor high-level temporal events across the IP sub-units that directly/indirectly affect or lead up to these output interactions. This is done by inserting "security monitors" inside the test/security wrapper to monitor and store a recent history of high-level events from strategic locations internal to the IP. An illustration of the architecture of a typical wrapper and associated security monitors is provided in Fig. 4. The selection of these strategic locations internal to the IP from which the necessary "MCE" are extracted, depends on the trade-offs between the desired range of Trojan coverage and constraints of hardware cost as well as increased design effort. For example, a Trojan in the control state machine of a RISC processor may lead to easy, stealthy payload expression at lower hardware costs from point of view of attacker, compared to malicious insertion in the data path sub-units. Hence SoC

TABLE III
CATEGORIZATION OF IP-TRUST AWARE EVENTS (MCE) AND POLICIES BY IP TYPES

IP	*Ex. IPs*	*Ex. MCE list*	*Ex. Associated Policies*
Processor Core	GPP, GPU, μC based device and system controllers (USB, video, power management etc.)	Instr. fetch, decoded value, memory req. generation, cache access, interrupts/exceptions, prog. counter calculation logic	Decode preceded by corr. fetch; Fetch preceded by corr. prog. counter(PC) calc.; Mem. system access preceded by corr. decode; Main mem. access only on cache miss; PC jump only on corr. instr. or trap
Storage Controller	Main memory controller cache controller flash, NVM control logic, DMA engines	New/next access in req. buffer, change in scheduler policy or priority of I/O channel (DMA), bank/col/row calc. by interface	Entry in req. buffer corr. to valid mem. req.; Req. scheduled corr. to prev. valid buffer entry; Req. scheduled acc. to policy by scheduler; DMA req. to mem. preceded by corres. μP grant
Communication Fabric Unit	Router, switch, bus controller, bus bridge	New/next req. in buffer, next router along which packet sent, scheduling policy change, freq. scale in bus bridge	Entry in buffer corres. to valid comm. req.; Scheduled packet corres. to prev. valid buffer entry; Next destination chosen from current routing policy; Next bus transaction selected acc. to current priority

designer may decide based on area, power constraints etc. that security monitors would be only inserted inside the controller logic components, thereby not including the data paths within the Trojan coverage range.

We developed a set of high-level IP-Trust aware security critical events or "MCE" and associated policies for broad categories of IPs. These are listed in Table III. Depending on the degree of untrustworthiness of the IP, extent of IP and system level validation, and the required range of Trojan coverage, events may be added or removed from this MCE list or may be selectively monitored via appropriate boot/run time configuration by SPC. Fig. 5 illustrates some typical potential sites for MCE extraction across IP types. For Trojans in the fetch stage of a simple in-order pipelined processor, a typical high-level correlation check is to verify the instruction/s fetched against the corresponding program counter calculated in last cycle or presence of recent asynchronous interrupts/exceptions through status registers. For untrusted program counter logic, this may be subject to check one cycle back to the past instruction fetched or decoded. Note that an attacker is not likely to insert these Trojans in multiple sub-units internal to the IP with a view of stealthiness and/or as part of the assumption of no collusion between trojans in sub-units. Hence, with most of the sub-components of the IP being actually legitimate, a malicious trigger can be detected and flagged near its origin.

2) IP-Trust Aware Interface Triggers: In our proposed solution, the event correlation verification is necessary only when the corresponding IP attempts to communicate with the other SoC components through its output interface. These are detected by additional logic "Interface Triggers" (IT), inserted as part of the security wrapper as shown in Fig. 4(right). On detection, the security monitors are triggered to send their recent security-critical event logs to the SPC. The constraints of communication bandwidth, SPC execution resources, power/energy profiles etc. demand for a selective, configurable trigger to verify these events. Any input IP interface in a SoC can be categorized into four types of signals: (1) *Control*, e.g., command, status, valid etc.; (2) *Data*; (3) *Test*, e.g., scan chain inputs; and 4) *Global*, e.g., clock, reset etc. [24]. The SoC designer analyzes the security criticality of the various inter-IP control and data inputs from system/subsystem level simulation. For example, all read/write data accesses for a processor may be critical at boot, while during normal execution only secure address ranges may be relevant. These conditions are configured by SPC in the interface trigger logic at power up to select trigger events.

Apart from these output interface triggers, the input control/data stream must also be accounted for, e.g., for a security critical IP it is important to take into account the source of these inputs in determining the triggers for untrusted activity verification. It is done with a tag (e.g., a bit), which is associated and propagated with the communicating IP inputs, if applicable. The tag is used in the policies to select events to trigger on and validate, and the specific security controls to apply at the interface logic A typical list of output interface trigger conditions for a processor core is provided in Table IV.

3) IP-Trust Aware Security Policies: IP-Trust aware security policies in the SPC dictate what correlation checks should be performed between MCEs and the applicable security controls at the interfaces of the untrustworthy IP/s before and after the verification of its behavior. They are programmed as firmware in the instruction memory of the SPC policy enforcer. Examples of typical correlation checks are given in Table III. These policies govern the security controls at the interface logic of the corresponding wrappers to prevent system compromise. Two obvious mitigation scenarios are the following, though more fine-grained checks can be programmed depending on SoC security requirements.

- Disable all interface actions until verification of the recent MCE for intended correlation by the SPC. This prevents system-level attacks at the potential cost of some performance overheads, which depend on frequency of such policy occurences, current SPC loads, communication bottlenecks etc.
- Allow the interface activities to continue, thereby allowing the IP actions to propagate to other SoC components. At the same time, the MCE are verified for correlation. In case of undependability detected, the system would be halted and possibly rolled back by the SPC (if applicable), via check points. In these cases, the relative timings of the assertion of proactive security controls (if necessary),

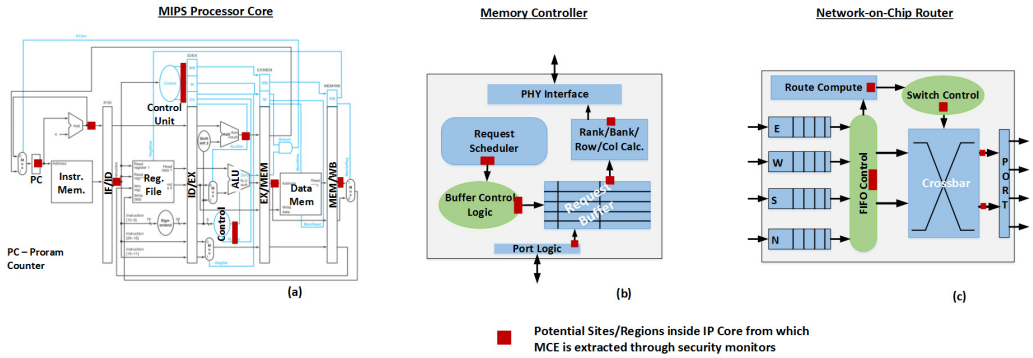

Fig. 5. Potential sites in the IP design for insertion of IP-Trust aware security monitors in a) MIPS processor core, b) representative memory controller and c) NoC router.

TABLE IV
REPRESENTATIVE INTERFACE TRIGGERS FOR AN UNTRUSTWORTHY PROCESSOR CORE

Interacting IP	*Security Critical? (Tag)*	*Example o/p Interface Trigger*
Memory Controller (I/O)	Partially (RD/WR from/to Secure Memory)	If secure tag or at boot, all control/data, else only RD or WR req. to secure memory
DMA Engine (I/O)	Partially (only I/O from/to specific devices)	If secure tag or at boot, all control/data, else only I/O req. to specific devices
Flash Controller (I)	Fully (Flash contains bootcode)	NA (Authenticated upgrade only in special debug mode)
Crypto-IP(I/O)	Fully	All control/data at all times
DSP Accel.(I/O)	No	Control/data only at boot

determine the extent of impact at the system level. The performance overheads are however negligible. Enhanced priority is applied to such policies in SPC to reduce chances of system compromise.

VI. USE CASE ANALYSIS

Consider the application of the framework in an illustrative system execution scenario. In this use case, we assume that the adversary inserts Trojans in both the wrapper and the core of the main memory controller (MC) and designs same rare event based trigger logic for both. The attack model is as follows:

A. Information Leakage via Malicious Shadow Store to Memory of Specific Device

For particular load/store request to protected region of memory during specific secure critical program execution (trigger), in the scenario of cache miss and associated eviction of dirty cache block to memory, the Trojan in the memory interface of the memory controller (MC) is triggered to cause a payload of an extra shadow store operation to the memory mapped region of device of interest to the attacker. The attacker, through a controlled program on processor can request an output to the corresponding device. Hence potential critical assets used by the original secure program may be potentially extracted. The trigger also leads to activated malicious logic in the MC security monitors to not communicate event logs to SPC on interface triggers.

B. Assumptions

We assume that the attacker (*e.g.*, memory controller (MC) IP provider), can utilize the configured secure memory range in the MC security wrapper to act as a trigger for the inserted Trojans. Although other IP cores can be independently malicious, in the context of this attack model, the memory controller is the only IP considered untrustworthy. This follows from the requirement the malicious IPs are collusion-free. All accesses associated with the secure memory region are disabled by the SPC until the corresponding microarchitecture events are verified for intended correlation. The MC DfD trace macrocell is utilized for untrusted security wrapper validation by the SPC. The MC is assumed to constitute a request buffer, access scheduler and controller-memory interface logic. Below are the sequence of steps as illustrated in Fig. 6.

C. Flow of Operation

1) At the boot phase, the SPC configures the security monitors (if applicable) and the interface trigger logic of the MC security wrapper to program for example "all accesses associated with secure portion of the memory should be disabled until the preceding MC actions are verified for trustworthiness". The DfD trace macrocell for MC are also configured for security wrapper verification.
2) During normal execution, for a program executing on the processor core, a cache miss occurs for a load-store from/to secure memory. Associated with it is also a cache block eviction. The processor wrapper checks whether the memory access is allowed according to the privilege level of the program. It passes the check and the accesses are added to the MC request buffer.
3) At some point, along with the missed load-store, the evicted store (ST) is scheduled by the MC scheduler arbiter. The associated address value triggers the Trojan to generate an additional shadow store to device memory of attacker's interest (payload). Consequently, both of these requests would access the physical memory next. However, the configured interface trigger controls of the MC wrapper (by SPC policy) has disabled the controller interface until the verification of past MC events. The interface logic triggers the security monitors to send

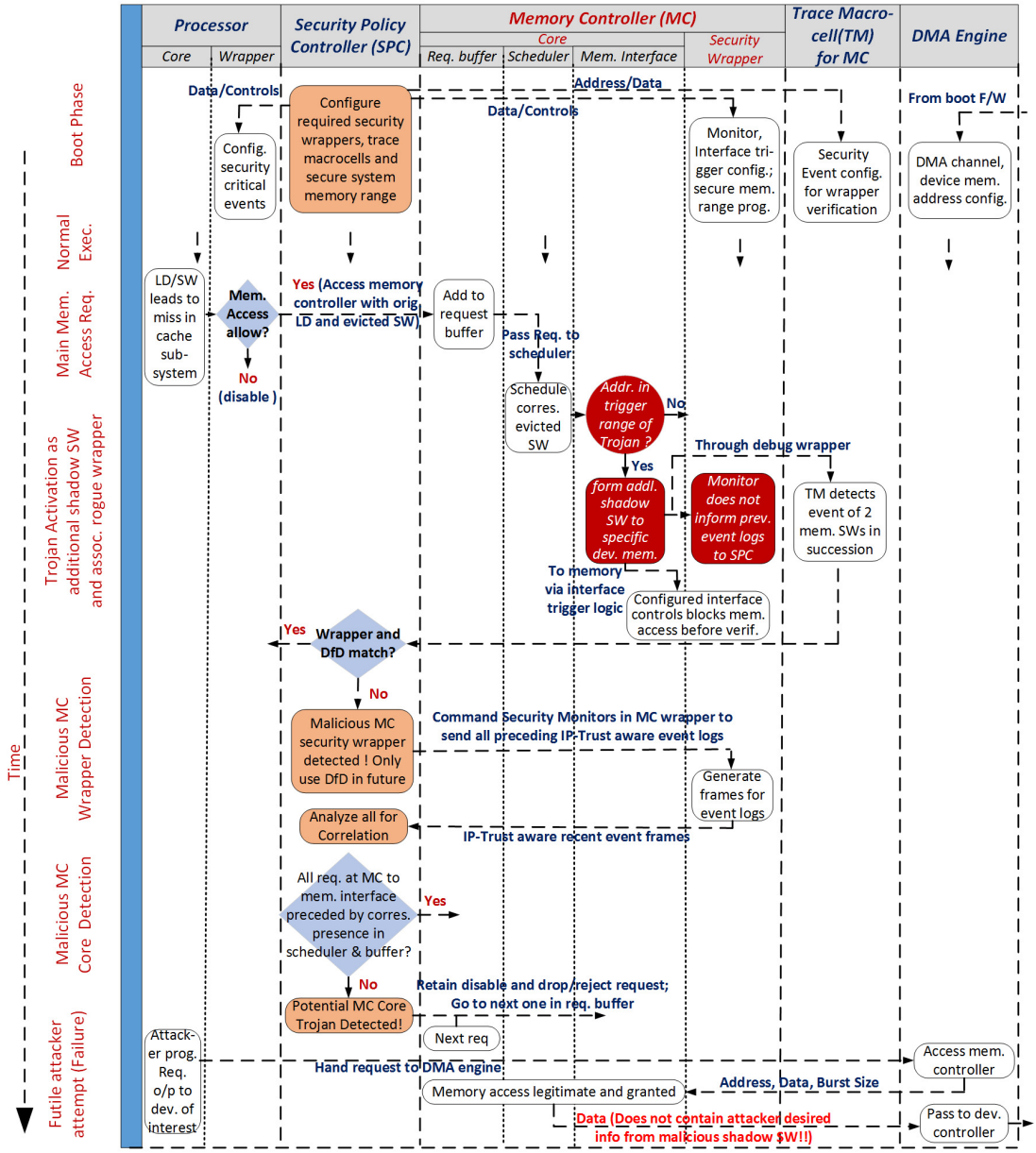

Fig. 6. Operation flow of the proposed solution for providing system level protection in use case scenario of Trojan in security wrapper and inside core of main memory controller.

the event logs for check to SPC. But a Trojan (same delayed trigger) in the monitor prevents the SPC of being notified of these events. At the same time, the DfD trace macrocell detects the corresponding events of 2 scheduled stores to physical memory and informs the SPC. Receiving nothing from the security monitors, the SPC detects the malicious MC wrapper. The SPC commands the MC security monitors to send the event logs for verification.

4) Assuming the event logs to be legitimate (which can be verified by DfD, but not shown here), the SPC analyzes them for satisfying a correlation according to MCE of the memory controller IP core. Consequently, according to rule "All read-write accesses at memory interface must be preceded by their presence in the scheduler and the request buffer sub-units", the correlation failure is caught and the Trojan action detected. The access is dropped.
5) The attacker, running a program remotely on the processor core, requests for data output from the corresponding memory (shadow address) to the device of interest. However, with the attack thwarted by the proposed solution, the adversary fails to extract any security critical data.

VII. Overhead Analysis

In this section, we consider the hardware overheads incurred by the IP-Trust aware security monitors, inserted in the wrappers of different IP types, for varying Trojan coverage scenarios. Due to lack of standard open-source models of studying SoC architecture, we have developed our own SoC model in Verilog RTL. Fig. 7 shows the key features of this model. Although simpler than an industrial SoC design, our

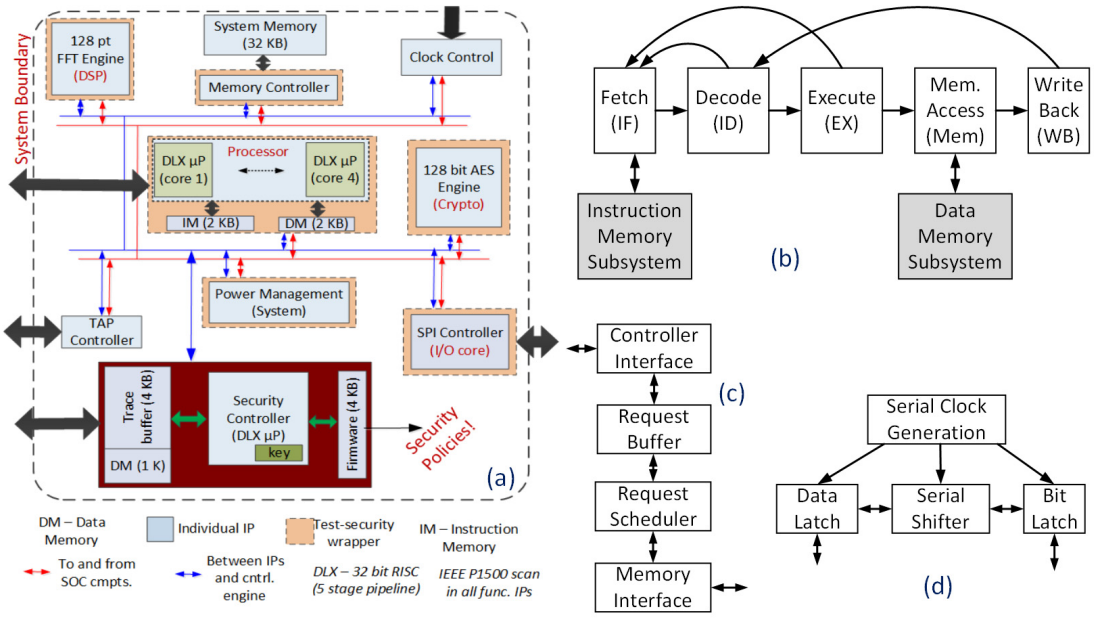

Fig. 7. a) Block Diagram of our SoC framework; the internal sub-units of the b) DLX processor, c) Representative memory controller and d) SPI controller.

model is substantial and can be used for implementing realistic security requirements. In particular, it includes a DLX microprocessor core (DLX), a representative memory controller IP, a 128b FFT engine, a 128b AES crypto core as well as a Serial Peripheral Interface (SPI) controller. We developed an instance of the E-IIPS framework [6] for this model; this includes implementing security wrappers for IPs to detect the standard set of critical events for implementing security policies, to ensure protection against attacks via system interfaces and S/W stacks. In Section VI we show some of the implemented events. We also implemented a representative design-for-debug infrastructure, based on a simplified version of ARM Coresight®. The framework has been functionally validated using ModelSim for typical use cases. All analysis for area/power was performed using 32nm technology library.

A. Security Monitor Implementations

To incorporate the proposed security monitors, we considered three IP blocks of different types, e.g., the DLX processor core, the memory controller and the SPI controller. According to the methodology described in Section V for untrusted IP cores, security monitors are inserted in their wrappers at varying spatio-temporal granularities, for verifying different Trojan coverage scenarios. Table V lists the Trojan logic (in terms of their effect/payload), against which protection is ensured by these monitors and the necessary IP-Trust aware security policies in the SPC. Interface trigger logic is also incorporated within the wrappers. For event communication to SPC, a separate, frame-based interface is incorporated in the IP wrappers to avoid contention between the two traffic types. Note that it is infeasible for the SPC to perform complete emulation of the high-level, intended IP operation. Hence monitors must be placed at intermediate strategic locations in the IP as discussed in Section V, to enhance efficiency of Trojan coverage. In scenario I of DLX Trojan coverage, monitors are placed at input of fetch, output of decode, and output of memory access stage, whereas in scenario II, monitors are additionally inserted at ALU outputs and write-back outputs. In the latter scenario, a malicious logic triggering a branch/jump for a particular unsatisfied condition (which implies there should be no branch/jump under normal operation) is detected using the ALU monitor. Similarly for the memory controller (MC) in scenario I, monitors at output of request buffer as well as scheduler can detect any malicious activity in scheduler hardware or firmware. Additionally, storing the memory requests arriving at the input interface of MC allows one to detect Trojans in the buffer (scenario II).

B. Results

The estimated area and power overheads for the inserted security monitors and interface triggers in the three scenarios of Trojan coverage for the processor core are shown in Table VI. We note that the overhead is calculated here with respect to the base design of DLX core with a standard security wrapper. The overhead varies between a relatively small (6% to 11%) across the coverage scenarios. Besides, increasing the additional frame interface width to 256 bits from 32 bits to transfer simultaneously the 8 temporal event logs stored in each monitor (8 chosen according to design details) incurs minimal additional overhead. In addition, a point to also highlight is that for significantly increased Trojan coverage from scenario II to III, the corresponding increase in hardware overhead is minimal. This signifies that one could potentially gain high run time security against an untrusted IP at minimal hardware cost.

Similarly, the incurred hardware overheads for the memory controller and the SPI controller are listed in Table VII. For the memory controller, a 4KB register-based functional memory is added to the base memory controller to calculate the security monitor overheads. Although for small IP designs

TABLE V

DIFFERENT SCENARIOS OF TROJAN (REPRESENTED BY PAYLOAD) COVERAGE BY INSERTION OF SECURITY MONITORS IN THREE IP CORES OF OUR FRAMEWORK

Sec. Monitor Coverage Scenarios	***Ex. Processor Trojan (Payload) Coverage***	***Ex. Mem. Controller Trojan (Payload) Coverage***	***Ex. SPI Controller Trojan (Payload) Coverage***
Scenario I	Addl. shadow (or hide) LD/ST by mem/ALU stage, wrong PC select logic, erroneous decode, wrong WB op. calc./propagate	Drop/Fabricate mem. req. by scheduler, wrong next req. select from policy (e.g. FIFO)	error in serial clk gen. logic, wrong counter op. for external bit Tx/Rx
Scenario II	*All Prev.* + PC jump/branch calc. w/o corres. inst., mem./WB stage perturbs ALU res./WB val., ALU perform wrong calc.	*All Prev.* + drop/fabricate req. by buffer, error in bank/row/col calc. at interface	*All Prev.* + data tamper at Tx/Rx shift, data latch at incorrect times
Scenario III	*All Prev.* + cache sub-system altering mem. req. data/addr. wrong prog. branch T/NT	NA	NA

TABLE VI

AREA & POWER OVERHEAD OF SECURITY MONITORS IN PROCESSOR IP (ORIG. AREA AND POWER WITH 1 KB INST., DATA MEMORY AT 32 nm - 352405 μm^2, 12.56 mW)

Different Security Monitor Scenarios	***Die Area Overhead*** (%)	***Power (Active + Leakage) Overhead*** (%)
Case I (32 b o/p)	6.68	6.92
Case I (256 b o/p)	7.17	7.32
Case II	10.44	10.82
Case III	11.68	11.62

TABLE VII

AREA & POWER OVERHEAD OF SECURITY MONITORS IN MEMORY CONTROLLER (MC) IP AND SPI CONTROLLER IP (ORIG. AREA AND POWER OF MC AND SPI WITH WRAPPERS AT 32 nm - 629433 μm^2, 13.81 mW;; 5456 μm^2, 0.298 mW)

Different Monitor Scenarios	Memory Controller		SPI Controller	
	Area Ovrhd.(%)	***Pwr. (Active+Leak) Ovrhd.***(%)	***Area Ovrhd.***(%)	***Pwr. (Active+Leak) Ovrhd.***(%)
Case I	10.77	14.04	29.08	19.12
Case II	11.16	18.53	101.88	66.77

TABLE VIII

DIE AREA OVERHEAD (OVH) OF SECURITY MONITORS (SMS) WITH MAXIMUM TROJAN COVERAGE WRT. TO OUR SOC FRAMEWORK (AREA - 13.1$X10^6$), APPLE A5 APL2498 (AREA - 69.6$X10^6$), INTEL ATOM Z2520 (AREA - 40.2$X10^6$), ALL AT 32 nm PROCESS TECHNOLOGY

IP Core	***OVH***(%)***wrt model***	***OVH***(%)***wrt A5***	***OVH***(%)***wrt Atom***
Processor	0.31	0.059	0.1
Memory Controller	0.543	0.103	0.175
SPI Controller	0.043	0.008	0.014

the area/power overhead could be significant with respect to the base (e.g., in scenario II of SPI controller), the overhead with respect to the full SoC die is insignificant for all IPs, as shown in Table VIII. Along with our representative SoC model, two commercial SoCs, manufactured at 32 nm, are also taken into consideration to calculate these approximate overheads. Note that the increase in the SPC overhead with respect to its base value, due to incorporation of additional IP interfaces for event logs, control signals and IP-trust aware policies as firmware in instruction memory, has not been taken into account in this work. However, from the sample values in Table VIII, we believe that even after incorporation of SPC overheads, the H/W overhead of this proposed architecture would be minimal with respect to the full SoC die. Perhaps more generally, our experiments show the design parameters and trade-offs that a security architect must analyze to deploy this framework in an industrial SoC design environment.

VIII. RELATED WORK

There has been significant previous work on understanding, detecting, and mitigating threats arising from Hardware Trojans [4], [5], [25]–[29]. This includes characterization of

Trojans based on design and activation characteristics, e.g., time-bombs, combinational event triggers, etc. Most testing methods have been aimed at static trust validation of IP cores [3], [4], [25], [30]. These approaches include the detection of suspicious nets, nodes, regions or unused circuits, and generation of optimized targeted test sets. Although effective on specific instances, these approaches have significant false negatives/positives based on chosen test set, threshold parameters, design type etc. The efficiency of these techniques with respect to test time and Trojan coverage reduce with increasing sizes of modern designs. Formal verification techniques have also been used for trust validation on small designs [30], [31], but are difficult to scale to industrial designs consisting of hundreds of IPs. There are also recent reports on use of run-time monitors for detecting hardware Trojans [24], [32], but they have been specifically designed for microprocessor cores and are not applicable for arbitrary IPs interacting with other components in a SoC design. Furthermore, these approaches do not address online mitigation of detected threats in a platform. Some approaches have been studied to harden or strengthen IP designs against hardware Trojan insertions [33]–[35], but they are complementary to this work.

IX. Conclusion

We have presented, for the first time to our knowledge, a comprehensive analysis of trust issues at the SoC level caused by untrusted IP blocks. We have also presented a novel architecture-level solution to achieve trusted SoC operation with untrusted IPs. With growing reliance on third-party IP blocks during the SoC design process, untrusted IPs are rapidly becoming major security concerns for SoC manufacturers. Design-time IP trust verification approaches proposed in the literature to date, fail to provide high confidence in identifying malicious implants of all forms and types, as well as possible exploits of apparently benign design artifacts, such as the design-for-test infrastructure. The proposed architecture provides a low-cost and robust run-time defense against untrusted IPs. The architecture employs fine-grained IP Trust aware security policies to detect and prevent malicious operation of an untrusted IP at the system level. It builds system trust by considering trustworthiness of a minimal set of standard components (e.g. design-for-debug structure and a security policy checker), which are suitable for comprehensive trust verification. The proposed architecture is evaluated over diverse use-cases obtained from industry, which proves its effectiveness for representative SoC designs. It is applicable, in general, to various types of SoC designs and is scalable to large complex SoCs with an arbitrary number of IPs. Future work will involve further evaluation of the proposed architecture with diverse SoC designs and automatic synthesis of the security policies.

References

[1] J. Rajendran, A. K. Kanuparthi, M. Zahran, S. K. Addepalli, G. Ormazabal, and R. Karri, "Securing processors against insider attacks: A circuit-microarchitecture co-design approach," *IEEE Design Test*, vol. 30, no. 2, pp. 35–44, Apr. 2013.

[2] A. Das, G. Memik, J. Zambreno, and A. Choudhary, "Detecting/preventing information leakage on the memory bus due to malicious hardware," in *Proc. IEEE DATE*, Mar. 2010, pp. 861–866.

[3] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in *Proc. ACM CCS*, Nov. 2013, pp. 697–708.

[4] M. Banga and M. S. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," in *Proc. IEEE HOST*, Jun. 2010, pp. 56–59.

[5] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2010, pp. 72–159.

[6] A. Basak, S. Bhunia, and S. Ray, "A flexible architecture for systematic implementation of SoC security policies," in *Proc. IEEE ICCAD*, Nov. 2015, pp. 536–543.

[7] X. Wang, Y. Zheng, A. Basak, and S. Bhunia, "IIPS: Infrastructure IP for secure SoC design," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2226–2238, Aug. 2015.

[8] J. Rushby, "Noninterference, transitivity, and channel-control security policies," SRI, Menlo Park, CA, USA, Tech. Rep. CSL-92-2, 1992.

[9] X. Li *et al.*, "Sapper: A language for hardware-level security policy enforcement," in *Proc. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014, pp. 97–112.

[10] S. Krstic, J. Yang, D. W. Palmer, R. B. Osborne, and E. Talmor, "Security of SoC firmware load protocol," in *Proc. IEEE HOST*, May 2014, pp. 70–75.

[11] M. Sastry, I. Schoinas, and D. Cermak, "Method for enforcing resource access control in computer system," U.S. Patent 20 120 079 590 A1, Mar. 29, 2012.

[12] M. Miettinen, S. Heuser, W. Kronz, A. Sadeghi, and N. Ashokan, "ConXsense: Automated context classification for context-aware access control," in *Proc. ASIACCS*, 2014, pp. 293–304.

[13] M. Conti, B. Crispo, F. Fernandes, and Y. Zhauniarovich, "CRêPE: A system for enforcing fine-grained context-related policies on Android," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 5, pp. 1426–1438, Oct. 2012.

[14] R. Hull, B. Kumar, P. F. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas, "Enabling context-aware and privacy-conscious user data sharing," in *Proc. IEEE Int. Conf. Mobile Data Manage.*, Jan. 2004, pp. 187–198.

[15] J. Goguen and J. Meseguer, "Security policies and security models," in *Proc. IEEE Symp. Security Privacy*, Apr. 1982, pp. 11–20.

[16] T. Amtoft, S. Bandhakavi, and A. Banerjee, "A logic for information flow in object-oriented programs," in *Proc. 33rd ACM SIGPLAN-SIGACT Symp. Principles Program. Lang. (POPL)*, Jan. 2006, pp. 91–102.

[17] B. Alper and F. B. Schneider, "Recognizing safety and liveness," *Distrib. Comput.*, vol. 2, no. 3, pp. 117–126, 1987.

[18] N. Borisov, R. Johnson, N. Sastry, and D. Wagner, "Fixing races for fun and profit: How to abuse Atime," in *Proc. 14th USENIX Secur. Symp.*, Jul. 2005, pp. 303–314.

[19] S. Ray and Y. Jin, "Security policy enforcement in modern SoC designs," in *Proc. IEEE ICCAD*, Nov. 2015, pp. 345–350.

[20] A. Basak, S. Bhunia, and S. Ray, "Exploiting design-for-debug for flexible SoC security architecture," in *Proc. IEEE DAC*, Jun. 2016, pp. 1–6.

[21] P. Patra, "On the cusp of a validation wall," *IEEE Design Test Comput.*, vol. 24, no. 2, pp. 193–196, Mar. 2007.

[22] C. P. Pfleeger and S. L. Pfleeger, *Security Computing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2007.

[23] *Coresight Debug and Trace*. [Online]. Available: https://developer.arm.com/products/system-ip/coresight-debug-and-trace

[24] A. Waksman and S. Sethumadhavan, "Tamper evident microprocessors," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 173–188.

[25] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in *Proc. IEEE HOST*, Jun. 2011, pp. 67–70.

[26] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 8–9, Feb. 2010.

[27] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Trojan: Threats and emerging solutions," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLVDT)*, Nov. 2009, pp. 71–166.

[28] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 49–63.

[29] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojan detection," in *Proc. CHES*, Sep. 2009, pp. 396–410.
[30] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf.*, Jun. 2015, pp. 1–6.
[31] E. Love, Y. Jin, and Y. Makris, "Proof-carrying hardware intellectual property: A pathway to trusted module acquisition," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 25–40, Feb. 2012.
[32] H. David, J. Dubeuf, and R. Karri, "Run-time detection of hardware Trojans: The processor protection unit," in *Proc. IEEE ETS*, May 2013, pp. 1–6.
[33] Y. Alkabani, "Trojan immune circuits using duality," in *Proc. IEEE DSD*, Sep. 2012, pp. 177–184.
[34] X. T. Ngo, S. Bhasin, J. L. Danger, S. Guilley, and Z. Najm, "Linear complementary dual code improvement to strengthen encoded circuit against hardware Trojan horses," in *Proc. IEEE HOST*, May 2015, pp. 82–87.
[35] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan attacks using key-based design obfuscation," *J. Electron. Test.*, vol. 27, no. 6, pp. 767–785, 2011.

Abhishek Basak (M'15) received the bachelor's degree in electrical engineering from Jadavpur University, India, in 2010, and the Ph.D. degree in computer engineering from Case Western Reserve University, Cleveland, OH, USA, in 2016. He is currently a Research Scientist in Security and Privacy Research, Intel Laboratories, Intel Corporation, Hillsboro, OR, USA. His current research interests include the design and implementation of new security features for processor cores across all domains, analysis of attacks on machine learning platforms, and potential countermeasures. During his Ph.D., his overall research focus was mainly on system and architecture level co-design of infrastructure and primitives for enhancing the security and trustworthiness of integrated circuits. As part of his Ph.D. studies, he has also led research in low-power design of wearable, miniaturized ultrasonic imaging system for point-of-care monitoring applications. He has over ten first author publications in premier conference/journals in his corresponding research areas of interest.

Swarup Bhunia (SM'09) received the B.E. degree (Hons.) from Jadavpur University, Kolkata, India, the M.Tech. degree from the IIT Kharagpur, Kharagpur, India, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA. He is currently a Professor with the University of Florida, Gainesville, FL, USA. Earlier, he was appointed as the T. and A. Schroeder Associate Professor of Electrical Engineering and Computer Science with Case Western Reserve University, Cleveland, OH, USA. He has over ten years of research and development experience with over 200 publications in peer-reviewed journals and premier conferences. His research interests include hardware security and trust, adaptive nanocomputing, and novel test methodologies. He received the IBM Faculty Award (2013), the National Science Foundation Career Development Award (2011), the Semiconductor Research Corporation Inventor Recognition Award (2009), and the SRC Technical Excellence Award (2005), and several best paper awards/nominations. He has been serving as an Associate Editor of IEEE TRANSACTIONS ON CAD, IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, the *ACM Journal of Emerging Technologies*, and the *Journal of Low Power Electronics*; served as a Guest Editor of the *IEEE Design & Test of Computers* (2010, 2013) and IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS (2014). He has served as a Co-Program Chair of IEEE IMS3TW 2011, IEEE NANOARCH 2013, IEEE VDAT 2014, and IEEE HOST 2015, and in the program committee of many IEEE/ACM conferences.

Thomas Tkacik received the B.S. degree from the University of Virginia in 1980, the M.S. degree from the California Institute of Technology in 1981, and the Ph.D. degree from the University of Virginia in 1986, all in electrical engineering. From 1986 to 1993, he was with General Motors Research Laboratories, focused on automotive computer architecture, VLSI design, and robotics. From 1993 to 1998, he worked at Motorola on behavioral synthesis, and then on security. In 2004, Motorola spun off Freescale, which merged with NXP in 2016. For the past 18 years, he was involved in adding increasing levels of hardware security and trust to processor SoCs used for cell phone, multi-media, and networking applications. He is an NXP Master Inventor. He has published about a dozen papers, and has over 35 U.S. patents. His current research interests include designing and testing hardware random number generators for cryptographic purposes.

Sandip Ray (SM'13) received the Ph.D. degree from The University of Texas at Austin. He was a Research Scientist with the Intel Strategic CAD Laboratories, where he was involved in the pre-silicon and post-silicon validation of security and functional correctness of SoC designs, and design-for-security and design-for-debug architectures. He is a Senior Principal Engineer with NXP Semiconductors, where he leads research and development on security validation for automotive and Internet-of-Things applications. He is the author of three books (two upcoming) and over 60 publications in international journals and conferences. His research involves developing correct, dependable, secure, and trustworthy computing through cooperation of specification, synthesis, architecture, and validation technologies. He has served as a Program Committee Member for over 40 international conferences, and as a Program Chair for the Formal Methods in Computer-Aided Design. He currently serves as an Associate Editor of the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS and Springer HaSS journals.