Resilient System-on-Chip Designs With NoC Fabrics

Atul Prasad Deb Nath, *Student Member, IEEE*, Srivalli Boddupalli, *Student Member, IEEE*, Swarup Bhunia, *Senior Member, IEEE*, and Sandip Ray, *Senior Member, IEEE*

***Abstract*— Modern System-on-Chip (SoC) designs integrate a number of third party IPs (3PIPs) that coordinate and communicate through a Network-on-Chip (NoC) fabric to realize system functionality. An important class of SoC security attack involves a rogue IP tampering with the inter-IP communication. These attacks include message snoop, message mutation, message misdirection, IP masquerade, and message flooding. Static IP-level trust verification cannot protect against these SoC-level attacks. In this paper, we analyze the vulnerabilities of system level communication among IPs and develop a novel SoC security architecture that provides system resilience against exploitation by untrusted 3PIPs integrated over an NoC fabric. We show how to address the problem through a collection of fine-grained SoC security policies that enable on-the-fly monitoring and control of appropriate security-relevant events. Our approach, for the first time to our knowledge, provides an architecture-level solution for trusted SoC communication through run-time resilience in the presence of untrusted IPs. We demonstrate viability of our approach on a realistic SoC design through a series of attack models and show that our architecture incurs minimal to modest overhead in area, power, and system latency.**

***Index Terms*— SoC security, NoC security, resilient architecture, untrusted IPs, trusted SoC.**

I. Introduction

MOST modern electronic systems are architected as System-on-Chip (SoC) designs by integration of predesigned hardware blocks, — referred to as “Intellectual Properties” or “IPs”, — which communicate through a variety of network-on-chip (NoC) fabrics to realize the system functionality. In current practice, an SoC design house procures IPs from third-party vendors across the globe. The use of such third-party IPs (3PIPs) vastly accelerates development of complex system functionalities and facilitates meeting of aggressive time-to-market constraints. However, the global complex supply-chains involved in 3PIP development and delivery make it difficult to ensure quality and trustworthiness of 3PIPs. It is common for many IPs to contain errors and misconfigurations that can undermine the security of the entire system and make it vulnerable to malicious, in-field exploits. Given the high design complexity of modern SoC designs, as well as the shortening validation cycle in response to time-to-market constraints, it is impossible to explore the large vulnerability space during SoC security validation. Unsurprisingly, recent years have seen several escapes of security bugs to post-silicon or even deployment, often resulting in costly and error-prone in-field patches, and damage to company profits and reputation [1]–[3].[1]

An important class of security vulnerabilities pertains to the *communication* of IPs through the NoC fabric. NoC fabrics have become popular in recent years, as it is getting increasingly difficult to scale traditional crossbar and bus-based communications with the increasing number of IPs. NoCs realize on-chip communication with a collection of routers connected in a topology customized for the target SoC, and can enable integration of diverse, heterogeneous IPs with a variety of interfacing protocols [5], [6]. Unfortunately, NoCs can introduce some unique security challenges. First, since NoCs connect all the SoC components in a single substrate, attacks on on-chip networks can easily undermine coordination and communication among IPs with potentially catastrophic system-level effects. Consequently, trustworthiness of modern SoC designs critically depends on resilience and robustness of on-chip networks. Furthermore, while the same individual IPs are used across different SoC products (and hardened through reuse), the NoC topologies and consequent communication pattern among IPs are typically unique to a specific SoC target. This makes it more likely for security vulnerabilities in a specific communication sequence to be missed or overlooked in validation and escape to silicon or production. Finally, since security vulnerabilities in communication can stem from a collaboration among malicious hardware, firmware, and software components, detecting such vulnerabilities is beyond the scope of static IP-trust verification tools [7]–[9].

In this paper, we introduce a security architecture that ensures resilience of SoC designs against attacks on NoC communications. The goal of the architecture is to ensure that a malicious IP cannot subvert the communication and

Manuscript received July 23, 2019; revised November 25, 2019 and February 14, 2020; accepted February 19, 2020. Date of publication March 2, 2020; date of current version March 24, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Chip-Hong Chang. *(Corresponding author: Atul Prasad Deb Nath.)*

The authors are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: atulprasad@ufl.edu; bodsrivalli12@ufl.edu; swarup@ece.ufl.edu; sandip@ece.ufl.edu).

Digital Object Identifier 10.1109/TIFS.2020.2977534

[1]Hicks *et al.* [1] explored a total of 301 bugs obtained from commercial errata documents/bug reports [2], [3] and demonstrated the security criticality of 28 bugs. Documented reports have been found on the exploitation of direct jumps to escape the sandbox of AMD CPUs and execute arbitrary code outside the sandbox; such violation can cause illegal root access and execution of malicious piece of code [2]. Intel published an errata document that enlists 129 known bugs of Intel’s Core 2 Duo processor family [3]. The presence of such security critical bugs can severely affect trusted SoC operation leading to functional failure [4].

coordination of the entire system. Our key idea is to develop a hierarchical, distributed architecture that enforces system-level resilience through a collection of local and global policy enforcements. To realize this approach, we develop an adversary taxonomy covering major communication attack models for NoC, and show how the architecture provides protection and resilience under a reasonable trust model. In particular, we show (1) how to ensure resilience against communication attacks through a collection of fine-grained system-level security policies, and (2) how to implement such fine-grained policies through a centralized Security Policy Engine (SPE) and distributed Satellite Units (SUs) even in the context of untrusted communication.

In spite of the ubiquity of NoC fabrics in modern SoC designs and the consequent critical need for a security architecture to protect the system against attacks on the fabrics, we found that the current research work is inadequate to address this problem. In Section VIII we provide a fuller comparison of the scope of our work with existing literature. Previous works on NoC security have focused on resilience against specific Trojan attacks, data leakage, and flooding attacks. However, security vulnerabilities for SoC designs with NoC fabrics go significantly beyond these attacks, *e.g.*, in addition to jamming or flooding, an IP can misdirect an in-flight message, masquerade as another IP, observe a message it is not authorized to observe, etc. We believe our work provides the first security architecture that enables comprehensive, systematic implementation of resiliency mechanisms against the spectrum of NoC communication attacks.

The primary contribution of the paper is an approach for implementing fine-grained security policies to protect an SoC design against a variety of attacks on communications through NoC fabrics, resulting in an architecture that addresses the critical problem of enabling systematic implementation of resilience mechanisms in modern SoC communications: in the absence of such a framework, the architect must resort to implementing security protection in an ad hoc manner across the entire SoC and interleaving those mechanisms with functionality. Such practice provide key source of design complexity in security architecture of today's SoC designs, and result in significant complexity, bugs, and misconfigurations [4]. It also makes it difficult to verify the implementations since such verification needs to consider the entire SoC and conflation of functionality with policy implementation.

In summary, the paper makes the following important contributions.

- We provide an architectural framework to ensure resilience of system communication in the presence of untrusted 3PIPs.
- We demonstrate viability of a centralized security architecture, in conjunction with distributed units, for security enforcement in the context of NoC-based SoC designs.
- We perform a comprehensive security analysis of NoC fabrics for each critical category of communication attacks, *i.e.*, message mutation, message misdirection, IP masquerade, message observability, and message flooding.
- We implement the proposed architecture on a realistic SoC design and showed that it incurs decent overhead in terms of area, power, and latency costs.

The remainder of the paper is organized as follows. Section II provides the relevant background SoC security policies, on-chip networks, and policy implementation through security architectures. Section III discusses our high-level idea and explains the key distinctions of our approach from other related architectural frameworks for SoC security. Section IV highlights the key security requirements of system level communication through the on-chip network. We give details of our solution in Section V, and demonstrate various policies and implementation details in Section VI. Our experimental results are described in Section VII. We discuss related work in Section VIII, and conclude in Section IX.

II. Background

A. SoC Security Policies

Security policies govern the confidentiality, integrity, and availability requirements of assets, *i.e.*, sensitive data or information in SoC designs. Such assets include system collaterals and artifacts (*e.g.*, cryptographic keys, defeature bits, manufacturer firmware, etc.) and sensitive end-user information (*e.g.*, health information, financial transaction records, contacts, emails, etc.). These assets are usually sprinkled across the entire SoC over various IP blocks. Security policies help IP designers and SoC integrators map the security requirements of the assets into actionable design constraints. Representative examples of SoC security policies would be as following:

- Example 1: At system boot phase, the data transmitted by the crypto blocks cannot be snooped by any other SoC component other than the target IP.
- Example 2: Secure key containers in the SoC platform are allowed to be updated during silicon validation phase; no updates are allowed after production.

Example 1 is a confidentiality requirement and Example 2 is an integrity constraint. There are three crucial aspects in security policy designs: (1) how to protect an asset; (2) from whom to protect; and (3) when to protect. Consequently, policy requirements vary significantly based on the state-of-execution (*e.g.*, normal operation, boot mode, crypto mode, etc.) and stage in SoC life-cycle (*e.g.*, architecture, implementation, security validation, etc.).

B. NoCs and Their Role in SoC Communication

The primary mechanism of coordination for IPs in an SoC involves message-based communication. Consequently, design of communication fabrics is a crucial activity of SoC design. In the past, communication fabrics were implemented through a point-to-point, cross-bar, and bus-based architecture. These architectures had the benefit of being simple, but unfortunately incur prohibitive expense in design complexity, area, and power dissipation for modern SoC designs with hundreds of integrated IPs. Fabrics based on on-chip networks have the advantage of being scalable and power-efficient, and have consequently superseded other communication architectures in modern SoCs. An NoC involves a collection of routers connected to realize a target topology. NoC topologies for many industrial SoC designs realize a tree network, although other networks such as cycle or mesh are also in use [10]. Routers in an NoC typically include configurable routing tables which can be reconfigured if necessary by the operating system at boot-time or under certain circumstances during execution (*e.g.*, on detection of congestion or transmission failure in certain paths). The configurability of routing tables enables the reuse of the same router IP to realize different network topologies, and enables error-tolerant communication during execution. A key advantage of NoC architecture is the ease of implementing power management with low overhead: low power mode can be implemented by simply shutting off (or reducing high-speed functionality) routers in the sub-network when message communication through the sub-network is reduced.

III. An Architectural Approach to SoC Security

Our work entails a security architecture framework for NoC-based SoC designs. The architecture can be used to systematically implement diverse security policies to protect inter-IP communications against a variety of communication attacks. The goal is to provide a platform for the security architect to implement the policies that ensure system resiliency when the inter-IP communication is susceptible to various attacks. The result is a trusted, robust SoC architecture even when built through integration of untrusted IPs.

Our approach is inspired by prior work on the E-IIPS architecture, [11], [12], an architectural framework developed for flexible implementation of SoC security policies. It includes a central "security policy engine" (SPE) and smart security wrappers implemented around each IP. The security wrappers detect security critical events at individual IPs and communicates them to SPE. SPE monitors the security state of the SoC at system execution time and enforces the policy restrictions upon notification of security critical events by the wrappers. The E-IIPS architecture was used to implement diverse SoC security policies. The proof-of-concept implementation of E-IIPS was demonstrated on an SoC model with point-to-point interconnects.

However, a key limitation of such architecture is the reliance on trusted communication between the IPs. The architecture operated correctly only under the condition that the inter-IP communications are tamper-proof (*i.e.*, messages cannot be tampered or interrupted while in flight) and are delivered within a short communication latency. This enabled efficient implementation of a centralized security policy engine to detect suspicious activities in different IPs and their security wrapper interfaces. Given these requiremements, previous work targeted only SoC designs with point-to-point and crossbar communications. However, these assumptions are infeasible for practical SoC designs where communications are implemented through NoC fabrics. In particular, since the policies are controlled by a centralized controller, every inter-IP communication must be routed to the policy engine to check for trustworthiness; in an NoC-based SoC, the result would be a prohibitive explosion in SoC communication and corresponding traffic congestion. Furthermore, the assumption of trustworthy communication is invalid for an NoC susceptible to communication attacks. In fact, communication is a critical target of attack in modern SoC designs; furthermore, they are difficult to detect during security validation as they might involve subtle and complex interleaving of inter-IP messages that are difficult to exercise.

Our work shows how to achieve system-level resiliency in the presence of untrusted on-chip communication through SoC security policies even in the context of NoC fabrics. Our approach removes the bottleneck of a centralized policy controller by introducing *satellite units (SUs)* integrated with individual IPs; policy decisions that can be handled locally in an IP are handled by these units while global decisions are taken by a centralized control. We show how this hierarchical implementation enables disciplined security protection for NoC communication with little additional overhead in performance or network congestion. The attack models and subsequent protection mechanisms considered in this paper are analyzed to account for the area and power cost as well as the latency and congestion due to the communication overhead. Note that development and augmentation of a centralized security architecture, originally developed for point-to-point SoC models, to work in tandem with distributed architectural units on an on-chip interconnect fabric is a non-trivial research task, as reflected in many of the design choices of our architecture. The proposed architecture is developed with augmented centralized security architecture that work in conjunction with distributed policy enforcement units, enhanced with standardized interfaces and communication protocol compatible with open-source bus and NoC protocol while preserving the core functionality of security policy enforcement on detection of violating events.

IV. Attack Model Analysis for NoC-Based SoC Designs

The advent of NoCs in on-chip communication has introduced novel vulnerabilities and attack surfaces compared to traditional point-to-point, shared bus, and cross bus-based SoC interconnects. The on-chip communication among the IPs via NoC components (*e.g.*, routers) can be maliciously altered to launch attacks leading to information leakage, data corruption, denial-of-service, etc. The following two scenarios illustrate the nature of these attacks. Albeit simplified, they are sanitized versions of actual vulnerabilities exercised on industrial SoC designs during security validation.[2]

[2]While security flaws in industrial NoCs have not been reported to date, the increased adoption of 3PIPs by fabless semiconductor companies make such threat models comprised of rogue NoC IPs significantly relevant [13]. We participated in many discussions with researchers from industry, academia, and government during technical conferences and discussion forums where NoC security flaws have been identified as a critical issue that needs to be addressed to ensure trusted system execution. Indeed, the motivation for the threat models and issues presented in the paper came from such discussions. The vulnerability exercises presented in this work reflect the authors' own industry experience and the common practices of security and reliability assurance in industrial NoCs. For instance, the resilience package of FlexNoC [14] protects packet headers to prevent masquerading and message mutation attack, checks the validity of packets and control registers to prevent non-observability and message misdirection attack, and implements strategies for transaction time-out and unit duplication to prevent denial-of-service attack.

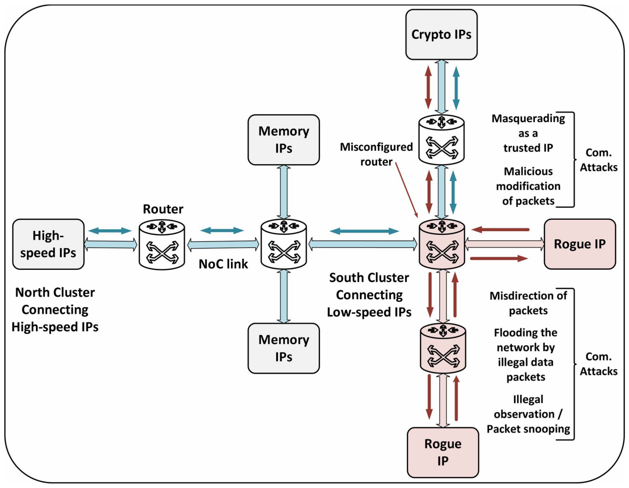
Crypto IPs
Memory IPs
Misconfigured router
Masquerading as a trusted IP
Malicious modification of packets
Com. Attacks
Router
High-speed IPs
NoC link
Rogue IP
South Cluster Connecting Low-speed IPs
North Cluster Connecting High-speed IPs
Memory IPs
Misdirection of packets
Flooding the network by illegal data packets
Illegal observation / Packet snooping
Com. Attacks
Rogue IP

Fig. 1. An illustrative depiction of potential vulnerabilities in on-chip communication amid the presence of untrusted IPs and compromised NoC (the tree topology network in this figure is inspired by realistic SoC designs in current industry practice).

Example 1: A rogue IP A (potentially running malicious software at a high privilege level) misconfigures the routing tables for a router to misdirect sensitive information to an accomplice IP B leading to eventual information leakage and violation of system confidentiality.

Example 2: A rogue IP A transmits a large number useless but valid message packets, filling the ingress and egress queues of the different routers. This can result in high network latency, packet drops, denial of service, or even overflow of ingress and egress queues leading to confidentiality or integrity breaches.

To our knowledge, there is no standard, universally accepted taxonomy for attack models in NoC-based SoC designs. In industrial practice, SoC integration teams usually define the key security requirements of on-chip communication based on system use-case scenarios and protection requirements of SoC assets at different points of system execution. We have developed a taxonomy of attacks on NoCs to fill this crucial gap. The attack taxonomy is depicted in Fig. 2. While not comprehensive, it covers a broad range of communication vulnerabilities of on-chip interconnects. Attacks are classified broadly into two categories *i.e.*, (i) attack surface and (ii) action characteristics. Each category is further classified into attack mediums, attack origin, attack payloads, etc. Note that any given attack can lead to more than one kind of system-level impact. In general, any malicious IP and compromised NoC can disrupt the on-chip communication through message passing, resource sharing, and altering the operation and data flow. However, based on the system-level impact of different attack models, the security requirements of on-chip communication in NoC-based SoC designs can be broadly categorized as follows:

1) *Misdirection prevention*: An unauthorized IP should not be able to act as a *diverter*, *i.e.*, misdirect the messages from their original destinations in the NoC fabric. If this requirement is violated, a rogue IP can maliciously change the router configuration to divert messages to invalid or unauthorized destination.

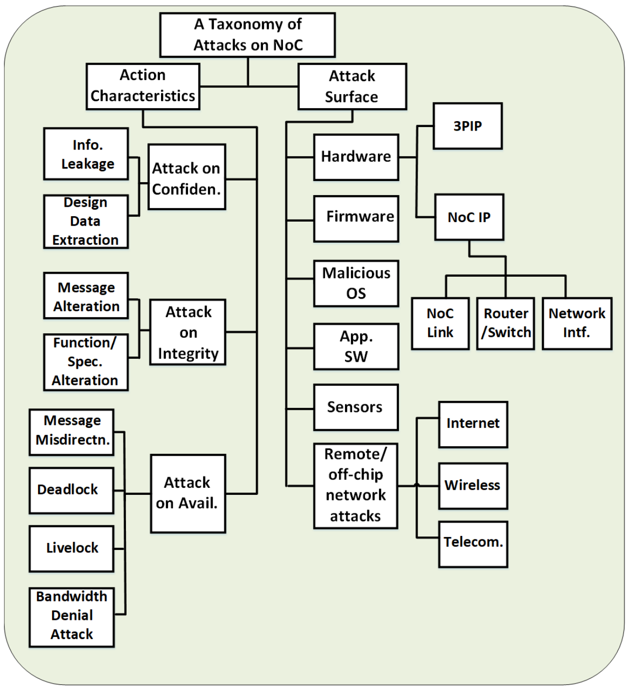

Fig. 2. A taxonomy of attacks on NoC.

2) *Masquerade prevention*: An unauthorized IP should not be able to act as a *masquerader*, *i.e.*, disguise itself as a different (trusted) IP. If this requirement is violated, a rogue IP can to get access to secure messages, request unauthorized service, or disrupt inter-IP communication.
3) *Message immutability*: An unauthorized IP should not be able to act as a *modifier*, *i.e.*, alter the messages passing through on-chip network. If this requirement is violated, a rogue IP can maliciously change or corrupt data packets between two trusted interacting IPs, possibly in collusion with a compromised NoC component.
4) *Non-observability*: An unauthorized IP should not be able to act as a *active/passive reader* and snoop/read/collect messages illegally through the on-chip network. If this requirement is violated, a rogue IP can to get access to secure messages and leak sensitive information.
5) *Flooding and congestion prevention*: An unauthorized IP should not be able to act as a *Denial-of-service (DoS) attacker* and generate or misdirect messages illegally to create flooding and congestion in the on-chip network. If this requirement is violated, a rogue IP can disrupt secure NoC communication leading to operational failure of the SoC.

The above categorization of NoC security requirements is based on the *system-level impact* of potentially malicious activity in the NoC rather than IP-level security vulnerabilities. Furthermore, collusion among untrusted/compromised IPs can lead to attack instances causing security violations in multiple categories. The consequence of these attacks leads to the violation of high-level security requirements of the SoC assets, namely confidentiality, integrity, and availability.

TABLE I
TRUST MODEL OF THE PROPOSED FRAMEWORK

	IP core and NoC Fabric	**Test and Debug Wrapper**	**SUs at IPs/Routers**	**SU to SPE Comm.**	**SPE**
Trust assumptions	Untrusted	Trusted	Trusted	Trusted	Trusted

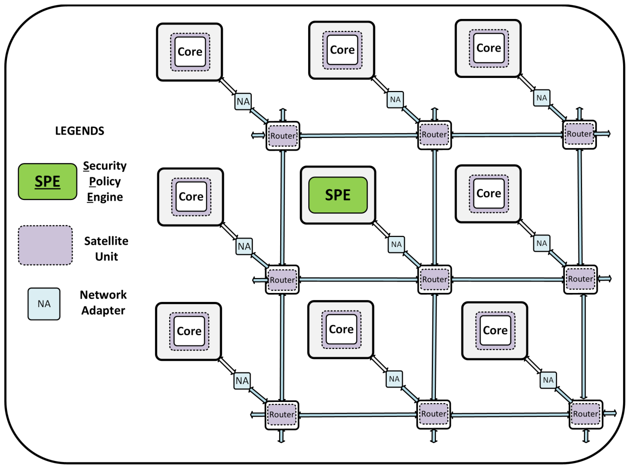

Fig. 3. A high-level overview of proposed security architecture in a tiled many core SoC design with on-chip network (a mesh topology network is shown in the figure to illustrate the compatibility and scalability of our architecture with complex SoC designs).

V. RESILIENT ARCHITECTURE FOR UNTRUSTED IPS

A. Architectural Components

Our architecture includes two major components: (1) a distributed *Satellite Unit (SU)* at each IP and router of the SoC platform to monitor security critical events, assert appropriate security controls, and establish a standardized communication among the IPs, routers, and the security policy engine; and (2) a centralized and dedicated *Security Policy Engine (SPE)* connected to the NoC fabric as an infrastructure IP for implementing, modifying, and updating SoC security policies and system-level security monitoring. Fig. 3 provides a high-level overview of our framework. It shows how the proposed framework can be implemented on complex heterogeneous tiled SoCs with on-chip networks.

1) Satellite Units (SUs): We implement satellite units (SUs) at each IP and router block. They are designed with security wrappers, interface action detection logic, and frame-based interface for communication with centralized SPE. These units are active agents designed to make mitigation decisions without intervention from SPE based on local information. This functionality is critical to ensure that the NoC bandwidth is not clogged with communications between IPs and SPE. Consequently, these are complex designs incorporating enough "intelligence" to act as local policy brains, while ensuring that cost in area and power are not excessive. These units monitor security critical events of interest, send triggered event information to the central SPE, and enforce security policy restrictions in co-ordination with SPE.

a) Security wrappers: Security wrappers are designed by extending the IEEE P1500 test wrappers and the ARM$^{®}$Coresight™ debug wrappers. IEEE P1500 standard is comprised of a core wrapper architecture, primarily to support access to internal signals of the cores. The wrappers include boundary register cells for the functional I/O ports of the IP cores. We tailored the standardized wrapper and boundary registers to monitor and store the events of interest. In addition, we achieved enhanced flexibility in policy implementation by obtaining improved controllability and observability over required signals through Macrocells, designed in accordance to ARM$^{®}$Coresight™ architecture. These local modules aka Macrocells are designed and augmented with configuration register interfaces to function as debug wrappers in conjunctions with the customized test wrappers. These are instantiated at each IP and interfaced with the wrappers for retrieving event information. The wrappers extract local security critical events relevant to the policies based on different operating states of the corresponding IP and router. We categorized the events according to the IP types. Examples of such categorization include events of the DMA (Direct Memory Access) read/write requests to specific address ranges in memory IPs, system controller interrupts in processor IPs, etc. The event related meta-data (*e.g.*, DMA burst size, page size, address range, etc) is extracted by the wrappers to be utilized by SUs and communicated with SPE.

b) Interface actions detectors: Our goal is to prevent the propagation of malicious activity through the NoC and result in system-level impact. The interface action detection logic monitors critical communication events. On detection of a triggered event, the current transaction is stalled by the SU and the event information is sent to the SPE. To avoid potential bottleneck due to traffic congestion, the logics are configured at boot time to detect and verify a specific set of security critical events based on the IP type and attempted transactions. To implement the trigger mechanism, IP interface signals are classified into four basic types (*e.g.*, control, data, global, and test signals), and logics are designed to monitor the control and data signals of the output interface.

c) Frame generation interface: The micro-architectural events are monitored, analyzed, and stored by the SUs. The communication interface of SU generates frame-based packets with security-critical event meta-data for IP-specific temporal events and critical actions detected by the activity detection circuitry. The frame-based interface facilitates a standardized communication for sending data packets over the NoC. To enable portability of event detection logic for multiple IPs, SUs include configuration registers which are configured at boot-time by the SPE. In this work, our assumption (Table I) is that the communication between SPE and SU is trusted. In practice, this trust is enforced in various ways, *e.g.*, through special encryptions and communication channels. For such methods to be effective, it is critical that the number of such messages is significantly less than normal traffic, which is realized in our architecture by ensuring minimal communication overhead between the SUs and SPE (refer to section VII-C3).

2) Security Policy Engine (SPE): A primary building block of our resilience architecture is a centralized "security brain", *i.e.*, a single IP called security policy engine (SPE) which enforces all system-level security policies. We develop fine-grained policies that enable on-the-fly detection and mitigation of suspicious run-time NoC communications that may affect the system-level functionality. SPE is developed from the ground up to implement, adapt, modify, and upgrade diverse system-level security policies. It is connected to the NoC fabric as an infrastructure IP and interfaced with the NoC components (routers and links) to adhere to on-chip-network protocol. The key functionalities of SPE is to configure registers at SUs for security critical events at boot time, analyze the packets sent by the SUs at run-time, enforce security policies by sending control signals as standardized frame-based packets, and update the security state of the overall system. The microarchitecture of SPE is primarily comprised of two major components namely, a security buffer and the policy engine.

a) Security buffer: The security buffer acts as the storage for buffer frames and interfaces with the Policy Engine through a buffer controller. The buffer storage is implemented as a static segment scheme that permits variable length of segments based on the size of metadata. The buffer controller allows access to the events logs through the buffer ports. The synchronization and coherence between the SUs and SPE is maintained by the buffer control logic with respect to segment sizes, read / write speeds, and frequency of events.

b) Policy engine: The policy engine is a micro-controlled implementation on a processor core. It functions as a micro-controlled state machine to assert and de-assert control signals to enforce the policies. It is comprised of dedicated instruction memory to store the policies in firmware. Intermediate computations are facilitated by a dedicated data memory inside the block. The module is augmented with configuration registers to activate particular sets of implemented policies based on the use cases of system. The register configuration is performed by a series of fuses, anti-fuses, and multiplexers.

B. Trust Model

Security assurance in SoC designs relies on the trust assumptions made for design components and parties involved in the supply-chain. For this paper, the trust model assumes a trusted SoC integration house that procures 3PIPs from multiple vendors with a varying level of trust. Table I shows our assumptions regarding the presence of untrusted IPs and their interaction with other on-chip components. This trust model assumes that IPs themselves (and the NoC fabric) might be untrusted but the wrappers, SUs, SPE, and the communication of SUs to SPE are trusted. This is a natural trust model from an SoC integration perspective: IPs are procured from a global supply chain, but the above components must be architected, designed, and standardized by the SoC integration house. For instance, a 3PIP might have a hardware Trojan located at strategic locations like control logic, data-path, the storage section, etc. or execute malicious firmware/software. Furthermore, we account for collusion among the untrusted IPs. For instance, we consider scenarios where a malicious IP launches attacks with cooperation of a compromised NoC component. In current SoC integration, IPs are classified routinely with various trust categories. The policies are designed to ensure that untrusted components do not have adverse system-level effects or the ability to compromise other system components.

Remark 1: Note that we consider the IP test and debug wrappers trustworthy. In practice, we can rely on the trustworthiness of the test and debug wrappers for two reasons. First, unlike any other functional IP (processor, memory, hardware accelerators, etc.), the wrappers are integrated to the SoC designs as infrastructure IPs. The primary purpose of such infrastructure IP is to facilitate the testing and debugging. Hence, the standard test and debug wrappers are highly validated compared to traditional functional IPs procured from third-party vendors. Second, the wrappers in our work are re-purposed and modified versions designed in accordance with existing architecture to achieve the functionality of security wrappers by while avoiding the naive extraction of the all data, control, and status signals; thus, customized to cater to our security goals. In the proposed design flow, these are highly validated by the security architect of the SoC and configured to monitor the security critical events and enforce the corresponding security policies.

Remark 2: Any viable security architecture must assume certain design components are trusted; if every design component can be compromised or malicious it is generally impossible to ensure resiliency of the overall system. Since SoC supply chain includes many players (*e.g.*, IP vendors, SoC integration house, foundry, various test and validation organizations, end users, etc.), SoC trust models in practice are defined from the perspective of a specific player in this supply chain and the goal is to ensure that the assets introduced by that player are not compromised by malicious activity of others. Since the SoC security architecture is designed by the integration house, the goal of our resiliency architecture is to ensure that *other players*, in particular potentially malicious 3PIPs, cannot compromise the overall system security. Given this goal, it is natural for our trust model to assume that the design components introduced specifically for system integration are trusted, while individual IPs may be untrusted. In particular, the design components constituting our security architecture (*e.g.*, satellite units, SPE) are trusted since they are developed and maintained by the SoC integration house.

VI. Implementing NoC Security Resilience Through Policies

We implement resilience against malicious NoC communication through a collection of fine-grained security policies implemented in SUs and SPE. The policies we have implemented include protection against a variety of attacks that violate confidentiality, integrity, non-repudiation, authentication, and availability requirements. In this section, we discuss some representative policy implementations. Note that these

TABLE II
REPRESENTATIVE SECURITY CRITICAL EVENTS AND ASSOCIATED POLICIES BASED ON ATTACK TYPES

Attack Type	**Attack Mechanism**	**System Level Impact**	**Illustrative Security Critical Events**	**Associated Security Policies**
Illegal Observability and Packet Misdirection	Malicious IP attempts to access secure memory region and leaks information through a compromised router.	Violation of non-observability and non-misdirection property (Information leak, breach of confidentiality).	DMA request type, requested memory address, base address of requested tile, illegal input queue assignment, flit register update, virtual channel allocation, arbiter activation, output queue assignment.	Valid DMA read request, transfer request in DMA table in address range, entry in input queue corres. to valid signal, entry of incoming flit in correct channel queue, activation of routing unit to corres. header flits, valid route assignment based on routing policy, valid update of flit register, arbiter activation with valid header flit, valid entry in output queue.
Masquerading and Message Mutation	Malicious IP attempts to masquerade as a trusted IP by modifying packets originating from itself and data packets en route traversing through a compromised router.	Violation of non-masquerade and message immutability property (man-in-the-middle attack, data corruption, breach of integrity).	Base address value assignment, base address register update, local address in DMA transfer table, header flit value assignment, header flit value update, flit register update at router.	Base address register assignment/update at boot, valid base address in DMA transfer entry, valid base register value in initiator module, correct value update at flit register, valid body/tail flit with corres. header in input/output queue, valid header flits for routing unit activation, arbiter activation, and link arbitration.
Flooding and Deadlock/Livelock	Malicious IP launches DoS attacks by flooding the network with spurious packets and altering routing paths to create deadlock/livelock.	Violation of flood prevention and non-misdirection property (network congestion, breach of availability).	Number of packets emanating from an untrusted IP and router at a given amount of time, failure to respond to access requests in specified time, illegal flit register update, virtual channel allocation, arbiter activation, output queue assignment.	Valid value of message count threshold for IPs/routers, configuration of the message counter at SUs, assignment of response timing requirements on IP basis, valid specification deadlock free and livelock rule by ensuring valid route assignment based on routing policy, valid update of flit register, arbiter activation with valid header flit, valid entry in output queue.

policies are only illustrative, and do not represent a comprehensive demonstration of the capability of the infrastructure. However, they *do* show how one can approach the problem of implementing resilience requirements as policies for a specific, targeted trust model. Similarly, determining optimal number of policies to implement for effective protection is not within the scope of the paper: the primary contribution of the paper is to provide a mean for implementing the policies systematically once the architect has determined what they want to implement. In Table II, we present a summary of illustrative security critical events and corresponding policies classified in to accordance to attack type, mechanism, and system level impact.

A. Attack Scenario I: Non-Observability and Misdirection Prevention

Following are two typical requirements of non-observability and misdirection prevention.

- Representative Policy 1: Untrusted IPs are prohibited from exploiting DMA to secure memory range.[3]
- Representative Policy 2: The routing units of untrusted NoC routers are subject to update only at secure boot or at run-time by a trusted IP.

[3]A formal description of the security policy can be provided by three tuples: <timing condition>, <predicate>, and <action>. For a given policy of secure memory range access via DMA, the timing condition will state when the request is made, the predicate tuple will state which entity placed the request to access secure memory, and the action tuple will state either to approve or deny the request based on the information associated to timing condition and predicate. A detailed description of such tuple-based formal representation of security policies can be found in our prior work [15].

1) Attack Model: Recall from non-observability requirements that untrusted IPs should not be able to observe information intended for other SoC components. We consider a malicious processor IP attempting to violate non-observability in two ways. It can attempt to read from the protected memory region of on-chip memory and it can misdirect data packets to itself or to an illegal destination through a compromised router. In this scenario, we assume the presence of a malicious IP that can initiate an illegal DMA read request to the secure memory region and alter the router configuration to misdirect data packets to illegal destinations including itself.

2) Flow of Operation: Fig. 4 shows the flow of events for a security policy to protect against the attack.

- During boot phase, the wrappers and interface action detectors at SUs of IPs and routers are programmed by the SPE for event detection. Events include triggers for micro-architectural flows, *e.g.*, value of program counter within secure memory address range, read access to secure data memory, incoming flit to valid channel queue, flit register content update, allocation to proper virtual channel, etc.
- When a DMA transfer request is detected by the interface action detectors at the DMA table corresponding to the untrusted IP, the request address and access type is evaluated by the wrapper logic. If there is no violation, the transfer request is not stalled. However, upon detection of a DMA request to illegal memory addresses, a security-critical event notification is triggered, the violation is logged at the security wrappers, and the entry request for DMA is discarded by the SU. The frame generation interface at SU form frames with event logs

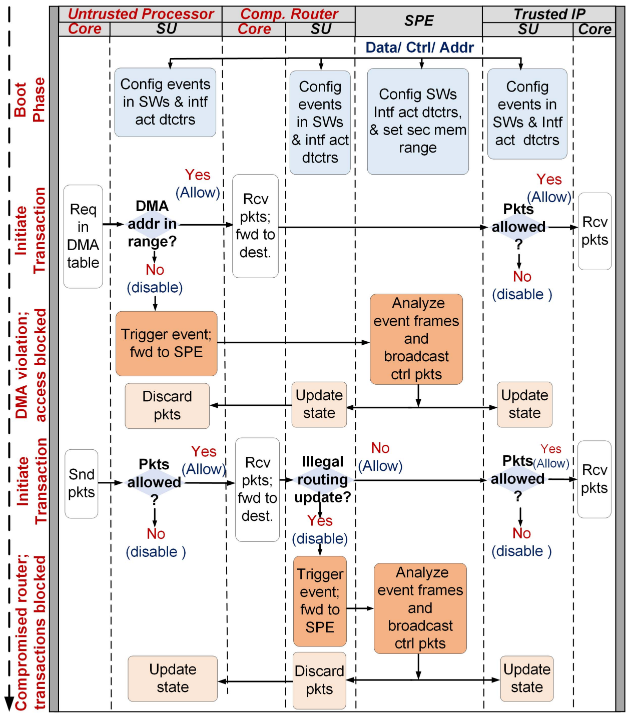
Untrusted Processor
Core
SU
Comp. Router
Core
SU
SPE
Trusted IP
SU
Core
Data/ Ctrl/ Addr
Boot Phase
Config events in SWs & intf act dtctrs
Config events in SWs & intf act dtctrs
Config SWs Intf act dtctrs, & set sec mem range
Config events in SWs & Intf act dtctrs
Initiate Transaction
Req in DMA table
DMA addr in range?
Yes (Allow)
No (disable)
Rcv pkts; fwd to dest.
Pkts allowed ?
Rcv pkts
DMA violation; access blocked
Trigger event; fwd to SPE
Analyze event frames and broadcast ctrl pkts
Discard pkts
Update state
Update state
Snd pkts
Illegal routing update?
Compromised router; transactions blocked
Trigger event; fwd to SPE
Discard pkts

Fig. 4. Message flow diagram of proposed solution for resilient SoC operation amid non-observability and message misdirection attack.

and meta-data, and send the corresponding packets through the communication interface to SPE.

- It is possible for a malicious IP to violate non-observability without memory access. The malicious processor attempts to clone and misdirect data packets through a compromised router IP. Misconfiguration of router involves micro-architectural events, *e.g.*, assigning incoming flit to invalid channel queue, illegal update of flit register content, allocation to improper virtual channel, etc. Any deviation from the standardized events in the routers is detected by the security wrapper and interface action detection logic. Consequently, the suspicious transaction is stalled by the SU and corresponding event frames are sent to SPE as packets.
- Upon reception of the packets from the SU of compromised IP or router, SPE extracts the event frames, updates the security status of the system, broadcasts packets to SUs at other IPs and routers to block subsequent DMA requests of the IP, and discard the data packets from compromised router until further verification in debug mode.

The attack scenario and mitigation technique illustrate the feasibility of policy-based solution for non-observability and misdirection prevention via proposed architecture. As the attack models are not mutually exclusive, the non-observability policies also prevent packet misdirection in this case.

B. Attack Scenario II: Masquerade Prevention and Message Immutability

We will now demonstrate how message immutability and masquerade prevention can be ensured by implementing fine-grained policies in the security architecture. Following are two typical scenarios.

- Representative Policy 1: The base address of untrusted IP cores can only be assigned and updated at secure boot time.
- Representative Policy 2: Data packet header flits can only be configured at the source but not at the router.

1) Attack Model: Masquerading and Man-in-the-Middle Attack via Packet Alteration: Consider a malicious processor IP attempting to violate the masquerade prevention property in two ways. It can attempt to illegally alter its own address *i.e.*, it can change the source address of packets and send illegal packets to the network. On the other hand, it can alter the packet headers of data en-route with the help of a compromised router. In this attack scenario, we assume that a malicious IP can illegally alter its address before sending packets and enforce a compromised router to alter the packet header flits of data traversing through it.

2) Flow of Operation: The message flow diagram for the use case scenario with corresponding policy implementation is depicted in Fig. 5. The key steps are as following:

- At boot phase, the SPE configures the wrappers at SUs of IPs and routers with the standard set of events for masquerade prevention. The set of relevant micro-architectural events include packet address check, core ID update, tile ID update, core address, address in local memory, etc.
- When the malicious IP attempts to send packets through the output interface, the interface action detection logic checks the source address of packets in accordance to boot time configuration. If there is no mismatch, the message passing event will not be stalled. Upon detection on a mismatch of source address, a violation is triggered and the SU discards the outbound packet. The frame generation interface located at the SUs of malicious IP generate frames with event logs and meta-data and sends the packet to the SPE.
- The malicious IP can violate the non-masquerade property without emanating spurious packets from its core. It can attempt to maliciously alter the header flits of data packets traversing through a compromised router and alter the source address of the header flit with an accomplice router. Such alteration involves suspicious micro-architectural events at a router, *e.g.*, alteration of header flit registers, invalid activation of routing unit by fake header flit, invalid scheduling of the arbiter by illegal header flit, etc. The wrappers at the router SU will trigger policy violation upon mismatch in any of of these events. Consequently, the suspicious event and generated packet or flit will be discarded and SPE will be notified about the violation.
- The SPE will receive the packets from the IP SU or router SU, extract the event frames, and update the system security status accordingly.

C. Attack Scenario III: Flooding Attack Prevention

We now illustrate how policy-based solutions can be implemented in the proposed security architecture to prevent

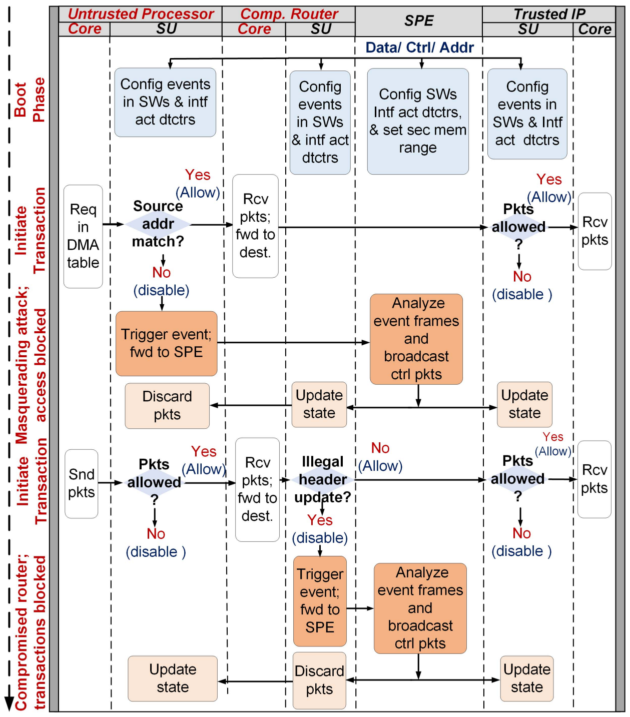
Fig. 5. Message flow diagram of proposed solution for resilient SoC operation during masquerading and message mutation attack.

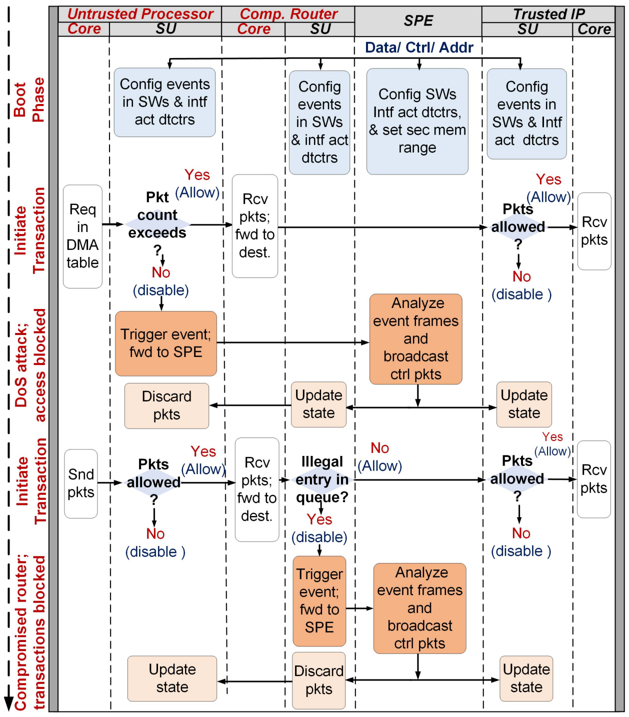
Fig. 6. Message flow diagram of proposed solution for resilient SoC operation amid flooding attack.

flooding attack by large amount of spurious and misdirected messages that can cause congestion, livelock, and deadlock situations in the on-chip network. Following are some typical security requirements.

- Representative Policy 1: During SoC operation, the number of messages sent by an untrusted IP should not exceed the threshold of maximum limit.
- Representative Policy 2: The limit on the number of packets generated by an untrusted IP cores can only be assigned and updated at secure boot time.

1) Attack Model: Flooding Attack via Spurious and Misdirected Packets: In this attack model, we consider a malicious processor IP attempting to launch a flooding attack to disrupt the secure NoC communication. It can create congestion in the network with spurious packets and misdirect the packets with the help of a compromised router to create a livelock/deadlock situation leading to eventual network congestion. Deadlock rule violation will create contention of the NoC channels whereas livelock rule violation will allow the packet to traverse in the network causing a waste of throughput, bandwidth, and latency. The protection mechanism against flooding attacks depend on the usage context of the IP. In the given scenario, we set an upper bound on the number of messages from the same source IP, implement a message counter at the SU as a part of security instrumentation, and enforce the corresponding policies.

2) Flow of Operation: Fig. 6 illustrates the message flow diagram for the attack scenario of flooding attack with the implemented policy. The operational flow can described as follows:

- During secure boot, the wrappers and interface action detection logic of SUs are configured by the SPE to detect the standard set of events for flooding attack prevention. The set of relevant micro-architectural events include setting the threshold value of messages from the source IP, valid configuration of message counter at SUs, etc.
- When the malicious IP attempts to send illegal packets in large volume through the output interface, the counter implemented at interface action detection circuitry raises a flag, the messages emanating from the rogue IP are stalled by the SU, and the outbound packets are discarded. The events frames are communicated with SPE with associated logs and meta-data.
- The malicious IP can violate the flooding prevention property without generating illegal packets from the core. The rogue IP can attempt to maliciously alter the header flits of data packets traversing through a compromised router to cause deadlock and livelock situation. Such alteration involves suspicious micro-architectural events at a router, *e.g.*, alteration of flit registers, invalid activation of routing unit by fake header flit, invalid scheduling of the arbiter by illegal header flit, etc. The wrappers at the router trigger a violation upon mismatch in any of of these events. Consequently, generated packets or flits are discarded and SPE is notified about the violation.
- The SPE receives the packets from the SU at the rogue IP or router, extract the event frames, broadcasts the event information, and update the system security status.

VII. Experimental Results

We implemented the architecture presented in Section V on illustrative SoC models, and implemented protection against

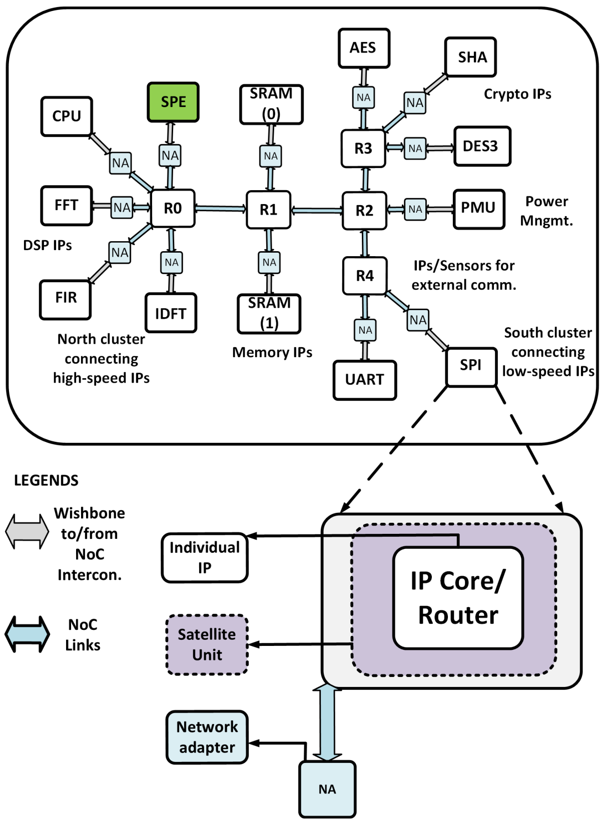

Fig. 7. ClusterSoC: Cluster-based SoC model with proposed security architecture: SPE acts as a centralized, flexible policy brain to enforce policies in collaboration with distributed satellite units (SUs).

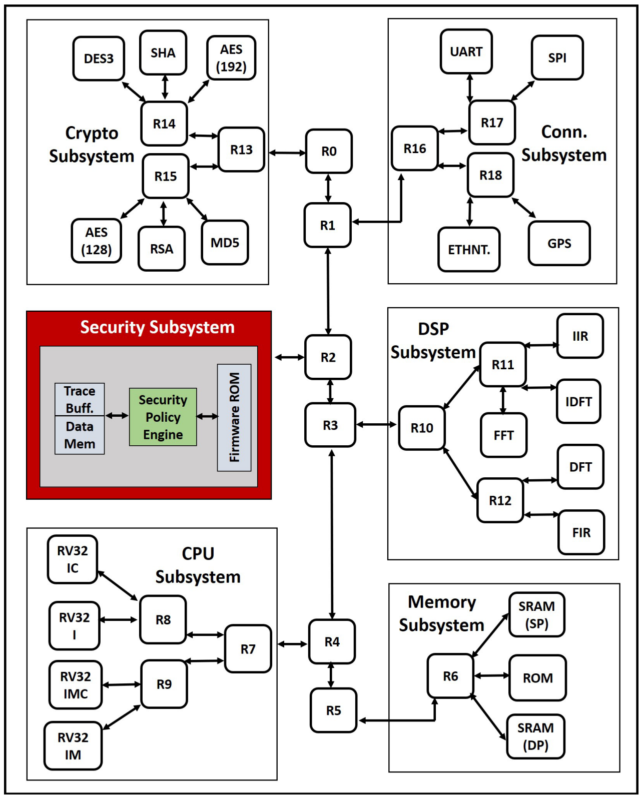

Fig. 8. AutoSoC: representative automotive SoC model with application specific subsystems.

diverse communication attacks through a systematic collection of fine-grained policies. Our experiments show that the architecture incurs modest overhead in our SoC models in terms of area, power, and network congestion (under representative message traffic patterns). Perhaps more significantly, our experimental results indicate the kind of trade-offs that must be studied to ensure viability of a similar architecture in industrial SoCs. We also study overhead in performance and latency for different locations of placement of SPE.

A. Evaluation Platforms

Given the dearth of standard open-source SoC designs for architectural study, we have been developing our own SoC models. Fig. 7 and Fig. 8 show illustrations of two different SoC models developed to analyze the proposed framework. Our models, albeit academic, is architected with many substantial SoC components and reflects the relevant features of NoC-based industrial SoC designs, including tree-based topology for routers inspired by commercial SoCs, a mix of high-speed and low-speed IPs, application specific subsystems, etc. [16]. The two models are used for two different purposes. The first model, referred to as `ClusterSoC`, is relatively simpler and more mature, which enables deeper exploration and understanding of the effects of architectural elements introduced in this paper. Most of our experiments and explorations have been performed on this model. The second model, referred to as `AutoSoC`, is more complex and inspired by industrial SoC designs targeted towards automotive applications; we use it to demonstrate the flexibility of the framework and its ability to be adopted for SoC designs of practical complexity with minimal overhead.

1) `ClusterSoC`*:* The `ClusterSoC` design includes an OpenRISC 1000 processor featuring 6 stage pipelining and 32-bit load-store RISC architecture with cache and MMU (Memory Management Unit) support. Two dual port SRAMs are integrated as memory modules. Three crypto modules *i.e.*, AES, 3DES, and SHA are included for generic cryptographic operations. It also features three high speed DSP blocks, including FFT, IDFT, and FIR. An SPI controller and a UART module are integrated with our model for external communication. A PMU (Power Management Unit) is included to facilitate scenarios involving power analysis. To study different system use case scenario of realistic industrial SoC designs, the IPs are segregated mainly into two clusters. The North cluster includes the high-speed IPs (*e.g.*, CPU and DSP accelerators) and the South cluster incorporates crypto engines, PMU module, and external communication IPs like UART and SPI.

2) `AutoSoC`*:* The `AutoSoC` model is inspired by commercial NoC-based automotive SoCs with separate application specific subsystems [16]. As with realistic implementations, it has a much larger number of IPs. The IPs are organized into a number of subsystems. The SoC incorporates four 32-bit RISC-V cores with variations in instruction set support. These cores are size optimized implementations of RISC-V processors. The crypto subsystem is augmented with RSA and MD5, DSP subsystem with DFT and IIR,

TABLE III
CLUSTERSOC (CLUSTER-BASED SOC): AREA AND POWER OVERHEAD OF SATELLITE UNITS IN REPRESENTATIVE IP CORES

Representative IP cores	CPU	FFT	FIR	IDFT	SRAM(DP)	AES(128)	DES3	SHA(256)	PMU	UART	SPI	Router
Area Overhead (%)	1.67	5.81	3.25	3.17	11.25	6.93	7.2	5.25	10.58	10.45	12.79	13.25
Power (Active+Leage) Overhead (%)	2.35	6.23	4.15	4.29	10.58	6.55	8.25	4.19	13.49	9.83	13.23	12.17

TABLE IV
AUTOSOC (REPRESENTATIVE AUTOMOTIVE SOC): AREA AND POWER OVERHEAD OF SATELLITE UNITS IN REPRESENTATIVE IP CORES

Representative IP cores	RV32I	RV32IC	RV32IM	RV32IMC	DFT	IIR	SRAM(SP)	DMA	AES(192)	RSA	MD5	ETHNT.	GPS
Area Overhead (%)	5.68	3.56	3.13	2.17	3.16	2.12	13.98	8.79	3.05	2.58	4.85	5.15	2.63
Power (Active+Leage) Overhead (%)	6.43	5.83	5.23	2.87	4.31	1.68	11.25	9.01	4.17	3.24	3.20	5.83	1.67

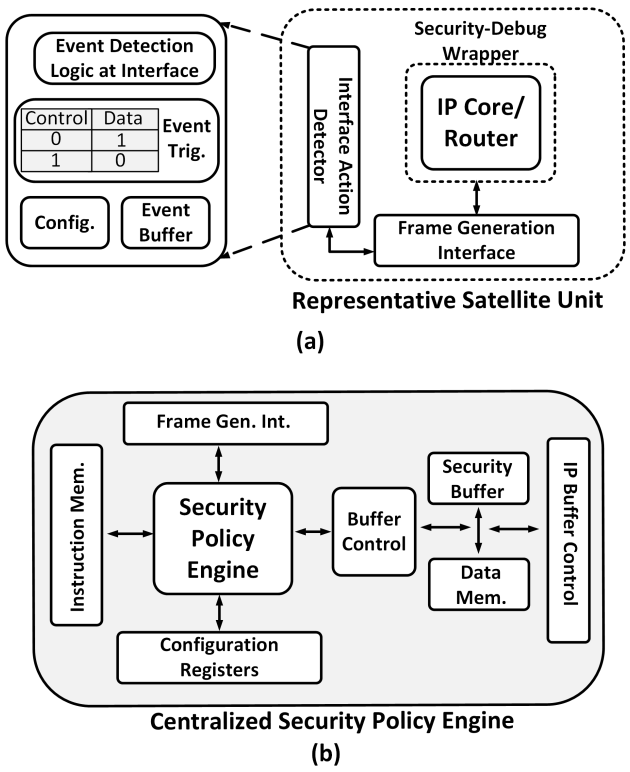

Fig. 9. A illustration of (a) Satellite units (SUs) deployed at each IP and NoC router. (b) Security Policy Engine (SPE) connected as an Infrastructure IP on the NoC fabric.

the memory subsystem with DMA controller and single port SRAM, and the connectivity subsystem with Ethernet and GPS IP.

Both SoC models incorporate a modular and configurable open-source Network-on-chip, named LiSNoC, as the interconnect fabric [17]. It supports wormhole-based flow control. The basic routing algorithm is dimension-ordered, deadlock-free XY-routing. The network interface permits transfer of one flit per cycle. The packet arbitration employs strict ordering through the router switch. The link multiplexing is performed by round-robin arbitration. The number of input ports, output ports, and allocation of virtual channels (VCs) to input-output ports are parameterizable. In our work, we employed routers with 5 input ports, 5 outputs ports, and allotted 2 VCs for each input and output port. Each VC has a fixed-length queue of 4 flits. The virtual channel flow control is performed in on-off manner with two control signals named valid and ready. The flit transfer occurs through a handshaking protocol implemented with the ready and valid signals of each VC. The transfer occurs when both valid and ready signals are high. The associated network adapters feature DMA engines and message passing functionalities. The IPs are wishbone-bus compatible. The network adapter facilitates their compatibility to NoC protocol. All the IPs are obtained from various open source repositories in Verilog and SystemVerilog RTL models. The SoC models are functionally validated in ModelSim. We augmented both SoC model with SPE units, implemented through a 32-bit 5-stage DLX micro-controller with dedicated instruction and data memory. Furthermore, each IP and router is augmented with a representative SU.

B. Area and Power Overhead Analysis

To obtain overhead values for area and power, we syntheized the SoC models in *Synopsis Design Compiler* using a LEDA standard cell library at TSMC 250nm technology node. The estimated area and power overhead results incurred by the proposed satellite units (SUs) at each IP module and router in both models are presented in Table III and Table IV. The percentile increase in area and power overhead is relatively small and varies over a range of 1.67 to 13.49 among the IPs. The low to modest overhead feature of SUs facilitates the implementation of arbitrary security policies tailored for each IP within the periphery of its design constraints including area and power. Note that the strategy of exploiting of standardized on-chip resources *e.g.*, test and debug wrappers, which are readily available in the IPs through existing design practices, helped maintain reasonable overheads. The comparatively higher area and power overhead due to the addition of SUs in the router observed in our experiments is primarily because the router designs themselves were simple. In an industrial SoC router the relative overhead will be significantly less. While there are SoCs with many routers, commercial SoCs for IoT and automotive platforms include NoC fabrics with a limited number of routers. Security requirements of these SoCs are complex because of heterogeneity of IPs and communications. Our work is targeted towards such SoCs. Table V shows the area and power overhead results incurred by the SoC

TABLE V

AREA AND POWER OVERHEAD OF IMPLEMENTED POLICIES FOR DIFFERENT USE CASE SCENARIOS

Use Case Scenarios		OpenRISC		RISCV32IMC		NoC Router	
		Die Area Overhead (%)	Power (Active+Leakage) Overhead (%)	Die Area Overhead (%)	Power (Active+Leakage) Overhead (%)	Die Area Overhead (%)	Power (Active+Leakage) Overhead (%)
Case I	Non-observability and non-misdirection	2.65	3.78	6.625	9.45	14.98	13.46
Case II	Masquerade prevention and message immutability	3.34	2.94	8.35	7.35	2.94	13.79
Case III	Flooding attack prevention	4.27	4.75	10.67	11.85	16.14	18.85

TABLE VI

DIE AREA OVERHEAD OF CENTRALIZED SPE (AREA: 3.3 × 10^6 μm^2; POWER: 13.85 mW) IN COMPARISON TO REALISTIC SoCs

SoC	Die Area (μm^2)	Overhead of SPE (%)
ClusterSoC	27.6×10^6	12.07
AutoSoC	68.9×10^6	4.85
Intel Atom Z2520	40.6×10^6	8.2
A6 (APL0598)	96.71×10^6	3.44
Qualcomm Snapdragon 800	118.3×10^6	2.81

security policies implemented for the use cases studied in this work. In comparison to the industrial SoCs, the die area overhead incurred by the proposed security policy engine is significantly low. Note that we compared our overhead against a baseline SoC without the framework to ensure that we do not underestimate overhead due to our infrastructure. In practice, since an SoC will have additional security instrumentation and communication overhead to implement protection requirements, the overhead would be higher than baseline consequently reducing the relative overhead. For commercial SoC designs, the overhead of proposed security architecture would be insignificant with respect to full SoC die area.

Remark 3: We briefly remark on our overhead calculation. We have calculated the percentage area overhead of the proposed security policy engine (SPE) with respect to the area of our own SoC models and the area of different commercial SoCs manufactured at similar process technology. Table VI shows estimated die area overhead of the SPE with respect to our SoC models and different commercial SoCs *i.e.*, Intel Atom Z2520, Apple A6 (APL0598), and QualComm Snapdragon 800, all manufactured at 32 nm node technology. Such relative overhead calculation helped us get an estimate of the area overhead incurred by the proposed SPE when implemented on commercial SoCs.

C. Performance Experiments and SPE Placement

Our SoC models are implemented at RTL level. It is impossible to use an RTL simulator on SoC designs to measure performance and congestion workloads.[4] Furthermore, our SoC models do not include sufficient instrumentation to provide adequate controllability of internal events, making it difficult to deterministically exercise specific traffic patterns necessary to study latency and congestion. To study performance overhead, we instead used a state-of-the-art configurable NoC simulator that provides flexibility in studying the network behavior in the context of varying traffic patterns and SoC operating conditions. Analyzing traffic patterns is a common activity in microarchitectural exploration of SoC designs, to determine variety of parameters. Our work shows how to reuse such analysis to determine optimal placement position for SPE. For this experiment we used the `ClusterSoC` model: it is ideal for this experiment since it includes most salient features of industrial SoC implementations while still being relatively simple (compared to `AutoSoC`), facilitating better comprehension and control of traffic patterns for different system-level use cases.

A realistic study of performance overhead due to NoC traffic must account for several issues. First, the use of the architecture may incur some overhead even in situations when there is no malicious activity, *e.g.*, communication among SPE and satellite units *to determine* if a packet indicates evidence of masquerade or misdirection could add to the NoC overhead even if the packet turns out to be benign. We call this overhead the *benign overhead*. Furthermore, additional communication overhead may be incurred in case there is actual malicious activity. We refer to this overhead as *malicious overhead*. A good security architecture should incur minimal benign overhead and tolerable malicious overhead.

One key challenge in our performance experiments is to develop a methodology to realistically estimate benign and malicious overheads. Ideally, we ought to compare the communication induced by our architecture in benign (resp., malicious) with communication induced in a corresponding SoC design that does not implement this specific security architecture but possibly with ad hoc security policy implementations. To simulate the "SoC design without our architecture", we illustrate a collection of real application induced scenarios that represent the normal traffic pattern of the SoC. The communication established through these scenarios involve interactions among the CPU, memory modules, hardware accelerators, and other peripherals to realize system functionality. We refer to these representative SoC functional traffic as *regular activity*. Our experiments are designed to compare the benign and malicious overhead with the regular activity. Note that in regular activity we do not consider any overhead because of security instrumentation. Obviously, a practical SoC design would include *some* security techniques resulting in additional communication and performance bottleneck; hence, the relative performance overhead of the proposed security architecture in that case will be even less than the results in our experiments. Consequently, given

[4]Realistic workloads on an SoC typically entail several hours of execution on real silicon. On the other hand, one second of silicon execution roughly requires 30 years to fully simulate on an RTL model [18], [19].

that our experiments suggest that the benign and malicious overhead are not significant (cf. Fig. 10) when compared with regular activity without security instrumentation, the same conclusion would be transferred to other SoC designs that do employ additional security techniques.

1) Regular Activity Details: We implemented the following scenarios as the regular activity:

- SPI stores data collected from an external source (*e.g.*, sensors) in the untrusted memory (SRAM0) via DMA access. CPU reads the data, processes it, and stores the processed data in trusted memory (SRAM1). UART requests access to read the processed data and stores it in an external storage device connected to it. CPU sends data and key to crypto-cores (AES) for encryption. Encrypted data is then transferred to the UART.
- Encrypted data from an external storage device connected via UART is read into the untrusted memory (SRAM0) via DMA access. CPU sends the data and the key to SHA for decryption. CPU forwards the decrypted data to DSP core (FFT) for processing and DSP sends it back to CPU. CPU stores the data in trusted memory (SRAM1).
- CPU sends a control packet to Power Management Unit (PMU) to activate sleep state. PMU sends sleep signals to all other High-speed IPs and sets the SoC operating mode to low-power. External device connects through UART to read processed data. UART triggers the PMU requesting to switch mode from low-power to high-power. PMU wakes all the High-speed IPs. CPU handles the UART read request and sends the requested data and key to AES for encryption. AES sends the encrypted data to UART.
- UART sends data to (SRAM0) via DMA. The data is stored in (SRAM1) after encryption by SHA. Sleep state in activated by CPU in co-ordination with PMU. Normal operating state is again retrieved by PMU in response to an interrupt from UART. Encrypted data stored in (SRAM1) is sent back to UART.

2) Experimental Setup and Performance Results: We adopted a NoC simulator to create SoC traffic patterns induced by the use cases. The traffic flow is simulated with open-source Garnet 2.0 interconnection network model inside Gem5 simulator [20], [21]. The NoC model is generated by running Garnet 2.0 in standalone mode. We chose a simulation period of 10K cycles. The scenarios described above as *regular activity* are instantiated at specific instances over this period, resulting in a total of 64 instantiated transactions. The initiation times intervals for the different transactions were chosen carefully to ensure a realistic overlap: the purpose of studying such overlapping and concurrent traffic patterns is to analyze the system latency under realistic usage scenarios. Fig. 10 shows the results. All overheads are measured with respect to the regular activity. For *Benign Overhead*, we additionally include (on top of regular activity) message traffic resulting from run-time monitoring of IP-level activity and successive message passing to maintain overall security status. This includes the SUs of each IP and routers monitoring various security critical events as specified by the policies and periodically communicating those with the SPE to update the IP and system level security status. For *Malicious Overhead* we randomly perturbed 6% of the transactions to be "malicious". The percentage of malicious transactions was chosen to be 6% mimicking the typical numbers used in microarchitectural explorations for determining optimal microarchitectural parameters, as a basis of realistic traffic patterns. Traffic analysis due to such activity is performed by simulating the attack scenarios illustrated in VI. The key steps of the message flow include detection and prevention of rogue IP and router activity by distributed SUs, the event logs being sent to the centralized SPE, and security messages being broadcast across the system. We interpreted the observed overhead in terms of following parameters: (i) *Packet Network Latency*, (ii) *Packet Queueing Latency*, and (iii) *Link Utilization*. The communication between the SUs and the SPE incurs a communication overhead of approximately 5% in benign activity where our proposed framework monitors system security and no attack is detected. During malicious activity (*i.e.*, ongoing attack scenarios), the implementation of the illustrated security policies incur an additional 0.5% communication overhead for the detection and mitigation of the attack. The minimal communication overhead of proposed framework validates the feasibility of encrypted communication between SUs and SPE through the on-chip network.

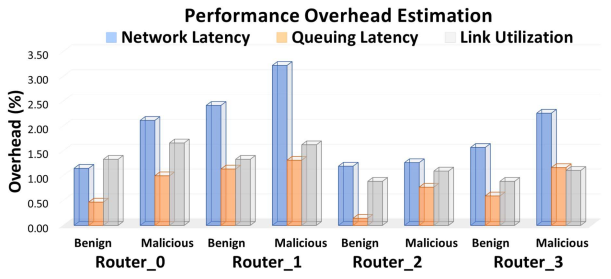

Fig. 10. Overhead estimation of the latency and link utilization for various positions of SPE in NoC during benign & malicious operating conditions.

3) Placement of SPE: Clearly, the placement of SPE affects the traffic pattern; in fact, one critical reason for studying traffic patterns is to identify the optimum location for SPE that incurs minimal latency. We connected SPE to each router of the topology and ran the simulation each time to analyze the resultant effect on the network performance, as shown in Fig. 10. The results indicate that, for our specific tree topology, connecting SPE to Router 2 results in an optimal performance and incurs the least overhead, while positioning the SPE to Router 1 results in the highest performance overhead. The key observation for our exercised use cases is that placing the SPE near to majority of the peripheral IP blocks reduces performance overhead due to additional security message traffic.

VIII. Comparison With Related Work

A. Untrusted Third-Party IPs and NoCs

The state-of-the-art research on trust verification of 3PIPs involves analysis, detection, and deterrence of malicious circuitry insertion aka hardware Trojans in untrusted IP cores [22]. Research on hardware Trojan design have been

TABLE VII
CONTEMPORARY RESEARCH ON UNTRUSTED IPs AND NoCs VS. SCOPE OF OUR WORK

Existing Research on Untrusted IPs and Compromised NoCs	**Scope of the Proposed Work**
Detection and protection mechanisms for IP level hardware Trojan mostly through static IP analysis	Run-time detection and mitigation of system level malicious activity propagating from untrusted IPs through inter-IP communication
Existing approaches of run-time detection have limited scope; hence, do not offer adequate coverage and mitigation strategies	Diverse IP and system level SoC security policies evaluate security status of the SoC and assert appropriate controls on the fly
Current solutions for compromised NoCs are ad-hoc to specific type of attacks; thus, fail to protect the SoC platform from the entire spectrum of attacks violating key security requirements of system-level communication	An architecture level solution for flexible implementation of SoC security policies that protect against vulnerabilities and exploitations by 3PIPs during inter-IP communication of NoC-based SoC designs

conducted to explore novel trigger conditions and payload delivery mechanisms and analyze the difficulty of Trojan activation and detection [23]. Trojan design optimization has been studied to minimize the side-channel impacts due to trigger and payloads [24]. The major portion of Trojan research focuses on countermeasures [25]–[27]. Post-silicon detection techniques have been extensively studied by researchers [28]. Functional testing for triggering rare nets is proposed by Chakraborty *et al.* [29]. Side channel signal analysis has been performed by researchers to detect Trojans [30]. Pre-silicon Trojan detection technique and vulnerability analysis have also been extensively performed through behavioral and structural code analysis and formal methods [8], [31]. Runtime monitoring in critical computation can significantly increase the trust-level against various attacks. These approaches can exploit existing resources or on-chip structures to detect suspicious behavior and operating conditions like security critical events and transient power to detect Trojan activity [32]. Jin and Sullivan [33] proposed an analog on-chip nueral network that is capable of detecting suspicios activity based on sensor measurements.

While the research on untrusted IPs applies to NoC IPs to varying extents, researchers have studied the untrusted NoCs threats over the past years. The threat of a Trojan inserted NoC has been studied by Ancajas *et al.* [34]. The authors consider a threat model comprised of compromised NoC and an accomplice piece of software. It combines packet certification, node obfuscation, and data scrambling to prevent information leakage through the compromised NoC. Boraten and Kodi [35] employed a packet sensitization technique that incorporates packet certification. The vulnerability of these works due to weakness of security primitives, however, is demonstrated by a later study [36]. Sepulveda *et al.* [37] addressed these limitations by developing a tunnel-based network interface.

B. Reliable and Secure Architectures

There has been significant research on secure NoC architectures for NoC-based MPSoCs. Secure NoCs have been developed to prevent illegal message mutation by attackers. Adoption of channel separation is proposed in literature to prevent data modification [38]. Florin *et al.* [39] introduced the deployment of data protection units for secure memory accesses. Insertion of firewalls to application specific NoCs is studied to deter denial of service attacks [40]. Kostrzewa *et al.* [41] designed a dynamic control layer combining global and local arbitration that supports real-time mixed critical systems. Non-interfering architecture is developed to ensure reliability of information flow [42]. Traffic analysis schemes with distributed management is also designed to develop architectures resilient to timing attacks like prime+probe [43]. Priority-based arbitration is employed by Wang and Suh [44] for protection againts timing information leakage through timing channels. Efficacy of hierarchical group-wise security protocols is demonstrated for dynamic security requirements of MPSoCs [45]. An energy efficient design for run-time detection of malicious activity in untrusted NoC is studied by Hussain *et al.* [46]. Bokhari *et al.* [47] proposed a multimode interconnect architecture that can be configured to run in different operating modes based on different operational scenarios.

While there has been an abundance of work on untrusted IP issues and security enhanced NoC architectures, we take a holistic approach to address the problem at system-level communication. Our goal in this work is to ensure system reliability and trustworthiness despite the presence of malicious hardware and colluding firmware and software attempting to subvert system functionality. Consequently, we focus on the system level impacts of malicious hardware, firmware, and software to achieve architectural resiliency. Similarly, we focus on the paylaods of system level Trojans and malicious software and firmware compared to existing works of IP level threats. Moreover, we address the limitations of ad-hoc solutions of SoC and NoC enhancement for specific threats by developing a comprehensive framework that helps the developer integrate a variety of security features to the SoC design based on requirements and the use case scenarios. The flexibility of policy based approach allows the designer optimize the system based on performance and security trade-offs while adopting a generic architectural solution for both untrusted IPs and rogue NoC fabric. We demonstrate that the proposed architecture can be employed efficiently to obtain resiliency against all major system level communication attacks.

Remark 4: To our knowledge, our work represents the first SoC resiliency architecture targeted towards the broad spectrum of inter-IP communication attacks presented in our threat model. Note that our approach is fundamentally different from ad hoc approaches used in current practice to implement security policies. Moreover, most of the existing works focus on only one attack model and present results based on that attack scenario whereas our architecture is designed and tailored to implement security policy-based solution for the entire spectrum of inter-IP communication attacks in NoC-based SoCs. Consequently, there is no relevant related work against which to compare performance or overhead of our architecture. Instead, we have performed extensive quantitative evaluation of our architecture on different SoC models in terms of area, power, and performance. To be stringent in our evaluations, we compared our security architecture integrated SoCs with baseline SoC models that do not incorporate any security architecture at all; whereas in practice, there would be some security scheme in those baseline models reducing the comparative overhead. We showed that the incurred overhead of our architecture addressing all major communication attacks is minimal (less than 15% approx.).

IX. Conclusion

We have presented, for the first time to our knowledge, a resilient SoC security architecture to build a trusted system involving NoC communication among untrusted IPs. We have shown how to systematically implement key security requirements of inter-IP communication and proposed an architecture-level solution for run-time detection and mitigation of on-chip communication threats. With the horizontal shift in semiconductor industry, there has been an increased reliance on untrusted 3PIPs and NoCs. Existing works on design-time IP-trust verification fail to guarantee trusted SoC operation with the increasing modalities of communication attacks. Our work presents a fundamentally different approach, *i.e.*, an architecture for run-time system-level resilience. We have shown the efficacy of the approach by implementing policy-based solutions for all major use case scenarios of secure system-level communication. Our experimental results on realistic SoC designs suggests that our approach is viable and scalable in terms of area, power, and performance overheads.

In future work, we will evaluate the framework on industrial SoC designs and investigate mechanisms for automating the synthesis of the fine-grained security policies for different communication attacks. Furthermore, we will augment the architecture with a CAD framework to determine and evaluate optimal security policies for effective system-level protection based on deployment requirements.

References

[1] M. Hicks, C. Sturton, S. T. King, and J. M. Smith, "SPECS: A lightweight runtime mechanism for protecting software from security-critical processor bugs," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 517–529, Mar. 2015.

[2] mseaborn@chromium.org. (2012). *X86–64: Data16 Prefix on Direct Jumps Allows Sandbox Escape on AMD Cpus. Nativeclient Bug Tracker*. [Online]. Available: https://bugs.chromium.org/p/nativeclient/issues/detail?id=2578

[3] *Intel Xeon Processor E7-8800/4800/2800 Product Families: Specification Update*, Intel Corp., Santa Clara, CA, USA, 2015.

[4] S. Ray, E. Peeters, M. M. Tehranipoor, and S. Bhunia, "System-on-chip platform security assurance: Architecture and validation," *Proc. IEEE*, vol. 106, no. 1, pp. 21–37, Jan. 2018.

[5] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.

[6] N. E. Jerger, T. Krishna, and L.-S. Peh, "On-chip networks," *Synth. Lectures Comput. Archit.*, vol. 12, no. 3, pp. 1–210, 2017.

[7] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 697–708.

[8] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, p. 112.

[9] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Nov. 2009, pp. 166–171.

[10] *Intel Baytrail Products*. Accessed: May 10, 2019. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/codename/55844/bay-trail.html

[11] A. Basak, S. Bhunia, and S. Ray, "A flexible architecture for systematic implementation of SoC security policies," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 536–543.

[12] A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security assurance for System-on-Chip designs with untrusted IPs," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1515–1528, Jul. 2017.

[13] K. Shuler. (2013). *Majority of Leading China Semiconductor Companies Rely on Arteris Network-on-Chip Interconnect IP*. [Online]. Available: http://www.arteris.com/press-releases/china_majority_arteris_pr_19_august_2013

[14] J. Probell and B. de Lescure, "SoC reliability features in the FlexNoC resilience package: A complementary IP packagefor use with arteris FlexNoC IP," Arteris, Inc., Campbell, CA, USA, Tech. Rep., 2014.

[15] A. P. Deb Nath, S. Ray, A. Basak, and S. Bhunia, "System-on-chip security architecture and CAD framework for hardware patch," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 733–738.

[16] K. Shuler. (2018). *How SoC Interconnect Enables Flexible Architecture for ADAS and Autonomous Car Designs*. [Online]. Available: http://www.arteris.com/blog/semiconductor-engineering-autonomous-driving-iso-26262-compliant

[17] S. Wallentowitz, A. Lankes, A. Zaib, T. Wild, and A. Herkersdorf, "A framework for open tiled manycore system-on-chip," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 535–538.

[18] P. Mishra, R. Morad, A. Ziv, and S. Ray, "Post-silicon validation in the SoC era: A tutorial introduction," *IEEE Des. Test. Comput.*, vol. 34, no. 3, pp. 68–92, Jun. 2017.

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

[20] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 33–42.

[21] N. Binkert *et al.*, "The Gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[22] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1–23, May 2016.

[23] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 5s, pp. 1–20, Oct. 2014.

[24] B. Cha and S. K. Gupta, "A resizing method to minimize effects of hardware trojans," in *Proc. IEEE 23rd Asian Test Symp.*, Nov. 2014, pp. 192–199.

[25] N. A. Hazari and M. Niamat, "Enhancing FPGA security through trojan resilient IP creation," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jun. 2017, pp. 362–365.

[26] M. T. Rahman *et al.*, "Defense-in-depth: A recipe for logic locking to prevail," *Integration*, Jan. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926019303694, doi: 10.1016/j.vlsi.2019.12.007.

[27] N. A. Hazari, F. Alsulami, and M. Niamat, "FPGA IP obfuscation using ring oscillator physical unclonable function," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jul. 2018, pp. 105–108.

[28] M. T. Rahman *et al.*, "Physical inspection & attacks: New frontier in hardware security," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 93–102.

[29] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware trojan detection," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2009, pp. 396–410.

[30] D. Forte, C. Bao, and A. Srivastava, "Temperature tracking: An innovative run-time approach for hardware trojan detection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 532–539.

[31] A. P. Deb Nath, S. Bhunia, and S. Ray, "ArtiFact: Architecture and CAD flow for efficient formal verification of SoC security policies," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 411–416.
[32] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia, "TeSR: A robust temporal self-referencing approach for hardware trojan detection," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust*, Jun. 2011, pp. 71–74.
[33] Y. Jin and D. Sullivan, "Real-time trust evaluation in integrated circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2014, pp. 1–6.
[34] D. M. Ancajas, K. Chakraborty, and S. Roy, "Fort-NoCs: Mitigating the threat of a compromised NoC," in *Proc. The 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.
[35] T. Boraten and A. Karanth Kodi, "Packet security with path sensitization for NoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2016, pp. 1136–1139.
[36] C. Reinbrecht, A. Susin, L. Bossuet, and J. Sepúlveda, "Gossip NoC—Avoiding timing side-channel attacks through traffic management," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 601–606.
[37] J. Sepúlveda, A. Zankl, D. Flórez, and G. Sigl, "Towards protected MPSoC communication for information protection against a malicious NoC," in *Proc. Int. Conf. Comput. Sci. (ICCS)*, Zürich, Switzerland, Jun. 2017.
[38] J.-P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "NOC-centric security of reconfigurable SoC," in *Proc. 1st Int. Symp. Networks-Chip (NOCS)*, May 2007, pp. 223–232.
[39] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure memory accesses on Networks-on-Chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1216–1229, Sep. 2008.
[40] Y. Hu, D. Muller-Gritschneder, M. J. Sepulveda, G. Gogniat, and U. Schlichtmann, "Automatic ILP-based firewall insertion for secure application-specific Networks-on-Chip," in *Proc. 9th Int. Workshop Interconnection Netw. Archit., Chip, Multi-Chip*, Jan. 2015, pp. 9–12.
[41] A. Kostrzewa, S. Saidi, and R. Ernst, "Dynamic control for mixed-critical Networks-on-Chip," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 317–326.
[42] H. M. G. Wassel *et al.*, "SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 583–594, 2013.
[43] C. Reinbrecht, A. Susin, L. Bossuet, G. Sigl, and J. Sepúlveda, "Timing attack on NoC-based systems: Prime+Probe attack and NoC-based protection," *Microprocessors Microsyst.*, vol. 52, pp. 556–565, Jul. 2017.
[44] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *Proc. IEEE/ACM 6th Int. Symp. Networks-Chip*, May 2012, pp. 142–151.
[45] J. Sepulveda, D. Flórez, V. Immler, G. Gogniat, and G. Sigl, "Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs," *Microprocessors Microsyst.*, vol. 50, pp. 164–174, May 2017.
[46] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: An energy efficient design for runtime hardware trojan detection in untrusted network-on-chip," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 345–350.
[47] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, "SuperNet: Multimode interconnect architecture for manycore chips," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, p. 85.

Atul Prasad Deb Nath (Student Member, IEEE) received the B.Sc. degree from the Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh, in 2011, and the M.Sc. degree from the University of Toledo, Toledo, OH, USA, in 2016. He is currently pursuing the Ph.D. degree with the University of Florida, Gainesville, FL, USA. His Ph.D. research focuses on investigating major aspects of SoC security and developing novel architectural features to protect assets, firmware, and on-chip communication against various adversarial models. He has published four book chapters, 11 peer-reviewed journals, and premier conference papers, and filed one patent. His research interests include system-on-chip (SoC) platform security and CAD for security and trust.

Srivalli Boddupalli (Student Member, IEEE) received the B.Tech. degree from the Chaitanya Bharathi Institute of Technology, Hyderabad, India, in 2016, and the M.S. degree from the University of Florida in 2018, where she is currently pursuing the Ph.D. degree with the Department of ECE. She has contributed to research projects on SoC security, and she is currently working on developing machine learning-based security primitives for connected vehicle applications. Her research interests are in the fields of trustworthy computing and automotive security, with a primary focus on autonomous vehicle technology.

Swarup Bhunia (Senior Member, IEEE) received the B.E. degree (Hons.) from Jadavpur University, Kolkata, India, in 1995, the M.Tech. degree from IIT Kharagpur, Kharagpur, India, in 1997, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

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 is currently the Director of the Warren B. Nelms Institute for the Connected World and a Semmoto Endowed Chair Professor of IoT with the University of Florida, Gainesville, FL, USA. He has over 250 publications in peer-reviewed journals and premier conferences. His current research interests include hardware security and trust, adaptive nanocomputing, and novel test methodologies. He received the IBM Faculty Award, the NSF CAREER Award, the SRC Inventor Recognition Award, the SRC Technical Excellence Award, and several best paper awards/nominations. He has been serving as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, the *ACM Journal of Emerging Technologies*, and the *Journal of Low Power Electronics*.

Sandip Ray (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, where he holds an Endowed IoT Term Professorship. Before joining University of Florida, he was a Senior Principal Engineer at NXP Semiconductors, and prior to that, he was a Research Scientist with Intel Strategic CAD Laboratories. During his industry tenure, he led industrial research and R&D in pre-silicon and post-silicon validation of security and functional correctness of SoC designs, design-for-security and design-for-debug architectures, and security validation for automotive and the Internet-of-Things applications. His current research targets correct, dependable, secure, and trustworthy computing through cooperation of specification, synthesis, architecture, and validation technologies. He is the author of three books and over 100 publications in international journals and conferences. He has also served as a Technical Program Committee Member of over 50 international conferences, as the Program Chair of ACL2 2009, FMCAD 2013, and IFIP IoT 2019, as a Guest Editor for IEEE DESIGN & TEST, IEEE TMSCS, and *ACM TODAES*, and as an Associate Editor of Springer HaSS and IEEE TMSCS.