A Flexible Architecture for Systematic Implementation of SoC Security Policies

Abhishek Basak[1], Swarup Bhunia[1], and Sandip Ray[2]
[1]Dept. of EECS, Case Western Reserve Univ., USA, [2]Strategic CAD Labs, Intel Corporation, USA
axb594@case.edu, skb21@case.edu, sandip.ray@intel.com

***Abstract*— Modern SoC designs incorporate several security policies to protect sensitive assets from unauthorized access. The policies affect multiple design blocks, and may involve subtle interactions between hardware, firmware, and software. This makes it difficult for SoC designers to implement these policies, and system validators to ensure adherence. Associated problems include complexity in upgrading these policies, IP reuse for systems targeted for markets with differing security requirement, and consequent increase in design time and time-to-market. In this paper, we address this important problem by developing a generic, flexible architectural framework for implementing arbitrary security policies in SoC designs. Our architecture has several distinctive features: (1) it relies on a dedicated, centralized, firmware-upgradable plug-and-play IP block that can implement diverse security policies; (2) it interfaces with individual IP blocks through their "security wrapper", which exploits and extends test/debug wrappers; (3) it implements a security policy as firmware code following existing security policy languages; (4) it can implement any security policy as long as relevant observable and controllable signals from the constituent IPs are accessible through the security wrappers; and (5) it realizes a low-overhead communication link between security wrappers of IP blocks and the centralized, dedicated controller. The approach builds on and extends the recent work on developing a centralized infrastructure IP for SoC security, referred to as IIPS, that interface with IP blocks using their boundary scan based wrappers. While this architecture is generic and independent of security policy types, we provide case studies with several common policies to show the flexibility and extendibility of the architecture. We also evaluate its viability in terms of overhead in area and power.**

I. Introduction

Recent years have seen rapid proliferation of embedded and mobile computing devices. Such devices come in a variety of form factors, including smartphones, tablets, automotive controls, wearables, medical and fashionable implants, and smart sensors. Given their diversity and personalization, security has emerged as a critical concern for them. Most of these devices contain confidential assets, which must be protected against unauthorized access. Examples of secure or sensitive assets present in virtually all modern computing systems include cryptographic and DRM keys, premium content, firmware, programmable fuses, and personal end-user information. Unauthorized or malicious access to these assets can result in leakage of company trade secrets for device manufacturers or content providers, identity theft for end users, and even destruction of human life. Consequently, it is vital to ensure that secure assets in computing devices are adequately protected.

Most embedded and mobile computing devices are architected around one or more System-on-Chip (SoC) designs. An SoC architecture involves coordination and communication of a number of pre-designed hardware blocks of well-defined functionality (referred to as "intellectual properties" or "IPs"). Security assets in SoC designs spread across different IPs, and access restrictions to these assets are defined by highly subtle, complex, and sometimes ambiguous *security policies* [1], [2], [3], [4]. These policies are defined by system architects as well as different IP design and SoC integration teams and often refined or modified during system development. This makes it highly challenging to validate a system against the security policies, develop architectures to provide built-in resilience against unauthorized access, or update system-level security requirements *e.g.*, in response to changing customer needs. To exacerbate the issue, security policies are rarely specified in any formal, analyzable form. Some policies are described (in natural language) in different architecture documents, and many remain undocumented.

In this paper, we propose a generic architecture for systematic implementation of diverse system-level security policies for modern SoC designs. The cornerstone of our architecture is a dedicated, plug-and-play, centralized IP block, referred to as *E-IIPS* (Extended Infrastructure IP for Security). It is a microcontroller-based firmware-upgradable module that realizes system-level security policies of various forms and types using firmware code following existing security policy languages, such as SAPPER [2]. The E-IIPS module interfaces with the constituent IP blocks in a SoC using "security wrappers" integrated with the IPs. These security wrappers extend the existing test (e.g. IEEE 1500 boundary scan based wrapper) and debug wrapper (e.g. ARM's coresight IP interface) of an IP. These security wrappers detect local events relevant to the implemented policies and enable communication with the centralized E-IIPS module. The architecture can be implemented on SoC designs incurring modest hardware overhead. Each security policy to be implemented is programmed into the E-IIPS module as firmware code, which realizes communication of E-IIPS with the wrapper interface of IPs. The E-IIPS module intervenes when a policy violation is detected. The wrapper and the E-IIPS architectures are flexible and agnostic to the SoC design functionality or security policy requirements. They can be applied in a scalable manner to existing SoC designs with varying number of IP blocks. We demonstrate how to use the proposed architecture

978-1-4673-8388-2/15/$31.00 ©2015 IEEE

to facilitate implementation and validation of system-level SoC security policies through several case studies involving common security policies of various types.

E-IIPS builds on our recently reported infrastructure IP for SoC security, IIPS [5]. IIPS alleviates SoC designers from separately addressing security issues through design modifications in multiple cores, and provides ease of integration and functional scalability. However, it was limited to protection against low-level hardware security vulnerabilities, *e.g.*IP piracy, hardware Trojan, side-channel analysis, and scan-based information leakage. E-IIPS extends IIPS for implementing security policies in SoC integration.

The paper makes three important contributions. We develop, for the first time to our knowledge, an on-chip flexible architecture using a configurable centralized controller IP for implementing, exploring, and analyzing diverse SoC security policies. Second, we present a general interface of security policy enforcement with functional IPs, that extends the existing test/debug wrappers and makes use of existing communication fabrics in SoC designs. Finally, we present simulation results on validating the module, estimating its hardware overhead, and demonstrating its capabilities in achieving protection of secure resources within SoC.

The remainder of the paper is organized as follows. Section II provides the relevant background on security policies in SoC designs. In Section III, we discuss our policy enforcement architecture, and explain its proposed usage in SoC design execution. We present some illustrative case scenarios in Section IV. In Section V, we present simulation results to estimate overheads introduced by the architecture, in terms of area and power. Section VI discusses related work, and Section VII concludes the paper.

II. SoC Security Policies

Modern SoC designs include a large number of critical assets, which must be protected against unauthorized access. At a high level, such access control can be defined by confidentiality, integrity, and availability requirements [6]. Security policies map such requirements to "actionable" design constraints that can be used by IP implementors or SoC integrators to define, analyze and implement protection mechanisms. Following are two representative examples for a typical SoC.

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

Example 1 is a confidentiality requirement while Example 2 is an integrity constraint; however, the policies provide definitions of (computable) conditions to be satisfied by the design for accessing a security asset. Furthermore, access to an asset may vary depending on the state of execution (*e.g.*, boot time, normal execution, etc.), or position in the development lifecycle (*e.g.*, manufacturing, production, etc.).

Unfortunately, security policies in a modern SoC design are significantly complex, and developed in an ad hoc manner

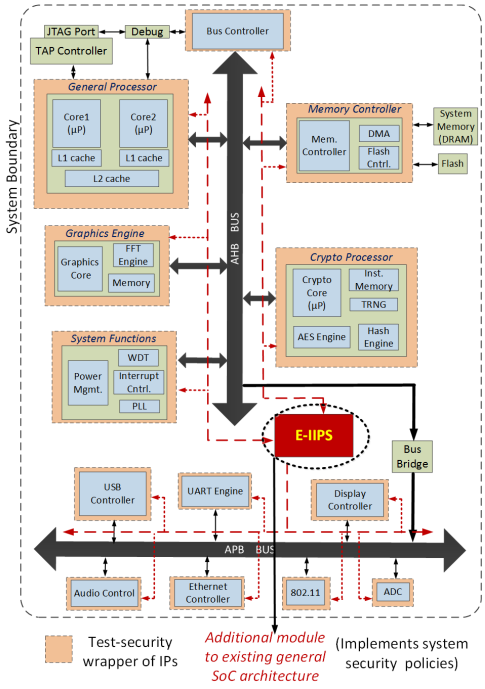

Fig. 1. Schematic of Proposed Architecture in a representative SoC.

based on customer requirements and product needs. Below we summarize some policy classes. It is beyond the scope of this paper to provide a comprehensive compendium of different policies, or even to discuss any of them in detail. The description below merely provides a flavor of some existing policies, and the interested reader is encouraged to refer to the cited bibliography for additional detail.

Access Control [7], [8], [9]: This is the most common class of policies, and specifies how different agents in a SoC can access an asset at different points of the execution. Here an "agent" can be a hardware or software component in any IP. Examples 1 and 2 above are examples of such policy. Furthermore, access control forms the basis of many other policies, including information flow, integrity, and secure boot.

Information Flow [10], [11]: Values of secure assets can sometimes be inferred without direct access, through indirect observation or "snooping" of intermediate computation or communications of IPs. Information flow policies restrict such indirect inference. Following is an example:

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

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

Liveness [12]: These policies ensure that the system performs its functionality without "stagnation" throughout its execution. A typical liveness policy is that a request for a resource by an IP is followed by an event response or grant. Deviation from such a policy can result in system deadlock or livelock, consequently compromising system availability requirements.

Time-of-Check vs. Time of Use (TOCTOU) [13], [3]: This refers to the requirement that any agent accessing a resource requiring authorization is indeed the agent that has been authorized. A critical example of TOCTOU is in firmware update, where the policy requires that firmware eventually installed on an update is the same one that has been authenticated.

Observe that the above policies relate to *integration* characteristics of SoC designs, not individual IPs. For this paper, we assume that the IPs themselves are trustworthy, *i.e.*, the underlying threat model includes external attacks through software or SoC interface but not malicious hardware introduced in the IPs. Our threat model is reasonable for SoC designs involving primarily in-house rather than third-party IPs or in cases where (orthogonal) IP trust verification has been accomplished to rule out malicious backdoors or Trojans.

III. Architecture

Fig. 1 illustrates our proposed architecture. It includes two main components: (1) a centralized security policy controller IP (referred to as E-IIPS or extended IIPS in the rest of the paper), and (2) security wrappers around individual IPs to facilitate communication with E-IIPS. To facilitate configurability across different products and use cases, E-IIPS is defined as a microcontrolled soft IP. SoC designers can program security policies in E-IIPS as firmware modules that are then stored in a secure ROM or flash memory; secure policy update is supported through an authenticated firmware update mechanism. E-IIPS communicates with other IPs via corresponding security wrappers as follows. For enforcing different security policies, E-IIPS may need different local IP-specific collaterals. For instance, suppose a policy prohibits access of internal registers of IP `A` by IP `B` when `A` is in the middle of a specific security-critical computation. To enforce the policy, E-IIPS must "know" when `B` attempts to access the local registers of `A` as well as the security state of the computation being performed by `A`. The security wrappers provide a standardized way for E-IIPS to obtain such collateral while abstracting the details of internal implementation of individual IPs. In particular, the wrappers implement a protocol to communicate with E-IIPS during the execution. Based on the policies implemented, E-IIPS can configure the wrapper of an IP at boot time to provide internal event information under specific conditions (*e.g.*, security status of internal computation, read requests to specific IPs, etc.); the security wrappers monitor for configured conditions and provide requested notification to E-IIPS. IP development teams are responsible for augmenting individual IPs with security wrappers, by extracting security-critical information (see below).

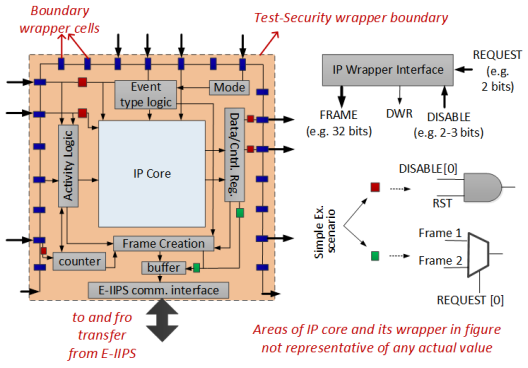

Fig. 2. Architecture of a generic IP security wrapper.

Design Choices. A key design choice for E-IIPS is its centralized firmware-upgradable architecture, *i.e.*, it is implemented as a single re-usable IP block in the SoC. This choice is governed by the need to provide a single place for understanding, exploration, upgrade, and validation of system-level security policies. Indeed, the current complexity in security policy analysis and modification is precisely that the policies are "sprinkled" across the different IPs in the SoC. Our centralized architecture is specifically intended to alleviate this complexity. On the other hand, this choice implies that communication wih E-IIPS is a bottleneck for system performance. We address this issue by making the security wrappers "smart" so that only security-relevant information is communicated to E-IIPS, possibly under the latter's directive. Finally, the choice of a microcontrolled rather than hardware implementation stems from the need to update security policies in-field, either due to customer requirements or in response to a known exploit or design bug. On the other hand, this makes E-IIPS itself vulnerable to attacks through firmware updates. In Section III-D we discuss authentication mechanisms to address this issue.

A. IP Security Wrappers

Security wrappers extract security-critical events from the operating states of the underlying IP for communication with E-IIPS. Note that the naive approach of simply extracting all data, control, and status signals from IPs to E-IIPS would incur prohibitive communication and routing overhead. To address this problem, we develop security wrappers on IPs that incorporate "smartness" to detect security-critical events of interest in the IP while providing both a standard communication interface between the IP and the E-IIPS and a standard template-based design that can be easily integrated on top of the IP implementation.

How can the wrapper identify security-critical events while still providing a standardized template-based design? The key observation is that IPs in a SoC can be divided into a small collection of broad categories and security-critical events. Table I shows some of the broad IP categories together with some of the security-critical information relevant to each. For instance, "Memory IPs" include all IPs controlling the access to different memory hierarchies, *e.g.*, memory

TABLE I

REPRESENTATIVE SET OF SECURITY CRITICAL EVENTS ACCORDING TO IP TYPE

Type of IP	*Example IPs*	*Type of Events*	*Associated Metadata*
Memory IP	Memory/cache controller, DMA engine	read/write request to specific address, DMA access, execution mode	page size, burst size(DMA) ECC type, low power clock rates
Processor Core	CPU, GPU, ethernet controller audio and video card	start/end of critical system threads, system interrupt, firmware upgrade request	stored operation logs, flag or register settings, process duration
Communication Core	Bus Controller, Bridge, Router, USB controller, PCIExpress	data transfer request, source/destination address peripheral IP transfer request, idle modes	transfer packet size, serial frequency arbiter priority, bus clk. rate
Hard logic Custom IP	AES, SHA engines, FFT, DWT block	firmware integrity check start/end, secure key access, FFT request by video card	duration of operation, local clk domain

controllers, Direct Memory Access (DMA) modules, cache controllers, flash control logic etc., and processor cores include general purpose CPUs, GPUs, as well as cores controlled by microcode/firmware *e.g.*, ethernet, UART controller, audio, video cards etc. The security-critical events of interest also standardize substantially within each IP category. For instance, events in a Memory IP include read and write requests to specific address ranges by particular IPs (including DMA), as well as functional/standby modes. On the other hand, events in a processor core include start and end of critical system or application processes and threads, computations generating exceptions, interrupts by system controllers (often utilized by adversaries to jump to critical system addresses), etc. Finally, any event is associated with metadata that is sufficient for information about the event, *e.g.*, DMA access details can be analyzed from page size, DMA burst size, and address range. This metadata is communicated by the security wrapper to E-IIPS, often under request from the E-IIPS (see below). Of course, in addition to standardized events, there are some IP-specific requirements in each IP. Our framework allows the SoC integrator to request additional security-critical events from specific IP which can then be mapped into its wrapper.

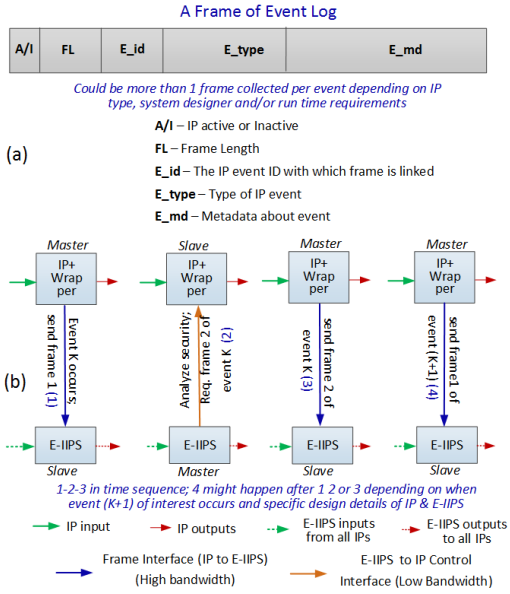

Fig. 3. (a) Fields of a typical event frame; (b) An example communication protocol between wrapper and security engine.

B. Security Wrapper Implementation

Our security wrapper design is frame-based, with a standard format for security-critical event definitions, which can be instantiated into corresponding events for specific IPs. A typical IP security wrapper architecture is shown in Fig. 2(a). Fig. 3(a) illustrates an event frame. The wrapper typically consists of an activity monitor logic (to identify whether the IP is active), an event type detector, and a buffer to store the event metadata. Some events require also corresponding local clock domains. The wrapper also incorporates registers (see below) that would be configured by E-IIPS at boot time to specify particular events which need notification. The frame-based interface is used to provide a standardized communication mechanism with E-IIPS. In general, E-IIPS provides two types of signals to a security wrapper: (1) `disable` to block IP functions (in varying granularity depending on policy) in response to a suspected security compromise, and (2) `request` to request more data or send controls. We exploit the existing boundary scan interface of the IP to transmit data in parallel shift/access mode in high bandwidth demands for certain functional security validation.

C. Security Policy Controller

E-IIPS acts as the "security brain" of the SoC, providing programmable interface to different security policies. Its key functionality is to analyze events communicated by the security wrappers, determine the security state of the system, and communicate IP-specific `request` and `disable` signals. Fig. 4 shows the top-level architecture of E-IIPS. The architecture includes the two major components, *viz.*, (1) a *Security Buffer* that provides access to the IP-specific event logs from the security wrappers, and (2) the *Policy Enforcer* that forms the analysis component of E-IIPS.

Security Buffer. The security buffer interfaces with the Policy Enforcer through a *buffer controller* that defines how the buffer frames are analyzed by the Policy Engine. We implement the buffer storage through a standard static segmentation scheme, permitting variable-length segments based on the volume of

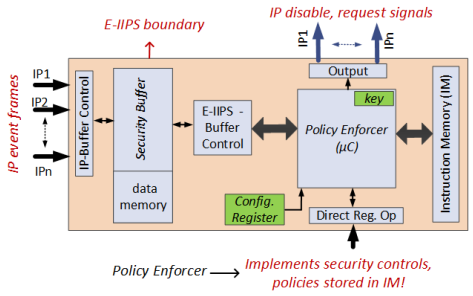

Fig. 4. Central E-IIPS top level architecture.

metadata. The event logs can be read by the controller through ports on the buffer (controlled by the buffer controller). The IP-buffer control logic maintains synchronization and coherence between the security wrappers and Control Engine with data frames from IPs at different read and write speeds, segment sizes, and event frequency.

Policy Enforcer. We implement the policy enforcer as a microcontroller-based implementation, which can be performed on a standard processor core. Functionally, the enforcer is a microcontrolled state machine, that asserts or deasserts the required `disable` or `request` signals for different IPs. In addition to a microcontrol engine, it also includes a standard instruction memory (for storing microcode or firmware implementing the policies), and a small amount of data memory for intermediate computation. The nature of the computation involved in security policy enforcement requires some custom modifications of existing commercial cores. We summarize a few of the illustrative necessary modifications.

- *Direct Register Writes.* Modifications to processor register file update logic is made to allow direct register updates, avoiding extra cycles for instruction and operand fetch from memory. This is necessary for time-sensitive policies, including TOCTOU and some access control policies.
- *Fused Datapaths and Secure Mode.* Often the event metadata width of an IP permits fusing the datapath (*e.g.*, two 32-bit registers into one 64-bit) which facilitates concurrent analysis of multiple frames.
- *Branch Prediction Buffers.* Branch prediction buffer design is a critical requirement for achieving low power and performance overhead, since security policy implementations involve the use of conditional branches with a much higher frequency than traditional application and system programs.

Finally, the E-IIPS module includes configuration registers to permit the SoC designer to activate only a subset of implemented policies for a specific application or use case. The register is configured at design time through a combination of fuse/antifuses and multiplexers. It also aids in extending E-IIPS as a plug-n-play standalone IP with a generic architecture.

D. Secure Authenticated Policy Upgrades

Recall from Section II that our threat model permits attacks through malicious firmware or software to subvert protection of system assets. Since E-IIPS itself is microcontrolled, it is also vulnerable to such attacks. Unfortunately, it is not possible to protect E-IIPS by merely disabling updates to its firmware. Since a key reason for a microcontrolled design is to permit policy upgrades on-field, it is critical to permit such upgrades through firmware updates. To address this problem, we implement an authentication mechanism based on challenge-response keys. Keys are generated at power-on using a standard technique based on Physically Unclonable Functions (PUF), that exploits intrinsic process variations in silicon to ensure robustness. Since keys are generated at power-on, we avoid on-chip key storage access control attacks through software or firmware. Finally, we avoid TOCTOU attacks during firmware updates by requiring single-threaded firmware updates, *i.e.*, a firmware update cannot be interrupted by an overlapping update request.

E. Policy Implementation in SoC Integration

Security policy implementation through our framework requires collaboration between IP developer and SoC system integrator.

IP Provider: The IP provider identifies the key standard security-critical events in the IP, based on the IP type (Table I); this content is incorporated into frames along with registers for configuration, to build the security wrapper.

SoC Integrator: The SoC integration team implements the security policies chosen for the application through the Policy Enforcer firmware of E-IIPS. Furthermore, since E-IIPS is centralized and IPs are delivered with security wrappers integrated, the SoC integration validation team is responsible for verification of policies against system-level use-cases through simulation, hardware accelerator, FPGA, or post-silicon.

IV. Use Case Scenarios

We present 2 use case scenarios of how generic security policies in the domain of secure-crypto and access control are mapped into our proposed architecture. The policy type, its function and the involved IPs in our implementation are shown in Table II.

TABLE II
Policies for Usage Case Analysis

Sec. Policy	*Desc.*	*IPs involved*
Secure Crypto Verification	Verify functionality of AES engine at power-on	E-IIPS, mem. cntrllr, crypto-proc., test-access cntrl.
Access Control	Prevent DMA access to system level addresses	E-IIPS, mem. cntrllr, DMA

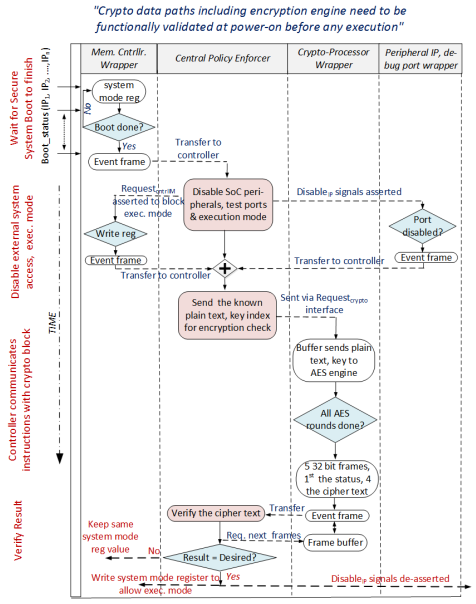

Fig. 5. Flow/message diagram representation of implementation of case I.

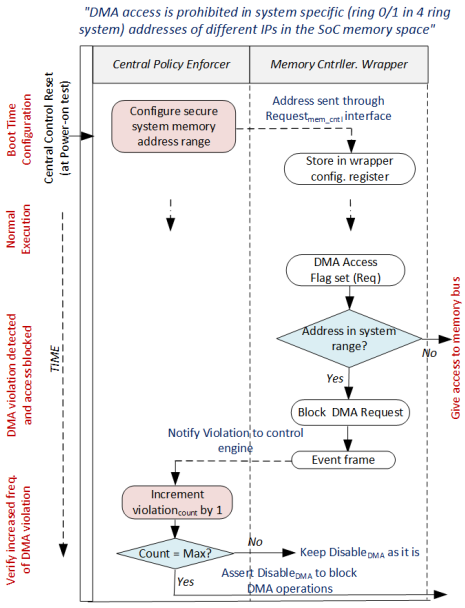

Fig. 6. Flow/message diagram representation of implementation of case II.

A. Use Case I: Secure Crypto

Policy: Crypto-processor data paths including encryption engine need to be functionally validated at power-on before any execution.

The above policy ensures trustworthiness of the system at boot time. The validation stipulated by the policy includes checks for correct operation of encryption (*e.g.*, AES) and hash (*e.g.*, SHA-1) engines as well as ensuring the stochasticity (randomness) of bits output by the True Random Number Generator (TRNG). Often an undetected functional failure of the crypto data path etc. results in compromise of the system security, mostly in terms of availability of resources (leading to denial of service attacks etc.). In this study, we provide a sample implementation of how a SoC designer maps the AES engine verification to the proposed platform. The flow of operations/messages between the E-IIPS and IP security wrappers through the standard interfaces is illustrated in Fig. 5.

In our implementation, the E-IIPS waits for the system boot process to finish (including power-on-self tests, firmware integrity check, system software load to memory) before proceeding with the AES verification. This ensures the full trustworthiness of other system components during the check. The "boot mode finish" is indicated by a particular value of system mode register, mapped to the memory. In response, the E-IIPS disables external system interfaces of peripheral cores, JTAG and other test/debug ports to eliminate possible attack surfaces. It also blocks transition to system execution mode. E-IIPS configures a set of known plaintext and key inputs in a buffer in the crypto-processor wrapper at boot time through potentially using the serial/parallel boundary scan interface for high bandwidth communication (not shown in Fig. 5). The appropriate crypto test access port settings are asserted by the JTAG/TAM controller, in response to E-IIPS configuration request during boot. The desired cipher outputs are stored inside E-IIPS boundary. E-IIPS sends a particular plaintext/key buffer index to crypto-wrapper for execution. The computed cipher text is communicated through frames (or boundary scan) to the E-IIPS for verification. If it matches the desired, the proactive holds are lifted and the system goes to the normal execution phase. The exact sequence in terms of event detection and message communications is summarized:

- Memory Control Wrapper (MCW) detects event "system boot finish" and transfers frame to E-IIPS.
- E-IIPS reads the event from security buffer and asserts the disable interface to peripheral cores and test/debug ports. It blocks system execution by write to register mapped in memory through the MCW request interface.
- Receiving confirmation through frames that these actions have been performed by IP wrappers, the E-IIPS sends the plain text and key buffer index through the crypto processor wrapper (CPW) request interface.
- The cipher text is computed, 5 frames are generated inside CPW, the first one containing the event "Encryption Complete" and metadata indicating that next 4 frames (of 32 bits) constitute the 128 bit cipher output.
- CPW sends the frame 1. E-IIPS sets the appropriate CPW request signals for the next 4 frames.
- E-IIPS verifies the computed cipher text.

This also shows an use case where the P1500 boundary scan infrastructure can be suitably used for high bandwidth data/control communication in our framework, thereby reducing routing complexity and overhead.

B. Use Case II: Access Control

Policy: Direct Memory Access (DMA) is prohibited in system-specific (ring 0/1 in 4 ring system) addresses of dif-

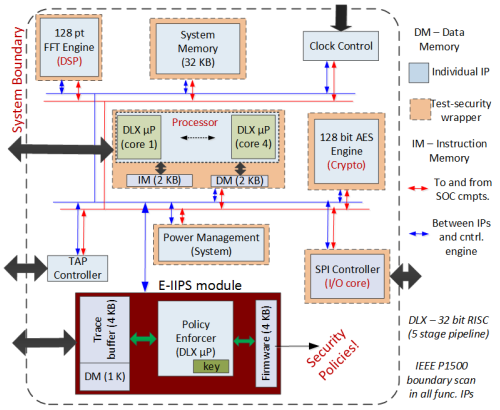

Fig. 7. Current architecture of functional toy SoC model

ferent IPs in the SoC memory space.

Most current SoCs involve DMA to the system memory through a dedicated DMA controller to reduce the workload on the processor cores. DMA by I/O peripherals (in memory-mapped I/O schemes) are utilized by attackers to snoop assets and modify system-level code. Policies like the one above protect against these security threats.

As illustrated in Fig. 6, E-IIPS configures the IP-specific system-level address ranges at boot time in the memory controller through its security wrapper (MCW). When an access from the DMA controller is detected by MCW, the requested address is checked with the system specific ranges inside the wrapper logic. In case of no violation, the system memory bus is granted for DMA. In case of system address overlap, the request is blocked, the violation is logged as an event, and is communicated to the E-IIPS through frames by MCW. E-IIPS maintains a buffer of DMA violations in the recent past. If the number exceeds a threshold within a set time (configured by SoC designer), memory access requests from the DMA controller are disabled. The specific events/message flows and interface signals, as shown in Fig. 6 are summarized:

- E-IIPS configures system address ranges in memory controller register through MCW request interface at boot.
- When a DMA request is detected, the MCW checks the corresponding address. If violation is detected, the request is blocked. The event is sent as a frame to E-IIPS.
- E-IIPS updates count of DMA violations and compares with threshold. If the number exceeds within set time limit, the DMA controller operations are disabled through it's disable interface.

V. Overhead Results

Given the dearth of appropriate open-source SoC design models to perform experiments, we are in the process of developing a simple SoC design to assist in the current research. Our toy model has IPs of different functionalities interacting with each other to perform specific system functions. The IPs are obtained from opencores [14] in Verilog RTL models. The current version of this model at the time of this writing is illustrated in Fig. 7. In this model, all memory are implemented as register files for ease of synthesis. IP address spaces are mapped to the memory. These IPs can access the memory directly, analogous to DMA. At present, all IP-IP communications are point-to-point. The model has been functionally validated in Modelsim.

All IPs are wrapped with security abstraction layers. Events detected by these wrappers include a major representative subset of those listed in Table I, *e.g.*, read/write requests to memory, duration of processes, specific conditional jumps in μP core, transfer start/end of SPI module etc. The 32-bit frame formation logic and metadata buffers are present. IPs contain configuration registers which are configured by E-IIPS at boot time (master system reset asserted). The E-IIPS is implemented with a single DLX 32-bit RISC core (5 stage pipeline). Firmware (security policies) is stored in local instruction memory of 4 KB. 2 bit disable and request signals are output to each IP from the controller.

To obtain representative overhead values, the IPs were synthesized at 32nm predictive technology library. The calculated area, dynamic (at 1GHz clock) and leakage power overheads are provided in Table III. The overheads are mostly minimal. In some scenarios, the power reduces after re-synthesizing with wrappers due to internal (heuristic) optimizations and hence reported as negligible. E-IIPS was synthesized at 32nm and the resulting area and power (at 1GHz clock) values are provided in Table IV. Finally, the area overhead of the control engine was estimated with respect to our toy model and commercial SoCs from Apple and Intel at 32nm ($\sim$ 1GHz clock) and provided in Table V. The 32KB system memory (major area component) area was estimated with established SRAM models. As our toy SoC is rather small with only handful of IPs, the overhead is comparatively higher. The overhead for the controller is minimal in realistic scenarios. For a generic SoC design, we can conclude that the hardware overheads due to the proposed architecture would be minimal. After analysis with NoC fabrics of different types, the routing complexity and transfer power/energy would be evaluated as part of future work.

VI. Related Work

The notions of high-level security requirements in a computing system were developed in the 1990s as part of research on information security [6]. Early research on security

TABLE III
Area & Power Overhead of IP Security Wrapper (at 32nm)

IP	Orig. Area(μm^2)	Area Ovhd(%)	Dyn. Pw. Ovhd(%)	Leak. Pw. Ovhd(%)
AES(128 bit)	101620	2.1	–	–
SPI Controller	3947	9.2	11	9.7
DLX μP core (w. 4KB I/D mem.)	290496	6.8	–	5.4
FFT(128 point)	1810	10.2	–	16.1

- negligible

TABLE IV
AREA & POWER OF CENTRAL SECURITY CONTROLLER(AT 32 NM)

Die Area(μm^2)	Dynamic Power(mW)	Leakage Power(mW)
2831860	13.67	34.13

TABLE V
DIE AREA OVERHEAD OF CENTRAL CONTROLLER(AT 32 NM)

SoC	Die Area(μm^2)	Overhead of controller(%)
Our Toy Model	$13.1X10^6$	21.7
Apple A5 (APL2498)	$69.6X10^6$	4.06
Intel Atom Z2520	$\sim 40X10^6$	7.1

policies looked primarily on software systems and developed analysis frameworks for access control and information flow policies [10], [15]. More recently, researchers have tried to develop languages for formal representation of hardware security policies [2]. On the other hand, with the increasing prominence of SoC designs, there has been significant research interest in SoC security. However, most recent research in this area has focused on *hardware security*, *i.e.*, protection of the system against a malicious hardware Trojan [16], various forms of counterfeiting attacks [17], and attacks to leak secret information through side channels or on-chip resources such as scan or debug infrastructure. A recent work has also targeted developing a centralized and scalable framework, referred to as an infrastructure IP for security, for efficiently realizing countermeasures against these attacks [5]. Infrastructure IPs refer to a range of IPs that are dedicated to facilitate SoC functional verification, testing or yield improvement [18]. On the contrary, the proposed solution can be viewed as an extended infrastructure to implement system-level security policies in SoC. There are also number of works that report efficient protocols [3] involving functional IP blocks, crypto IPs, and security wrapper and associated architecture-level custom modification for specific security policies [4]. However, they do not propose a generic architecture involving a reusable centralized IP and its flexible interface with the IP blocks, as proposed in this paper.

VII. CONCLUSION

We have presented a novel architectural framework for implementing diverse security policies in a system-on-chip. It enables systematic implementation of security policies, and hence can greatly facilitate the process of secure SoC design involving various security assets and can potentially reduce the design/hardware overhead. Furthermore, it enables effective validation and debug and update of security policies during post-silicon validation, which often imposes major roadblocks in SoC production cycle. The architecture consisting of a centralized security policy controller, referred to as E-IIPS, and generic security wrapper per IP block, is easily scalable to large number of IPs of varying structural properties. The proposed E-IIPS module builds on the functionality of recently reported infrastructure IP for security, which was proposed for implementing multitude of security functions (e.g. protection against scan-based attack, Trojan attacks, power analysis attack and hardware piracy) using a shared centralized architectural fabric. The E-IIPS module, when combined with the capability of implementing conventional hardware security functions, can serve as an even more powerful security infrastructure IP. We have verified functional correctness of the architecture through extensive simulations and evaluated the hardware overhead. The overhead for the proposed architecture is expected to go down significantly for realistic SoCs. Future work will include automatic synthesis of security policies into the proposed architectural fabric and extension of the architecture to account for untrusted third-party IPs.

VIII. ACKNOWLEDGEMENT

This research was supported in part by the National Science Foundation (NSF) under Grants CNS-1054744 and DUE-1245756 and Semiconductor Research Corporation (SRC) under grant 2014-HJ-2507.

REFERENCES

[1] John Rushby, "Noninterference, Transitivity, and Channel-Control Security Policies," SRI, Tech. Rep., 1992.
[2] X. Li, "Sapper: A Language for Hardware Level Security Policy Enforcement," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.
[3] S. Krstic, J. Yang, D. W. Palmer, R. B. Osborne, and E. Talmor, "Security of SoC Firmware Load Protocol," in *IEEE HOST*, 2014.
[4] M. Sastry, I. Schoinas, and D. Cermak, "Method for enforcing resource access control in computer system," in *US Patent 20120079590 A1*.
[5] X. Wang, Y. Zheng, A. Basak, and S. Bhunia, "IIPS: Infrastructure IP for Secure SoC Design," *IEEE Transaction on Computers*, 2014.
[6] S. J. Greenwald, "Discussion Topic: What is the Old Security Paradigm," in *Workshop on New Security Paradigms*, 1998, pp. 107–118.
[7] M. Miettinen, S. Heuser, W. Kronz, A. Sadeghi, and N. Ashokan, "ConXsense: automated context classification for context-aware access control," in *ASIACCS*, 2014, pp. 293–304.
[8] M. Conti, B. Crispo, F. Fernandes, and Y. Zhauniarovich, "CRePE: A system for enforcing Fine-grained Context-related Policies on Android," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 5, pp. 1426–1438, 2012.
[9] R. Hull, B. Kumar, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas, "Enabling Context-aware and Privacy-conscious User Data Sharing," in *2004 IEEE International Conference on Mobile Data Management*, 2004, pp. 187–198.
[10] J. Goguen and J. Meseguer, "Security Policies and Security Models," in *Proc. 1982 IEEE Symposium on Security and Privacy*, 1982, pp. 11–20.
[11] T. Amtoft, S. Bandhakavi, and A. Banerjee, "A Logic for Information Flow in Object-Oriented Programs," in *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2006)*. ACM Press, Jan. 2006, pp. 91–102.
[12] B. Alper and F. B. Schneider, "Recognizing Safety and Liveness," *Distributed Computing*, vol. 2, no. 3, pp. 117–126, 1987.
[13] N. Borisov, R. Johnson, N. Sastry, and D. Wagner, "Fixing Races for Fun and Profit: How to Abuse Atime," in *Proceedings of the 14th USENIX Security Symposium*, 2005, pp. 303–314.
[14] "www.opencores.com."
[15] J. T. Haigh and W. D. Young, "Extending the Non-Interference Version of MLS for SAT," in *Symposium on Security and Privacy*, 1986.
[16] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A Statistical Approach for Hardware Trojan Detection," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2009.
[17] U. Guin, D. DiMase, and M. Tehranipoor, "Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead," *Journal of Electronic Testing*, vol. 30, no. 1, pp. 25–40, 2014.
[18] Y. Zorian, "Embedded memory test and repair: Infrastructure IP for SOC yield," in *International Test Conference*, 2002, pp. 340–349.