Correctness and Security at Odds: Post-silicon Validation of Modern SoC Designs

Sandip Ray Jin Yang
Strategic CAD Labs, Intel Corporation
Hillsboro, OR 97124. USA
{sandip.ray, jin.yang}@intel.com

Abhishek Basak Swarup Bhunia
EECS Dept., Case Western Reserve University
Cleveland, Ohio 44106. USA
{axb594, skb21}@case.edu

Invited Paper

ABSTRACT

We consider the conflicts between requirements from security and post-silicon validation in SoC designs. Post-silicon validation requires hardware instrumentations to provide observability and controllability during on-field execution; this in turn makes the system prone to security vulnerabilities, resulting in potentially subtle security exploits. Mitigating such threats while ensuring that the system is amenable to post-silicon validation is challenging, involving close collaboration among security, validation, testing, and computer architecture teams. We examine the state of the practice in this area, the trade-offs and compromises made, and their limitations. We also discuss an emerging approach that we are contemplating to address this problem.

1. INTRODUCTION

Recent years have seen the emergence of System-on-Chip (SoC) design technology as a centerpiece in computing system architecture. These systems encompass a diverse range of platforms and form factors, including smartphones, tablets, automotive controls, medical instruments, etc. The trend is towards even higher proliferation and diversification as we move towards a future with Internet of Things, wearables, implants, and smart sensors. A consequence of these pervasive and personalized applications is that modern SoC designs include a large amount of highly sensitive assets that must be protected from unauthorized and malicious access. Examples of assets included in a typical SoC designs are cryptographic keys, firmware, operation modes, programmable fuses, etc. Unauthorized access to secure assets can have catastrophic consequences, including loss of billions of dollars in economy, threats to personal and national security, even large-scale destruction of human life [1]. Furthermore, malicious attacks themselves are becoming increasingly sophisticated and diverse, including attacks on the underlying hardware, exploitation of system-level bugs through software or firmware, and reverse-engineering information from communication of the device with other systems. Consequently, security validation is a key component of the SoC verification and validation flow. Indeed, many security requirements (*e.g.*, data integrity, authentication, privacy requirements, access control policies, etc.) are considered part of the correctness specification and form key targets for functional validation. Furthermore, security architectures often form a cooperative counterpoint to validation, providing built-in resiliency to vulnerabilities.

However, there is one critical situation where security constraints are at odds with validation requirements, *viz.*, post-silicon validation and debug. Post-silicon validation entails executing tests and applications on a fabricated, pre-production silicon implementation; the goal is to ensure that the fabricated silicon operates correctly under actual operating conditions with real applications. The goal is to detect errors that are missed in pre-silicon validation (*e.g.*, simulation and emulation of RTL and software models), check for compatibility with software and applications, identify electrical noise margins, determine frequency ranges for reliable operations, etc. Post-silicon validation is a critical and complex activity performed under aggressive schedules. It represents more than 50% of the validation cost of a modern IC design [2]. Perhaps more importantly, mass production of the design can start only after the product has been "vetted" by post-silicon validation; delays in post-silicon schedule can result in a company missing product release deadlines or even having to cancel the production due to missed market opportunities, with consequent loss in revenues, reputation, and market share [3]. It is therefore critical to enable fast, streamlined workflow in post-silicon validation.

Why are security constraints at odds with post-silicon validation requirements? Post-silicon validation requires instrumentation of the design with a significant amount of additional circuitry, often referred to as *Design-for-Debug* or DfD circuitry, to provide requisite observability and control during silicon execution. Unfortunately, instrumentations can also account for significant security vulnerabilities. In particular, access to security assets are governed by complex, subtle, and often ambiguous system-level policies involving multiple IPs[1] in the SoC design, and it is tricky to determine whether an innocuous instrumentation has com-

DAC '15 June 07 – 11, 2015, San Francisco, CA, USA
http://dx.doi.org/10.1145/2744769.2747917

[1]For the purpose of this paper, an "IP" or "intellectual property" is a hardware or software block designed to provide a specific functionality. SoC design architectures typically involve composition of a number of IPs, many of them standardized and pre-designed, interacting with one another through an interface of communication fabrics.

promise some of these policies. To exacerbate the problem, some of the DfD circuitry must remain enabled even after post-silicon validation, *e.g.*, when the product is shipped to customers. This is done to ensure correlation of timing and power characteristics between the shipped product and that used for validation, and to provide "hooks" for debugging or patching problems discovered on-field. Unfortunately, it also means that any security vulnerability that remains undetected from DfD is also available for exploitation on-field.

In this paper, we discuss trade-offs between security and post-silicon validation in modern SoC designs. We examine the state of the industrial practice and its inadequacy, and discuss some current and emerging approaches.

In spite of its importance, trade-offs between validation and security have received scant attention of researchers in academia or in industrial research. Much of the solutions that exist in the industrial practice today are ad hoc, point solutions based on designers' experience and expertise. We believe that a viable approach to this problem will require a collaborative enterprise between security, debug, and computer architecture research. A key goal of this paper is to bring the problem to the attention of these communities, explain its criticality, and explain the constraints that must be satisfied for the solution to be viable.

The remainder of the paper is organized as follows. Section 2 provides the relevant background on post-silicon validation, DfD, and SoC security policies, and describe some security vulnerabilities arising from conflicts with post-silicon debug. In Section 3, we expand upon the problem, identifying the key questions involved in the trade-offs between security and validation. In Section 4, we discuss some approaches used in current industrial practice, and their limitations. Section 5 discusses some of the challenges that need to be addressed in developing an effective solution, accounting for the different stake-holders involved and constraints from each stake-holder. Section 6 covers some emerging approaches, and Section 7 concludes the paper.

2. BACKGROUND

2.1 SoC Security

Modern SoC designs include a large number of critical assets which must be protected from unauthorized or malicious access. Some examples of security assets common in most modern computing systems are cryptographic and DRM keys, premium content, programmable fuses, firmware execution flows and private information of the end user. Unfortunately, recent years have also seen significant sophistication diversity of security attacks on these systems. Figure 1 illustrates the diversity of potential security attacks in a modern smartphone. Unsurprisingly, security is a critical parameter in the architecture and design of a modern SoC design. Security considerations in an SoC design can be broadly divided into the following three categories.

Hardware Security: This refers to security issues arising from problems in the underlying hardware. In particular, malicious hardware can be introduced during the system development or fabrication by a participant in the SoC supply chain either inadvertently or with malicious intent. Examples of malicious hardware include hardware Trojans [4] introduced by a malicious IP vendor or system integrator (possibly through a malicious design automation tool)

Figure 1: Some Potential Attacks on a Modern Smartphone

which can leak secret information, counterfeit or cloned ICs from the foundry [5], etc.

System or Platform Security: This refers to vulnerabilities resulting from functional or performance bugs in the system that can be exploited by a malicious third party during execution [6]. Examples of such vulnerabilities include functional bugs in security-critical IPs (*e.g.*, cryptographic engine), information leakage due to unanticipated behavior when the system encounters inputs of unexpected types, information leakage from system power profile, etc.

Cloud Security: This refers to vulnerabilities arising from communication of an embedded computing system either with other embedded systems in the Internet or with servers and data centers in the cloud. It includes eavesdropping or "man in the middle" attacks, breach of confidentiality in the stored data in the cloud, etc.

For this paper, we will primarily focus on issues related to system and platform security, although much of the discussion can be generalized to other categories.

What does a "security vulnerability" mean? At a high level, the definition of security requirement for assets in a SoC design follows the well-known "CIA" paradigm, developed as part of information security research [7]. In this paradigm, accesses and updates to secure assets are subject to the following three requirements:

- **Confidentiality:** An asset cannot be accessed by an agent unless authorized to do so.
- **Integrity:** An asset can be mutated (*e.g.*, the data in a secure memory location can be modified) only by an agent authorized to do so.
- **Availability:** An asset must be accessible to an agent that requires such access as part of correct system functionality.

Of course, mapping these high-level requirements to constraints on individual assets in a system is non-trivial. This is achieved by defining a collection of security policies that specify which agent can access a specific asset and under what conditions. Following are two examples of representative security policies. Note that while illustrative, these examples are made up and do not represent security policy of a specific company or system.

Example 1: During boot time, data transmitted by the cryptographic engine cannot be observed by any IP in the SoC other than its intended target.

Example 2: A programmable fuse containing a secure key can be updated during manufacturing but not after production.

Example 1 is an instance of confidentiality, while Example 2 is an instance of integrity; however, the policies are at a lower level of abstraction since they are intended to be translated to "actionable" information, *e.g.*, architectural or design features. The above examples, albeit hypothetical, illustrate an important characteristic of security policies: the same agent may or may not be authorized access (or update) of the same security asset depending on (1) the phase of the execution (*i.e.*, boot or normal), or (2) the phase of the design life-cycle (*i.e.*, manufacturing or production). These factors make security policies difficult to implement. Exacerbating the problem is the fact that there is typically no central documentation for security policies; documentation of policies can range from microarchitectural and system integration documents to informal presentations and conversations among architects, designers, and implementors. Finally, the implementation of a policy is an exercise in concurrency, with different components of the policy implemented in different IPs (in hardware, software, or firmware), that coordinate together to ensure adherence to the policy.

Given the above factors, validation is a crucial aspect of SoC security policy. Security validation encompasses the entire design life-cycle, including both pre-silicon and post-silicon phases. Security validation encompasses a variety of activities, including threat modeling, fuzzing, penetration testing, and hackathons.

2.2 Post-silicon Validation and DfD

Post-silicon validation is another crucial activity in a SoC designs. It entails running tests and software on a fabricated, pre-production silicon. The goal is to ensure that the system works under actual operating conditions. Since the silicon operates at target clock speed, its execution speed is about a billion times that of RTL simulation, and even several orders of magnitude higher than hardware accelerators, emulators, etc. This permits exploration of deep design states and running real applications (*e.g.*, booting and executing an operating system or software applications) which are infeasible during pre-silicon validation. Indeed, a significant amount of security properties are validated during post-silicon validation through penetration testing or hackathon. Furthermore, it provides an opportunity to explore and validate non-functional behaviors of the design, *e.g.*, power, performance, frequency limitations, noise margins, etc.

Unfortunately, post-silicon validation is also a and complex activity, requiring elaborate planning and involving a large number of stake-holders. Figure 2 illustrates the number of complex activities that must be performed throughout the design cycle to enable post-silicon validation.

In this paper we confine ourselves to one of its critical requirements, *e.g.*, on-chip instrumentation, or DfD. Its primary goal is to provide observability and control of internal design states during silicon execution. Limited observability and control are fundamental challenges in post-silicon validation: due to limitations in the number of output pins and area/power overhead of internal trace buffers, only a few

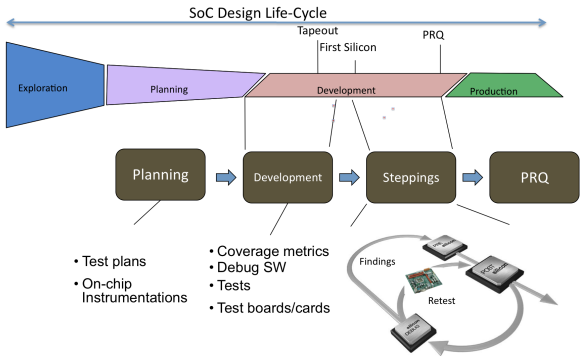

Figure 2: Activities Pertaining to Post-silicon Validation Throughout a System Design Life-cycle. The validation activity occurs in the timeframe between the First Silicon and PRQ. However, planning for the activity starts much earlier, aligned with the product planning itself. PRQ or or "Product Release Qualification" is the decision point to initiate mass production.

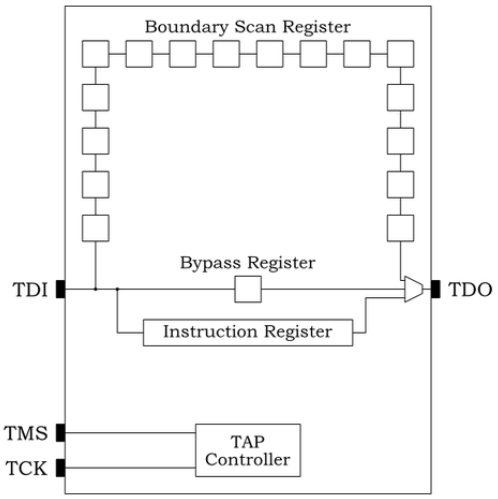

Figure 3: Basic JTAG architecture. The basic operations include capturing the value of a register, shifting out the value on TDO, shifting in a new value from TDI, and updating the register with this new value. The bypass register is used for bypassing the update when boundary scan is not used.

hundreds among the millions of internal signals of the design can be observed during silicon execution. Furthermore, for a signal to be observed, the design must be instrumented *a priori* with appropriate control hardware that routes the signal to an observation point. Consequently, the DfD architecture must be determined early in the design flow, typically through analysis of the functional architecture and the available RTL implementation of some of the component IPs. Indeed, a significant amount of recent research in post-silicon validation has focused on approaches to identifying, through analysis of pre-silicon design collaterals, signals to be observed and controlled during silicon execution [8–10]. The DfD architecture is defined based on this analysis, and realized as a collection of IPs which are integrated with the system as part of the SoC integration process.

In addition to internal instrumentation for observability and controllability, DfD also governs the interface through which data is transported off-chip. One critical transport interface is the IEEE *Standard Test Access Port and Boundary Scan Architecture*, commonly referred to as JTAG [11].

It is part of the IEEE Std. 1149.1-1990 and was originally devised to perform board-level testing for printed circuits using boundary scan. However, because of its standardization and extensive documentation, it is one of the most important components for off-chip data transport available in virtually all digital ICs, and has therefore found extensive application in post-silicon validation. Figure 3 shows the basic JTAG architecture. It includes a *test access port* (TAP) used by the tester or external host to interact with the internal scan network of the design, together with facilities for state switching, shifting data in and out of scan registers, and executing the system from a specific state under the control of a customized instruction set. In addition to supporting standard JTAG, many microprocessor and SoC design vendors have their own extensions, in the form of an extended instruction set. In addition to JTAG, modern SoC designs carry several other transport mechanisms [12].

2.3 Security Exploits through Debug

There have been several attacks on SoC designs that exploit the debug interface. One well-advertised attack was a hack on XBOX 360 gaming control: using the JTAG interface, it was possible to upload a buggy firmware that permitted execution of unauthorized code [13]. Indeed, exploiting JTAG to install buggy or malicious firmware is a well-known approach to circumvent system security; similar approaches have been demonstrated for jail-breaking smartphones or unlocking premium services without authorization [14].

In addition to the above, it is possible to exploit instrumentation for observability and control to thwart system security. Such exploits can be very subtle and difficult to determine in advance, while having a devastating impact to the product and company reputation once carried out. Consequently, a "knee-jerk" reaction is to restrict DfD features available in the design. On the other hand, lack of DfD may mean making post-silicon validation difficult, long, and even intractable. This may mean delay in product launch; with aggressive time-to-market requirement, a consequence of such delay can be loss of billions of dollars in revenue or even missing the market for the product altogether.

3. TRADE-OFF CHALLENGES BETWEEN SECURITY AND DEBUG

The trade-off challenge between security and validation is the following. For post-silicon validation, we must to observe internal design behaviors during system execution; however, security policies on certain assets may disallow their observability. Put this way, the challenge may appear to be an instance of inconsistency between requirements from availability and confidentiality/integrity as follows. The DfD architecture is, after all, a collection of IPs that need access to some internal data as part of its correct functionality at different points of system execution; this need may be appropriately viewed as an availability requirement. On the other hand, a security policy that denies observability is an instance of a confidentiality or integrity requirement. Such trade-offs are abound in SoC designs, *e.g.*, an analogous trade-off is between (1) the need for two IPs to communicate data across the fabric in an SoC to satisfy some protocol (Availability); and (2) the requirement to restrict the communication when there is a possibility of eavesdropping in the channel by an unauthorized third IP (Confidentiality). Indeed, debug requirements can be specified as security policies protecting specific assets. Consider a third-party IP $\mathcal{A}$ that includes secret keys from an IP vendor. Then the following security policy may be imposed to protect the keys from being exposed to the SoC design house during system-level post-silicon validation. As with all previous policy examples, this one is also hypothetical albeit reasonable.

Key Protection Policy: If the system operates in debug mode then $\mathcal{A}$ responds to key request with a dummy key; otherwise $\mathcal{A}$ responds with the actual key.

If the restriction on DfD is indeed specified as a security policy as above, then clearly it can be treated and validated like any other security policy. Unfortunately, several factors make the trade-off between security and debug more challenging than a general conflict between confidentiality and availability. Here we discuss some of the key factors.

Ambiguity: Observability requirements are rarely as clear-cut as requirements arising from functionality. A key reason is that it is unclear a priori which component of the system would exhibit a bug and therefore should be a target for observability. Indeed, much of DfD is designed in an ad hoc manner based on experience of the designers and validators on past systems. Furthermore, DfD decisions are made by validators and designers having relatively little familiarity with security policies. Consequently, observability architectures tend to focus primarily on functionality. When security constraints are imposed, often late in the design, one of the following two situations is likely: (1) some critical observability or control is inadvertently removed as a conservative measure; or (2) some subtle security flaw remains.

Feed-through: Security requirements may affect observability indirectly. Consider a signal s in IP $\mathcal{A}$ that we wish to observe during post-silicon. Assume further that observing s does not compromise any security policy. However, in order for s to be observed, its value must be routed to an observation point, *e.g.*, an output pin or system memory. If this route includes a high-security IP $\mathcal{B}$ then confidentiality requirement may cause $\mathcal{B}$ to be unobservable during system execution, thereby making s unobservable as well. On the other hand, the placement of IPs $\mathcal{A}$ and $\mathcal{B}$ in the system layout, and consequently, the route of signal s, may only be determined at an advanced stage of the design life-cycle making it impossible to account for that consideration when defining the signals to trace.

Lack of Centralization: Both DfD and security components are sprinkled across a large number of IPs in the SoC design. This, coupled with the lack of a rigorous documentation or specification of DfD requirements (and in many cases, also security policies), implies that it is sometimes unclear what the purpose of a specific feature is, how it is excited, and what vulnerabilities it exposes. This makes it extremely hard to analyze or determine security risks arising from DfDs. Consequently, depending on the target market segment, SoC design products tend to be either too conservative (*i.e.*, secure, difficult to debug) or too aggressive (*i.e.*, a significant amount of DfD, risking security).

4. STATE OF THE PRACTICE

In spite of the murky situation painted above, obviously SoC designs with security collaterals are being created, de-

signed, implemented, and produced for mass-market. Traditionally, a general principle regarding the trade-off between security and validation for IC design has been to progressively increase security features (and decrease DfD) as the design progresses along its life-cycle, from design to manufacturing, and production. Disabling DfD is possible permanently through blowing fuses during manufacturing and production. The situation is more complex for modern SoC designs, with the need to keep DfD features available for patching the product on-field. Nevertheless, the progressive increase is still a valid principle with a few adjustments. The first adjustment is that one cannot *permanently* disable DfD features because of the need to address this principle. Second, when such reversal is needed it is only for specific stake-holders with special authentication (*e.g.*, an entity authorized to patch a design functionality). Finally, the reversal must be temporary, and once the activity needing the reversal (*e.g.*, fixing an on-field bug) is complete, the system reverts to its default "higher security" state appropriate for the current phase of its design life-cycle.

The above approach requires identification of security assets, their default security needs at specific points of the design life-cycle, and authentication requirements for reversal. This is typically achieved by understanding (manually) and deconstructing the intended modes of operation of the system, identifying the assets that need protection at each mode, and ensuring that the accesses to those assets, even through debug interface, follow authentication. Authentication requirements are typically imposed at the transport interfaces (*e.g.*, JTAG), rather than at individual IPs. In particular, since JTAG is a common data transport architecture there has been several recent efforts on developing authentication mechanisms for JTAG, including challenge-response schemes [15] and multi-level authentication [16].

Note that the approach can leave open the possibility for an unauthorized IP to snoop the debug information in the system's internal fabric and obtain unauthorized information about a classified security asset. Addressing such issues is typically left to creativity of individual architects and designers, and their solutions are validated (or broken) through penetration testing or hackathons.

5. CONSIDERATIONS FOR DEVELOPING A COMPREHENSIVE SOLUTION

The approaches discussed above are at best pragmatic workarounds. A key problem in developing a viable, comprehensive solution is that both post-silicon validation and security definition and architecture for modern SoC designs are complex and elaborate processes, involving significant planning and a large number of stake-holders. Consequently, any solution to their trade-off problem must necessarily address a large number of parameters. In this section, we highlight some of these parameters. The goal is not to be complete, but to provide the reader a flavor of the diversity of constraints and requirements involved.

HVM Considerations. High-volume manufacturing test is the process of identifying manufacturing defects during production. This is done by placing the fabricated silicon in a tester, where it is exercised with a large number of test vectors. The test patterns are generated by accounting for the functional definition of the design under test, the target faults, a fault model, the fabrication process technology, etc. The accuracy of coverage inferred from the results of these tests is highly sensitive to the test patterns being applied and the fidelity of the silicon design with respect to its pre-silicon netlist model. Consequently, it is important that irrespective of security constraints and access control restrictions, the test patterns work the same way as much as possible on silicon designs as expected from pre-silicon models, and their results accurately observed.[2] Furthermore, it must be possible to have a simple access to the IP being exercised with the test, without requiring too many workarounds.

Reusability: A key source of complexity in the current state of the practice discussed in Section 4 is the need to manually identify assets and accesses for different products and usage scenarios. This job is highly tedious and error-prone. Consequently, solution to the problem must provide a reusable infrastructure for systematically identifying and classifying assets and analyzing usage scenarios.

Late Variability: DfD is notorious for late changes in requirements and implementation. Indeed, DfD requirements can change during IP design, SoC integration, or even after a silicon stepping; the latter can happen on realization that observability or control of certain signals is critical for a future stepping. Consequently, any solution for addressing security challenges with DfD must be easy to adapt with such changing requirements. In particular, it should be possible to quickly validate an updated DfD architecture against a given set of security policies and identify vulnerabilities.

Self-Securability It is obvious that any architecture introduced to address the security-validation trade-off must be self-securing and must not introduce additional security back-doors (or complexity with debug).

Architecture: It is critical for any architecture that permits the trade-off to be centralized. The reason is that a decentralized architecture (both for security and DfD) is difficult to follow and can accidentally break or introduce vulnerability. To circumvent this possibility, it is critical that the architecture can be viewed as a centralized IP which can itself be effectively analyzed for possible violation of either security or debug requirements.

6. EMERGING APPROACH: IIPS

We are working on an architecture, named *Infrastructure IP for Security* or *IIPS*, to address the security-validation trade-offs. Here we briefly discuss IIPS and how we contemplate its use to address the problem. Note that the architecture is preliminary and a work in progress. We discuss it not to put it forward as a complete solution but to illustrate our thinking regarding potential approaches to the problem.

The goal of IIPS is to provide an easy-to-integrate, scalable infrastructure IP that serves as a centralized resource for SoC designs to protect against diverse security threats at minimal design effort and hardware overhead. The centralized IP serves as a plug-and-play reusable IP for SoC designers. It interfaces with functional IPs in the system through an IEEE 1500 Standard Embedded Core Test (SECT) interface, which forms a standard interface for communication among IP blocks in most SoC designs. Current application

[2]We say "as much as possible" since it is not possible to have exact equivalence, *e.g.*, if the system uses dummy keys in debug mode and test result depends on value of keys.

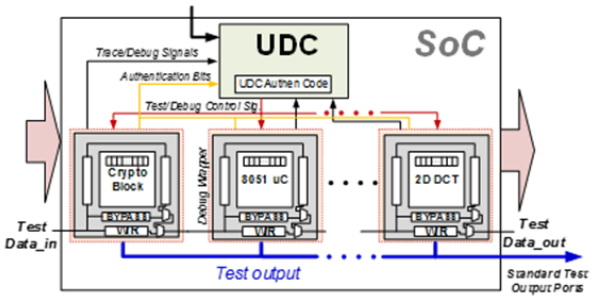

Figure 4: Schematic of architecture for the proposed secure debug framework in a representative SoC model.

target for IIPS is to protect the SoC design against hardware security attacks possibly from rogue or counterfeit IPs [17]. However, we are contemplating extension to implement secure post-silicon debug. In particular, we are currently extending the paradigm into a centralized, firmware-controlled framework to implement system level security policies. A representative schematic of the debug controller with distributed debug interfaces on individual IP blocks is illustrated in Figure 4. The policies include requirements for debug and security during normal, test, and boot modes. Since the extended framework is firmware-upgradeable, an identified vulnerability or an update to security or debug requirement can be fixed through a firmware update rather than hardware modifications or silicon respin. The firmware code is stored in an embedded non-volatile memory in the IIPS module and upgrades to secure debug policies is performed only through a strong authentication process to prevent any unauthorized access. We anticipate the architecture to eventually provide a systematic, formal approach to SoC security policies and enable automated synthesis of instrumentation based on these policies both during the SoC design phase and post-silicon validation.

7. CONCLUSION

We have discussed trade-offs arising from conflicting requirements from security and post-silicon debug in modern SoC designs. We discussed the scale of the problem, the challenges involved, the state of the practice, and some of the factors to be considered for a comprehensive solution.

One may ask whether this trade-off is new during the SoC era. The answer is "no". Clearly, similar trade-offs needed to be addressed for servers, desktops, and laptops. However, heterogeneity of constituent IPs and high integration complexity on the one hand, and the diversity of both security assets and debug requirements on the other, make the problem significantly more challenging for SoC designs.

We have not put forward any solution to the problem in this paper. Our goal has been to bring the problem to the attention of a broad audience in the research community. We believe a comprehensive solution to the problem can only be achieved through close collaboration among architects, security experts, validators, and VLSI testers.

8. ACKNOWLEDGMENT

The work is supported in part by Semiconductor Research Corporation (SRC) grant 2014-HJ-2507 and National Science Foundation Grants 1054744, 1245756, and 1441705. We thank our colleagues in Intel Labs and Intel Security Center of Expertise for suggestions and advice.

9. REFERENCES

[1] NIST. Computer Security Division 2005 Annual Report. Technical report, NIST Technology Administration, US Department of Commerce, 2005.

[2] A. Nahir, A. Ziv, R. Galivanche, A. J. Hu, M. Abramovici, A. Camilleri, B. Bentley, H. Foster, V. Bertacco, and S. Kapoor. Bridging Pre-silicon Verification and Post-silicon Validation. 47*th Design Automation Conference*, pages 94–95, 2010.

[3] S. Yerramili. Addressing Post-silicon Validation Challenge: Leverage Validation and Test Synergy. *International Test Conference*, 2006.

[4] R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia. MERO: A Statistical Approach for Hardware Trojan Detection. *Cryptographic Hardware and Embedded Systems*, pages 396–410, 2009.

[5] U. Guin and D. DiMase and M. Tehranipoor. Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead. *Journal of Electronic Testing*, 30(1):25–40, 2014.

[6] John Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical report, SRI, dec 1992.

[7] S. J. Greenwald. Discussion Topic: What is the Old Security Paradigm. In *Workshop on New Security Paradigms*, pages 107–118, 1998.

[8] K. Basu, P. Mishra, and P. Patra. Efficient Combination of Trace and Scan Signals for Post-silicon Validation and Debug. In *International Test Conference*, pages 1–8, 2011.

[9] D. Chatterjee, C. McCarter, and V. Bertacco. Simulation-based Signal selection for State Restoration in Silicon Debug. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 595 –601, 2011.

[10] K. Rahmani, P. Mishra, and S. Ray. Scalable Trace Signal Selection Using Machine Learning. 31*st IEEE International Conference on Computer Design*, pages 384–389, 2013.

[11] IEEE Joint Test Action Group. IEEE Standard Test Access Port and Boundary Scan Architecture. *IEEE Std.*, 1149(1), 2001.

[12] E. Ashfield, I. Field, P. Harrod, S. Houlihane, W. Orme, and S. Woodhouse. Serial Wire Debug and the CoreSight^{TM} Debug and Trace Architecture, 2006.

[13] Homebrew Development Wiki. JTAG-Hack. `http://dev360.wikia.com/wiki/JTAG-Hack`.

[14] L. Greenemeier. iPhone Hacks Annoy AT&T but Are Unlikely to Bruise Apple. *Scientific American*, 2007.

[15] C. J. Clark. Anti-tamper JTAG TAP design enables DRM to JTAG registers and P1687 on-chip instruments. In *Hardware-Oriented Security and Trust*, pages 19–24, 2010.

[16] L. Pierce and S. Tragoudas. Enhanced Secure Architecture for Joint Action Test Group Systems. *IEEE Transactions on VLSI Systems*, 21(7):1342–1345, 2012.

[17] X. Wang, Y. Zheng, A. Basak and S. Bhunia. IIPS: Infrastructure IP for Secure SoC Design. *IEEE Transaction on Computers*, 2014, ISSN: 0018-9340.