Poster: Steps Toward Automated Proof-Carrying Hardware Intellectual Property (PCHIP) Framework

Mohammad-Mahdi Bidmeshki and Yiorgos Makris
Department of Electrical Engineering, The University of Texas at Dallas
Email: {bidmeshki, yiorgos.makris}@utdallas.edu

***Abstract*—Proof carrying hardware intellectual property (PCHIP) introduces a new framework in which a hardware intellectual property (IP) is accompanied by formal proofs of certain security-related properties, ensuring that the acquired IP is trustworthy and free from hardware Trojans. PCHIP framework adds extra burdensome tasks in the hardware IP development process, hindering its wide acceptance by the hardware design community. In this poster, we review the PCHIP framework and present our efforts toward automating PCHIP, in order to simplify the adoption of PCHIP by hardware IP developers and consumers and, thereby, increase hardware IP trustworthiness.**

I. Introduction

Financial and other market realities have made integrated circuit (IC) design and manufacturing a highly geographically dispersed endeavor. As such, contemporary ICs have also become more vulnerable to inconspicuous modifications resulting in the proliferation of malicious capabilities known as hardware Trojans, and many research efforts have been devoted to preventing and/or detecting hardware Trojan inclusion in various phases of IC design and fabrication. Utilization of previously developed designs in the form of hardware intellectual properties (IPs), in-house or third-party, is commonly practiced in the hardware design flow, in order to enhance time-to-market of the final product. Among various types of hardware IPs, Soft IPs delivered in the form of HDL code, are more susceptible to malicious modifications and hardware Trojan insertion due to their flexibility and the fact that functional testing can by no means reveal the design capabilities exhaustively. Moreover, soft IPs are also widely used in FPGA-based designs. Consequently, hardware Trojans concealed in soft IPs have a significantly wider domain of action, as compared to hardware Trojans which are implanted during the later fabrication stages. Considering this intensified threat, prevention and/or detection of hardware Trojans in soft IPs has become extremely important. Proof carrying hardware IP (PCHIP) achieves this principal objective by utilizing formal methods, and adopting ideas from proof carrying code in software domain [1].

The overall view of PCHIP framework is depicted in Fig. 1. In this framework, along with the HDL code for a design, IP developers are required to develop and deliver another essential piece: formal proofs that the code abides by a set of security properties that are agreed upon by both the IP developer and the IP consumer. These properties do not necessarily impose restrictions on the details of implementation. Rather, they institute a high level boundary of trusted functionality, which prevents misbehavior or unsolicited actions. For example, a security property for a microprocessor IP could be defined as follows: Each instruction is only allowed to access memory locations which are specified in the corresponding fields of its op-code [2]. This property prevents stealthy information leakage. However, it does not restrict the details of instruction implementation. As another example, security properties might impose restrictions on the flow of information in a design [3], [4], [5] to avoid propagation of sensitive information to unauthorized sites within the chip and eventual leakage.

Mechanized proof development and checking requires a theorem proving language and proof checking environment, such as Coq [6] and CoqIDE, respectively. Therefore, in order to be applicable and leverage the rich collection of hardware IPs developed in HDLs such as Verilog and VHDL, PCHIP defines conversion rules from HDLs to a Coq representation. Consequently, PCHIP does not intervene in the current hardware IP design and test methodology, as is the case when introducing a new formal HDL. Rather, it adds extra steps in parallel to the current design methodology, namely conversion to Coq representation, stating security properties as theorems in Coq, constructing proofs for such theorems based on the hardware design and delivering those proofs along with the HDL code to the IP consumer. PCHIP does not inflict IP consumers with much extra burden. Along with the IP developers, they need to agree on the desired security properties. The onerous task of proof development is, then, the responsibility of the IP developers.

PCHIP is a very promising framework, can be employed in various types of hardware designs, and can be adaptively modified to fit the requirements of the design and IP consumer. For example, the applicability of PCHIP to ensure the trustworthiness of microprocessor IPs in terms of instruction set architecture (ISA) [2], and cryptographic cores [3], [4] in terms of information flow policies has been successfully demonstrated. However, the broad adoption of PCHIP faces a few challenges. First, developing security properties is not straightforward. These properties are usually specific to each design, and cannot be reused for others. Second, converting HDL code to a formal representation, such as the Coq language used in PCHIP, and developing proofs for security properties, requires additional knowledge of formal methods, theorem proving environments, and proof writing. Even for someone who has this expertise, the process is tedious and time consuming, making the barrier to entrance rather high for IP developers. Evidently, automating the PCHIP framework to the extent possible, targeted in this work, could make it more appealing and could help in its broader utilization, leading to lower risk in hardware IP acquisition.

II. *VeriCoq*: Automated Verilog to Coq Converter

Towards automating PCHIP framework, we introduced *VeriCoq* [7], a Verilog-to-Coq converter based on the rules developed in the PCHIP framework. *VeriCoq* supports most of

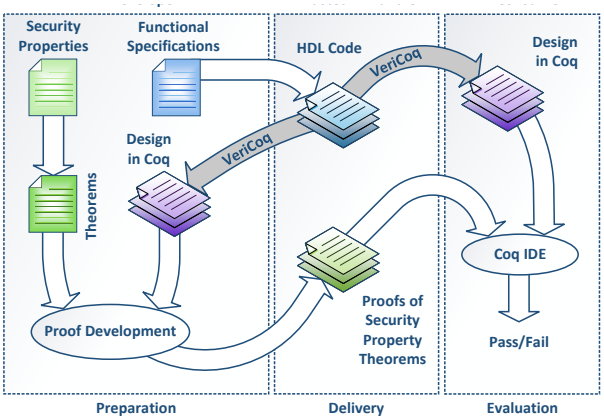

Fig. 1. PCHIP framework

the synthesizable Verilog constructs and converts parameters, arrays, module hierarchy, and module instantiations effectively to their Coq representation. While automating the entire PCHIP framework is a much broader endeavor and may not be completely feasible, given the strenuous details of proof construction, *VeriCoq* is a fundamental step towards this end. Crucially, it not only automates the conversion process, but also makes proof construction by IP developers and proof checking by IP consumers less perplexing, since both can rely on the common Coq representation of the Verilog code, which is now automatically generated by *VeriCoq*, as indicated by shaded arrows in Fig. 1. It is extensively applicable to convert various designs, such as microprocessor IPs, to their corresponding Coq representation [2].

III. *VeriCoq-IFT*: Automated PCHIP Framework for Information Flow Policies

In cryptographic hardware, the emphasis goes on the flow of information in the design, not the actual functionality and one goal is to prevent sensitive information leakage. For this purpose, PCHIP introduces a framework with information flow tracking capabilities [3], [4], which assigns a sensitivity (secrecy) level tag to the signals in the design. By tracking those signals through the design, this methodology ensures that no sensitive information reaches the outputs without going through proper sensitivity reducing (declassifying) operations. We developed *VeriCoq-IFT* [5], shown in Fig. 2, which redefines this base framework toward automation of the entire process, and is a step toward a *fully automated* PCHIP framework, focusing on information flow policies. In addition to automating the conversion of the HDL code to the Coq formal representation, *VeriCoq-IFT* automatically generates security property theorems to ensure the abidance of the design by the information flow policies, constructs proofs for such theorems and checks their validity for the design with minimal user intervention. *VeriCoq-IFT* gathers required user inputs through special comments inside the HDL design. Users only need to provide such information in the HDL code and *VeriCoq-IFT* performs the rest of the work automatically.

We successfully tested this automated framework by utilizing it to evaluate the trustworthiness of several genuine and Trojan infested DES and AES cryptographic cores. Specifically, we evaluated two implementations of DES algorithm provided in [8]. The first DES core is designed for area efficiency and implements only a single round of encryption. Several iterations are then invoked to perform the entire

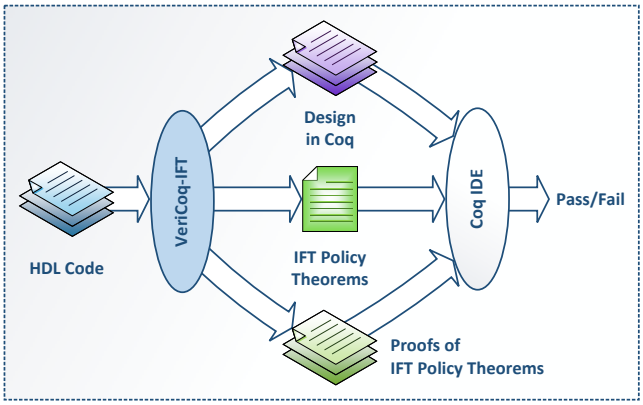

Fig. 2. Automated PCHIP framework for information flow policies

encryption. Automatic evaluation of this core using *VeriCoq-IFT* reveals a design flaw capable of leaking sensitive information. We also employed *VeriCoq-IFT* to evaluate a genuine implementation of AES and various Trojan infested AES designs provided in [9]. While the proof of security theorems passed for the genuine AES design, they were not valid for Trojan infested AES cores. This means that *VeriCoq-IFT* could successfully capture hardware Trojans capable of leaking sensitive information.

IV. Future Work

In our ongoing research, we plan to expand the capabilities of *VeriCoq-IFT* and *VeriCoq* by adding support for a few other Verilog constructs which are not supported in the current version. We also plan to explore more automation possibilities in the general PCHIP framework for various hardware designs like microprocessor IPs and communication cores. Moreover, we are working on a hierarchy preserving conversion to the Coq representation which reduces the burden of proof construction and enables the development of hybrid libraries containing the HDL code and various lemmas, to be used for higher level designs and proofs. The expansion of the automated PCHIP framework will help its wide adoption in hardware design community, resulting in a more secure and trustworthy third party IP acquisition protocol.

References

[1] G. C. Necula, "Proof-carrying code," in *Proc. Symp. Principles of Programming Languages*. ACM, 1997, pp. 106–119.

[2] Y. Jin and Y. Makris, "A proof-carrying based framework for trusted microprocessor IP," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2013, pp. 824–829.

[3] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *Proc. IEEE VLSI Test Symposium*, 2012, pp. 252–257.

[4] Y. Jin, B. Yang, and Y. Makris, "Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing," in *Int. Symp. Hardware-Oriented Security and Trust*. IEEE, 2013, pp. 99–106.

[5] M.-M. Bidmeshki and Y. Makris, "Toward automatic proof generation for information flow policies in third-party hardware IP," in *Int. Symp. Hardware-Oriented Security and Trust*. IEEE, 2015, pp. 163–168.

[6] INRIA. (2014, Oct.) The Coq proof assistant. [Online]. Available: http://coq.inria.fr/

[7] M.-M. Bidmeshki and Y. Makris, "VeriCoq: A Verilog-to-Coq converter for proof-carrying hardware automation," in *Int. Symp. Circuits and Systems*. IEEE, 2015.

[8] OpenCores. [Online]. Available: http://opencores.org/

[9] Trust-Hub. [Online]. Available: https://www.trust-hub.org/