# Hierarchy-Preserving Formal Verification Methods for Pre-Silicon Security Assurance

Xiaolong Guo, Raj Gautam Dutta, and Yier Jin
Department of Electrical and Computer Engineering, University of Central Florida
guoxiaolong@knights.ucf.edu, rajgautamdutta@knights.ucf.edu, yier.jin@eecs.ucf.edu

***Abstract*—The wide usage of hardware intellectual property (IP) cores from untrusted vendors has raised security concerns in the integrated circuit (IC) industry. Existing testing methods are designed to validate the functionality of the hardware IP cores. These methods often fall short in detecting unspecified (often malicious) logic. Formal methods, on the other hand, can help eliminate hardware Trojans and/or design backdoors by formally proving security properties on soft IP cores despite the high proof development cost. To alleviate the computation burden, we propose a new hierarchy-preserving formal verification (HiFV) framework for circuit trust evaluation at the pre-silicon stage. This framework is derived from the Proof-Carrying Hardware (PCH) and is dedicated for security property verification of System-on-Chip (SoC) platforms, where third-party soft IPs are integrated as sub-modules. The key novelty lies in the improvement of the proof construction process of the previously developed security property verification framework, so that the framework can support building theorem proofs in a hierarchical way. We assume a trusted third-party verification house exists, which can use the proposed framework for security theorem construction and proof writing. The applicability of the proposed framework is demonstrated by formally verifying the memory integrity property on an 8051 microprocessor whose sub-modules were treated as untrusted third-party IPs.**

## I. Introduction

The improvement of manufacturing technology makes it possible to integrate billions of transistors in one chip, but it increases the difficulty of designing such large-scale circuits. To alleviate the workload of the circuit designers and to shorten the time-to-market (TTM), the hierarchical design methodology has been widely used in the IC industry with the leading example being system-on-chip (SoC) platforms. System integrators use IP cores from trusted and untrusted third-party vendors to build SoCs in a hierarchical structure.

The prevailing usage of third-party soft IP cores in SoC designs raises security concerns as current IP core verification methods focus on IP functionality rather than IP trustworthiness. Moreover, lack of regulation in the IP transaction market adds to the predicament of the SoC designers and forces them to perform verification and validation of IPs themselves. To help SoC designers in IP verification, various methods have been developed, which leverage enhanced functional testing and/or perform probability analysis of internal nodes for IP core trust evaluation and malicious logic detection [1], [2]. However, these methods were easily bypassed by sophisticated hardware Trojans [3]–[5]. Formal methods were also introduced for IP core trust evaluation [1], [6]–[10]. Among all the proposed formal methods, proof-carrying hardware (PCH), which originated from proof-carrying code (PCC), emerged as one of the most prevalent methods for certifying the absence of malicious logic in soft IP cores and reconfigurable logic [6]–[10]. In the PCH approach, synthesizable register-transfer level (RTL) code of IP core and informal security properties were first represented in *Gallina* - the internal functional programming language of the Coq proof assistant [11]. Then, Hoare-logic style reasoning was used to prove the correctness of the RTL code in the Coq platform.

However, previous PCH-based formal verification is mostly effective in evaluating trustworthiness of individual IP cores [8]–[10]. These methods cannot be directly applied for verifying security properties on hierarchical designs such as SoCs. It can only be used by flattening the SoC design and treating the design as one IP core. This approach will incur high computational effort and reduce the flexibility of proof construction in PCH based methods. To solve this problem, we developed a hierarchy-preserving formal verification (HiFV) framework for ensuring trust in SoCs designed using third-party IPs. Compared to the previous PCH frameworks [8]–[10], the proposed scheme can be used to prove security properties on hierarchical designs, thereby eliminating the effort required to verify a flattened design. Also, the process for proving security properties is improved to make the proposed framework scalable to large designs. The proof construction process of this paper requires dividing the security properties into sub-properties in such a way that each sub-property corresponds to an IP module of the SoC. The main contributions of this paper are:

- A hierarchy-preserving formal verification (HiFV) framework is developed, which is scalable to large-scale designs and helps to reduce the effort required for proving security properties;
- Our method enables proof reuse, which can significantly reduce the workload for SoC security property verification. Proofs for individual IP cores can be stored in a library and accessed during the security verification process;
- A distributed proof construction approach is developed, which reduces the effort required for modifying the proof when certain IP modules are added, deleted, or modified in a SoC platform.

The rest of the paper is organized as follows: In section II, we provide the background on proof-carrying code and its hardware variant, proof-carrying hardware. In section III, we introduce the threat model, explain our hierarchy-preserving formal verification framework, and elaborate the proof construction procedure. Section IV presents demonstrations of the proposed framework in proving a sample memory integrity property on an 8051 microprocessor. Final conclusions are drawn in Section V.

## II. Proof-Carrying Hardware

Various methods have been proposed in the software domain to validate the trustworthiness and genuineness of software programs. These methods protect computer systems from untrusted software programs. Most of these methods place the burden on software consumers to verify the code. However, Proof-Carrying Code (PCC) switches the verification burden to software providers (software vendors/developers) [12].

A similar mechanism, called Proof-Carrying Hardware (PCH), was used in the hardware domain to protect third-party soft IPs [8]–[10]. The PCH framework certifies that soft IPs are trusted if certain carefully specified security properties hold. In this approach, the IP consumer provides functional specifications and security constraints to the IP vendor. Upon receiving the request, the IP vendor develops the RTL code using hardware description languages (HDLs). Before proving the trustworthiness of the RTL code with respect to the formally specified security properties in Coq, the IP vendor needs to perform semantic translation of the HDL code and informal security properties into *Gallina*. After the proof has been constructed, the IP vendor provides the IP consumer with the RTL code, formalized security theorems of security properties, and proofs of security theorems. The IP consumers also translate the design and security properties to *Gallina*. Then, the proof checker in Coq is used to automatically validate the proof of security theorems on the translated code. Correspondingly, the PCH and its applications were introduced in detail in [13], where the use of theorem proving methods for providing high level protection of IP cores is demonstrated.

## III. SoC Verification Procedure

In this section, we outline the threat model and the assumptions used in designing our framework. We also introduce our HiFV framework and explain its working procedure.

### *A. Threat Model and Trusted Verification House*

The HiFV framework is developed to prove the presence/absence of malicious logic inserted by an adversary at the design stage of the supply chain. We assume that there is a rogue agent in the third-party IP design house who has access to the HDL code and can insert a hardware Trojan or backdoor in the design. Such a Trojan can be triggered either by a counter at a predetermined time, by an input vector, or under certain physical conditions. Upon activation it can leak sensitive information from the chip, modify functionality, or cause a denial-of-service to the hardware.

We also assume that our framework can be used by a trusted third-party verification house to guarantee the security of soft IPs for the set of informal security properties, obtained from the SoC integrator. We also assume that the SoC integrator and the trusted third-party verification house use the same platform (such as Coq) to prove and validate the SoC design. Using our scheme, proofs can be constructed for formal security theorems derived from informal security properties. The existence of proofs for the security theorems indicates the absence of Trojans in the design whereas non-existence of a proof indicates the presence of malicious logic. Note that the framework may not provide protection to an IP from Trojans that do not violate the defined security properties.

### *B. Hierarchy-preserving Formal Verification (HiFV)*

Previously proposed PCH frameworks treat the whole circuit design as one module and prove security properties on them [8]–[10], [14]. In PCH, the entire design is first flattened before translating the HDL code of the design into the formal language and proving it with respect to formal security theorems. Design flattening increases the complexity of translating HDL code into *Gallina*. It also adds to the risk of introducing errors during the code conversion process. Due to flattening, a verification expert has to go through the entire design in order to construct proofs of security theorems, which significantly increases the workload for design verification. Also, any updates to the HDL code will significantly change the proof for the same security property. Moreover, the PCH framework prevents proof reuse, i.e., proofs constructed for one design cannot be used in another design even though the same IP modules are used. All of these limitations prohibit a wide usage of the PCH framework in modern SoC designs.

To overcome these limitations, we developed the Hierarchy-preserving Formal Verification (HiFV) framework for verifying security properties on SoC designs. The HiFV framework is an extension of the PCH framework. In the HiFV framework, the design hierarchy of the SoC is preserved and a distributed approach is developed for constructing proofs of security properties. In the distributed approach, security properties are divided into sub-properties in such a way that each sub-property corresponds to an IP module of the SoC. Proofs are then constructed for these sub-properties and the security property for the SoC design is proven through the integration of all proofs from sub-properties. Similar to PCH, the HiFV framework requires semantic translation of the HDL code and informal security properties to *Gallina*. For proving the trustworthiness of the HDL code of the SoC, Hoare-logic style reasoning is used. The whole HiFV framework is carried out in Coq. The advantage of this method is four-fold:

- The HiFV framework is scalable to large-scale and complex designs such as SoCs;
- The distributed approach of proof construction reduces the effort required to modify the proof when the HDL code of the IP cores in the SoC changes;
- The distributed approach allows the collaboration of multiple verification experts to prove security properties on SoCs;
- The HiFV framework allows for proof reuse, thereby enabling verification experts to use existing proofs of IP modules to prove security properties on different SoCs where the same IPs are used.

### *C. Working Procedure of the HiFV Framework*

SoCs are made of several IP cores, which are provided by either trusted or untrusted third-party vendors. However, integration of IP cores from different vendors makes the SoC vulnerable to malicious attacks. The HiFV framework of this paper can be used by a trusted third-party verification house to determine the trustworthiness of SoCs.

Three entities are involved in the SoC verification process.

- **IP Vendors (IPV):** IP vendors design and sell soft, firm, or hard IP cores. Such a vendor specializes in designing

IP cores for specific device families. In our framework, we consider untrusted IPVs who provide soft IP cores to IP consumers.

- **IP Consumer (IPC):** IP consumers in this framework are SoC integrators. To design the SoC, they integrate IPs from multiple IP vendors. In our framework, we assume that SoC integrators develop the set of informal security properties and validate the proofs of these properties (given by the trusted third-party verification house) using an automated proof checker (such as Coq).
- **Trusted Third Party (TTP):** The TTP in our framework is the verification house [15]. Upon receiving the HDL code of the design and the set of informal security properties from the SoC designer, the TTP uses the HiFV framework to prove the security properties (in Coq) on the design. Subsequently, they provide the SoC integrator with the translated code of the HDL, the formal security theorems, and their proofs.

The working procedure of the proposed framework is shown in Figure 1 which can be divided into five phases.

**Phase I: Functional Specifications.** In the first phase, the IPC (SoC designer) provides functional specifications to the IPVs. Based on the request, vendors provide HDL codes of the IPs to the SoC integrator.

**Phase II: Security Properties and SoC Design.** In the second phase, the IPC provides the informal set of security properties and the HDL code of the SoC to the TTP (verification house).

**Phase III: Translation of HDL Code and Security Properties to *Gallina*.** In this phase, the TTP translates the syntax and semantics of the HDL code of the SoC into *Gallina*. This is referred to as *Coq equivalent code* in Figure 1. Translating the security properties, expressed in natural languages, to *Gallina* give the desired formal security theorems. These translations make the security verification of SoCs possible on the Coq proof assistant.

**Phased IV: Proof Generation.** In this phase, the TTP use the distributed proof construction approach to prove the formal security theorems, which are defined at the top module of the SoC design hierarchy. Before proving this theorem, the TTP divide the theorem into lemmas. This division is made by analyzing how the behaviors of sub-modules would be restricted by the security theorems. We assume that the verification expert has a complete understanding of the SoC design hierarchy as well as all sub-modules. For each of these sub-modules, lemmas are developed. A verification expert first constructs proofs for these lemmas and then integrates these proofs to obtain the proof of the security theorem on the entire SoC design. In each of these proof construction stages, Hoare-logic is used to prove the trustworthiness of the HDL code. This approach to proof generation enables proof reuse, quick correction and modification, and the scalability of the framework when applied to complex designs.

**Phase V: Proof Validation.** After proving the security theorems, the verification house provides the SoC integrator with the translated HDL code, formal security theorems, and proofs of these theorems. The SoC designer then validates the proofs using the proof checker. The presence of malicious logic in the design is revealed if the proof fails. Otherwise, SoC integrators are assured that the SoC with untrusted IP cores is trustworthy.

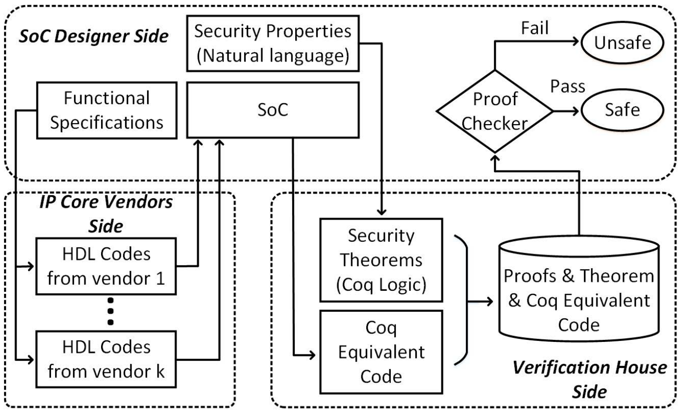

Figure 1: Working Procedure of the HiFV Framework.

In our framework, proof construction is the most time consuming step. Thus, the task of proving security theorems on the design is delegated by the IP consumer to the TTP verification house. This reduces the workload of IP consumers who are only responsible for informal security property development and proof checking.

## *D. Semantic Translation*

The semantic translation method of this paper is based on the *formal HDL* developed in [10]. In this method, we convert the HDL code of the SoC to *Gallina*, the underlying formal language of the Coq proof assistant. *Gallina* is based on dependently typed lambda calculus and it defines both types and terms in the same syntactical structure. During the translation process, syntax and semantics of the HDL are translated to *Gallina* using the *formal HDL*. In addition, *interface* and *module* are incorporated in the formal HDL to preserve the design hierarchy of the SoC. The procedure of semantic translation including code translation, interface development and module representation are briefly introduced below.

**Translation of Syntax and Semantics of HDL.** Before building the formal model for the SoC system, the syntax and semantics should be defined and then shared by any parties who need to design or check the proof. The *formal HDL* is used which can represent basic circuit units, combinational logic and sequential logic.

**Interface.** To make distributed proof construction applicable on hierarchical designs, an *interface* is developed in the HiFV framework. It makes the verification process flexible and efficient for the TTP (verification house). To define the *interface*, information about each IP and its corresponding I/O are needed, such as the name, number, data type, etc. By using the *interface*, the management of the plenty of formal modules would be much easier in the verification house side. The structure of the *interface* is shown in Figure 2 which helps one IP module access other modules.

**Module.** The functionality of Coq enables representation of IP modules of SoC in a hierarchical form.

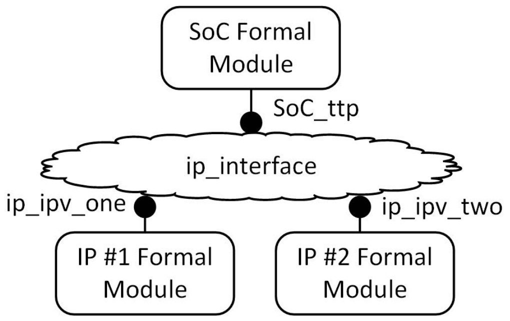

Figure 2: Structure of the SoC with Interface

### E. Distributed Proof Construction

The distributed proof construction process follows Hoare-logic style reasoning, where the trustworthiness of the SoC formal HDL code is determined by ensuring that the code operates within the constraints of the pre-condition and the post-condition. The pre-condition of the formal HDL code is the initial configuration of the design and the post-condition is the security theorem. Although the PCH framework of [8], [14] provides a high-level of security assurance to soft IP cores, it suffers from the problem of scalability. The key issue limiting scalability of the framework to SoCs lies in proof construction. In PCH, proof construction is carried out on a flattened design, which increases the effort required for proof generation and modification. To overcome this limitation, we develop a distributed proof construction approach, which is dedicated for SoC designs with hierarchical structures. This approach makes the HiFV framework scalable by reducing the time required for proof construction, proof correction, and proof modification.

In the HiFV framework, the translated HDL code of the SoC, formal security theorems, and the initial configuration of the design is represented as a Hoare Triple (Eq.1).

$$(\phi) CoqEquivalentCode\_SoC(\psi) \tag{1}$$

In this equation, $\phi$ is the pre-condition corresponding to the initial configuration of the design. The translated HDL code of the SoC design hierarchy in *Gallina* is given by $CoqEquivalentCode\_SoC$. In the process of translation, modules in the SoC HDL code, which correspond to IPs from different vendors, are also translated. The post-condition is given by $\psi$ which represents the formal security theorem.

The security theorem is divided into lemmas (Eq. 2), which are post-conditions for individual IP modules. In Eq. 2, post-condition for IPs (lemmas) are represented as $\psi_i$ $(1 \leq i \leq n)$, *n = maximum number of IP modules required to prove the security theorem* and $\psi$ is the security theorem. These lemmas correspond to those IP modules that are required to satisfy the security theorem.

$$\psi := \psi_1 \wedge \psi_2 \cdots \wedge \psi_n \tag{2}$$

Similarly, the pre-condition of the SoC design $(\phi)$ and the translated HDL code of the SoC design $(CoqEquivalentCode\_SoC)$ are divided according to Eq.3 and Eq.4. Here, $(\phi_i)$ and $(CoqEquivalentCode\_IPmodule\_i)$ $(1 \leq i \leq n)$ represent the pre-conditions and translated HDL code of each IP module of the SoC respectively.

$$\phi := \phi_1 \wedge \phi_2 \cdots \wedge \phi_n \tag{3}$$

$$\begin{aligned} &CoqEquivalentCode\_SoC := \\ &CoqEquivalentCode\_IPmodule\_1 \\ &\wedge CoqEquivalentCode\_IPmodule\_2 \ldots \\ &\wedge CoqEquivalentCode\_IPmodule\_n \end{aligned} \tag{4}$$

For proving the trustworthiness of IP modules and the SoC, Hoare-logic style reasoning is used. In this approach, the translated HDL code of individual IPs $(CoqEquivalentCode\_IPmodule\_i)$, the pre-conditions $(\phi_i)$ of each module, and the post-conditions $(\psi_i)$ are represented as a Hoare Triple (Eq.5). The HDL code of the IP core is certified to be trustworthy only if it satisfies the pre-condition and the post-condition. When all the modules of IP cores satisfy the post-conditions (lemmas), we can state that the security theorem is proven for the SoC design.

$$(\phi_i) CoqEquivalentCode\_IPmodule\_i(\psi_i) \tag{5}$$

The distributed approach of proof construction also enables proof reuse. After certifying the trustworthiness of each IP core of the SoC, the proofs can be stored in a library and accessed by the TTP verification house for verification of other SoC designs in which the same IP modules are used and similar security properties are applied. In this way our framework further reduces the time for verifying complex designs.

## IV. Demonstration

In this section, we demonstrate the working procedure of the proposed HiFV framework where the 8051 microprocessor serves as the SoC platform under verification since the design is constructed on top of various IP modules. The block diagram of the microprocessor is shown in Figure 3 and the source VHDL code is available at [16]. We treat all sub-modules of the microprocessor as untrusted third-party IPs. In our demonstration, security properties for memory integrity are also developed to verify the trustworthiness of the target design. Compared to PCH, the overall workload for proof writing has been significantly reduced through the HiFV framework[1].

In our framework, no modifications are required on the original code throughout the verification procedure. As a result, there is no performance overhead on the computer system under security verification.

### A. Security Property Construction

In this demonstration, we consider attacks which can manipulate the memory unit (special function registers) of the

[1]While it is difficult to quantify the time used for proof writing since this procedure is still performed manually, the elimination of HDL code flattening and the reuse of proofs for submodules clearly show the workload reduction in the new HiFV framework.

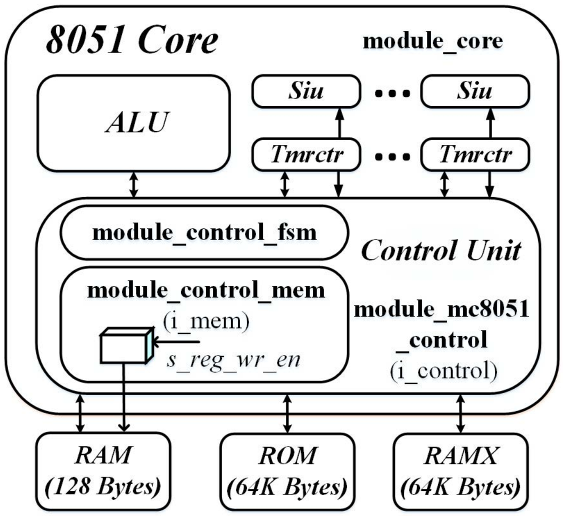

Figure 3: Hierarchical Structure of the 8051 Microprocessor

8051. Therefore, the security property is developed to detect these type of attacks. The informal security property, given by the SoC designer to the TTP can be described as, *Contents of the special function registers (SFR) of the 8051 core will not be changed if the executing instructions are not allowed to modify the register's contents*. Since there are a large set of instructions which are not allowed to change the contents of the SFR, the developed security property should apply to all of them.

## B. Semantic Translation of HDL Code

To make the demonstration process easy to follow, we only present the operation details of the control unit (the corresponding formal module is named *module_mc8051_control*) in the 8051 microprocessor. According to the hierarchical structure of the 8051 microprocessor, given in Figure 3, there are two sub-modules, *module_control_mem* and *module_control_fsm*, of the module *module_mc8051_control*. We consider the scenario where the module *module_mc8051_core* loads the sub-module, *module_mc8051_control*. This sub-module oversees the functionality such as writing to special function registers and addressable memory locations.

Based on the developed *semantic translation rules*, *interface*, and *module*, we define *interface* for each module as: *ip_control_fsm* for *module_control_fsm*, *ip_control_mem* for *module_control_mem*, and the *ip_mc8051_control* for *module_mc8051_control* as shown in the following code.

```
Inductive ip_interface :=
 | ip_control_fsm : bus->bus->bus->...->ip_interface
 | ip_control_mem : bus->bus->bus->...->ip_interface
 | ip_mc8051_control : ...->ip_interface->ip_interface
 ...
 | ip_mc8051_core : ip_interface->ip_interface.
```

The description of the formal module *module_mc8051_control* which implements the interface *ip_mc8051_control* is shown below. In this module, the circuits detail is described by using the keyword *Fixpoint*. Through using the keyword *Axiom*, the *control_core_rtl* is generated as the instantiation of the circuit in the module *module_mc8051_control*. The corresponding information of these modules can be found in Figure 3. After that, the instantiation can be used as precondition for proving the security theorem *Security_ControlUnit*. At this point the proof of *Security_ControlUnit* can be integrated to prove the security theorem *Security_SoC*.

```
Module Type module_mc8051_control.
 Declare Module i_mem : module_control_mem.
 Declare Module i_fsm : module_control_fsm.
 Parameters s_pc_inc_en s_regs_wr_en ... all_th1_i : bus.
 ...
 Fixpoint module_inst (m:ip_interface) (t:nat) :=
 match m with
  | (ip_control_mem pc_o rom_data_i ram_data_o ...
          lcase13 lblock13)=> i_mem.module_inst m t
  | (ip_control_fsm state_i help_i...)=>i_fsm.module_inst m t
  | (ip_mc8051_control s_pc_inc_en ...) =>
       (update ((upd_expr acc_o (econb acc))) t) ...
  ...
  | (ip_mc8051_core mymodule1) => True
 end.
 ...
 Axiom control_core_rtl : forall (t:nat),
 module_inst (ip_mc8051_control s_regs_wr_en ...) t.
 ...
 Theorem Security_ControlUnit :
 forall (t : nat) (dt1 : bus_value) ,
  i_mem.state t = FETCH ->
  i_mem.rom_data_i t = ADD_A_DATA ->
  i_mem.rom_data_i (S t) = dt1 ->
  i_mem.reset t = lo::nil->
  i_mem.reset (S t) = lo::nil ->
  i_mem.ie t = lo::lo::lo::lo::lo::lo::lo::lo::nil ->
  i_mem.ie (S t) = lo::lo::lo::lo::lo::lo::lo::lo::nil ->
  i_mem.s_intpre2 t = lo::nil ->
  i_mem.s_intpre2 (S t) = lo::nil ->
  (bv_eq (i_mem.s_regs_wr_en t) (hi::lo::lo::nil) = lo/\
  bv_eq (i_mem.s_regs_wr_en t) (hi::lo::hi::nil)=lo)/\
  (bv_eq (i_mem.s_regs_wr_en (S t)) (hi::lo::lo::nil) = lo/\
  bv_eq (i_mem.s_regs_wr_en (S t)) (hi::lo::hi::nil)=lo).
 Proof.
 ...
 Qed.
End module_mc8051_control.
```

## C. Distributed Proof Construction

For the security assurance of the SoC, the security properties on a large set of instructions need to be proven. Due to page limits, only the ADD instruction is shown as an example in this section. For this specific instruction, the security property is refined to *The contents stored in the special function registers should not be modified during the execution of the immediate add instruction - ADD A, #immediate*. The formal security theorems are then derived from the informal security property and are verified with respect to the instruction *ADD A, #immediate*.

An analysis of the 8051 microprocessor structure shows that the permission to write on the special function registers depends on the enable signals *s_regs_wr_en* of the IP module named *module_control_mem*. The special function registers are updated when the following conditions are satisfied: the control signal *s_regs_wr_en* equals 100 or 101. Then the formal theorem (*Security_SoC*) for the security property is constructed as following:

```
Module Type module_core.
 Declare Module i_control : module_mc8051_control.
 ...
 Theorem Security_SoC :
 forall (t : nat) (dt1 : bus_value) ,
 i_control.i_mem.state t = FETCH ->
 i_control.i_mem.rom_data_i t = ADD_A_DATA ->
 i_control.i_mem.rom_data_i (S t) = dt1 ->
 i_control.i_mem.reset t = lo::nil->
 i_control.i_mem.reset (S t) = lo::nil ->
 i_control.i_mem.ie t = lo::lo::lo::lo::lo::lo::lo::lo::nil->
 i_control.i_mem.ie (S t) = lo::lo::lo::lo::
                      lo::lo::lo::lo::nil ->
 i_control.i_mem.s_intpre2 t = lo::nil ->
 i_control.i_mem.s_intpre2 (S t) = lo::nil ->
 (bv_eq (i_control.i_mem.s_regs_wr_en t) (hi::lo::lo::nil)=lo/\
 bv_eq (i_control.i_mem.s_regs_wr_en t) (hi::lo::hi::nil)=lo)/\
 (bv_eq (i_control.i_mem.s_regs_wr_en (S t))
                    (hi::lo::lo::nil) = lo/\
 bv_eq (i_control.i_mem.s_regs_wr_en (S t))
                    (hi::lo::hi::nil)=lo).
 Proof.
 intros.
 apply i_control.Security_ControlUnit with (t:=t0)(dt1:=dt1).
 ...
 Qed.
End module_core.
```

The example code calls entities and instantiations within the sub-module: *i_control.i_mem.s_regs_wr_en* indicates the variable *s_regs_wr_en* in the sub-sub-module *i_mem*, which is inside another sub-module *i_control*. Also, pre-conditions of the formal security theorem can be explicitly specified as follows:
1) *forall (t : nat)* means that the execution can begin at any time while *(S t)* stands for the *t+1*.
2) *state t = FETCH* indicates that the function is executed from the initial state - FETCH at the *t* clock-cycle;
3) *reset t = lo::nil*, *ie t = lo::...* and *s_intpre2 t = lo::nil* imply that our framework does not handle reset and interrupt during the SoC operation;
4) *i_control.i_mem.rom_data_i t = ADD_A_DATA* implies that *rom_data_i* in the sub-module *i_mem* takes the op-code of the *ADD A, #immediate* instruction at *t* clock cycle;
5) *i_control.i_mem.rom_data_i (S t) = dt1* indicates that *rom_data_i* can take any input at the *t+1* clock cycle.

The variable *dt1* of the formal theorem indicates that the input can be any binary code. The function *bv_eq* compares two binary codes and returns the result *lo* when there is a match between the codes and *hi* otherwise.

In this approach, the theorem in the top module (such as the module *module_core*) accesses the lemmas (or theorems) in the lower level (such as the *module_mc8051_control*) during the proof construction process. At the lowest level of the hierarchy, the lemma *Security_ControlUnit* is proven for the instruction *ADD A, #immediate* with respect to the pre- and post-condition of the lemma of the Control Unit module. In the top module *module_core*, the theorem already proven at the sub-module *module_mc8051_control* is reused by accessing the instantiation *i_control*. With the same procedure, contents in the other sub-module *module_control_mem* can also be called by using the corresponding instantiation *i_mem*.

The proof was successfully constructed for the security theorem for the Control Unit IP of the 8051 microprocessor. Consequently, we can conclude that the SoC is secure from any malicious memory manipulation attack within the domain of the defined security property.

## V. Conclusion

In this paper, we have extended the previously developed PCH framework into the SoC design flow and largely simplified the process for proving security properties through a hierarchical proof construction procedure. To reduce the workload for circuit verification, the proof of the security properties for individual IPs can be encapsulated and reused in proving security properties at the SoC level. Also, in the hierarchical framework, the amount of updates that need to be done to existing proofs when SoC designs are modified is significantly lowered. The developed HiFV framework paves the way for large-scale circuit design security verification.

## VI. Acknowledgement

This work was supported in part by the National Science Foundation grants CNS-1319105.

## References

[1] M. Banga and M. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 56–59.
[2] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, 2013, pp. 697–708.
[3] D. Sullivan, J. Biggers, G. Zhu, S. Zhang, and Y. Jin, "FIGHT-metric: Functional identification of gate-level hardware trustworthiness," in *Design Automation Conference (DAC)*, 2014.
[4] N. Tsoutsos, C. Konstantinou, and M. Maniatakos, "Advanced techniques for designing stealthy hardware trojans," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, 2014.
[5] M. Rudra, N. Daniel, V. Nagoorkar, and D. Hoe, "Designing stealthy trojans with sequential logic: A stream cipher case study," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, 2014.
[6] S. Drzevitzky, U. Kastens, and M. Platzner, "Proof-carrying hardware: Towards runtime verification of reconfigurable modules," in *International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 189–194.
[7] S. Drzevitzky and M. Platzner, "Achieving hardware security for reconfigurable systems on chip by a proof-carrying code approach," in *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, 2011, pp. 1–8.
[8] E. Love, Y. Jin, and Y. Makris, "Proof-carrying hardware intellectual property: A pathway to trusted module acquisition," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 25–40, 2012.
[9] Y. Jin, B. Yang, and Y. Makris, "Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 99–106.
[10] Y. Jin and Y. Makris, "A proof-carrying based framework for trusted microprocessor IP," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 824–829.
[11] INRIA, "The coq proof assistant," 2010, http://coq.inria.fr/.
[12] G. C. Necula, "Proof-carrying code," in *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1997, pp. 106–119.
[13] X. Guo, R. G. Dutta, Y. Jin, F. Farahmandi, and P. Mishra, "Pre-silicon security verification and validation: A formal perspective," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '15, 2015, pp. 1–6.
[14] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *IEEE 30th VLSI Test Symposium (VTS)*, 2012, pp. 252–257.
[15] Oski Technology: http://www.oskitechnology.com/.
[16] Oregano Systems, "8051 IP core," http://www.oreganosystems.at/?page_id=96.