Chapter 4
IP Trust: The Problem and Design/Validation-Based Solution

Raj Gautam Dutta, Xiaolong Guo and Yier Jin

4.1 Introduction

A rapidly growing third-party intellectual property (IP) market provides IP consumers with more options for designing electronic systems. It also reduces the development time and expertise needed to compete in a market where profit windows are very narrow. However, one key issue that has been neglected is the security of electronic systems built upon third-party IP cores. Historically, IP consumers have focused on IP functionality and performance than security. The prejudice against the development of robust security policies is reflected in the IP design flow (see Fig. 4.1), where IP core specification usually includes functionality and performance measurements.

This lack of security assurance on third-party IPs is a major threat for the semiconductor industry. For example, a large number of side-channel-based attacks have been reported, which extract sensitive information from systems that were purportedly mathematically unbreakable [1–7]. The emergence of hardware Trojans (malicious logic) embedded in third-party IP cores has largely re-shaped the IP transaction market and there are currently no comprehensive detection schemes for identifying these Trojans. Some Trojan detection methods such as side-channel fingerprinting combined with statistical analysis have shown promising results, but most of the post-silicon stage Trojan detection methods rely on golden models which may not be available given the existence of untrusted IP cores.

R.G. Dutta · X. Guo · Y. Jin (✉)
University of Central Florida, Orlando, FL, USA
e-mail: rajgautamdutta@knights.ucf.edu

X. Guo
e-mail: guoxiaolong@knights.ucf.edu

Y. Jin
e-mail: yier.jin@eecs.ucf.edu

S. Bhunia et al. (eds.), *Fundamentals of IP and SoC Security*,
DOI 10.1007/978-3-319-50057-7_4

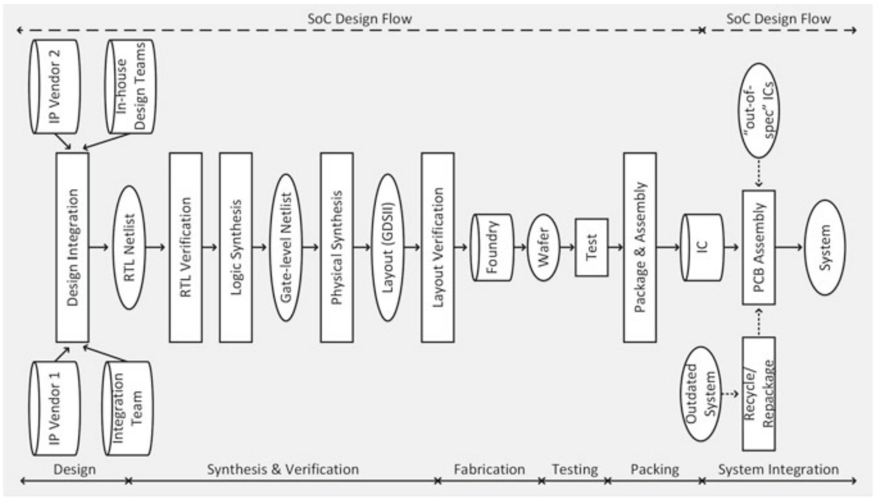

Fig. 4.1 Semiconductor supply chain: IC design flow

Upon the request for trusted IP cores, various IP protection and certification methods at the pre-silicon stage have been recently developed. In this chapter, most of these approaches will be introduced including hardware locking/encryption, FPGA bitstream protection, theorem proving, and equivalence checking.

The rest of the chapter is organized as follows: Sect. 4.2 discusses various hardware locking/encryption methods for preventing various threats to IP cores. For a better explanation, we divide these methods into three categories (i) combinational logic locking/encryption, (ii) finite state machine locking/encryption, and (iii) locking using reconfigurable components. In this section, we also discuss FPGA bitstream protection methods. Section 4.3 primarily discusses the existing equivalence checking and theorem proving methods for ensuring trustworthiness of soft IP cores. Finally, Sect. 4.4 concludes the chapter.

4.2 Design for IP Protection

Design for IP protection encompasses methods for authentication and prevention of IP from piracy, reverse engineering, overbuilding, cloning, and malicious tampering. Authentication approaches include IP watermarking [8–10] and IP fingerprinting [11–13], which can be used by IP owners for detecting and tracking both legal and illegal usages of their designs. However, these methods cannot prevent reverse engineering of designs and insertion of malicious logic. These limitations are overcome by the prevention methods. Most of the currently existing prevention approaches can be grouped under hardware locking/encryption (see Fig. 4.2). Hardware locking/encryption methods can be further divided into (i) combinational logic

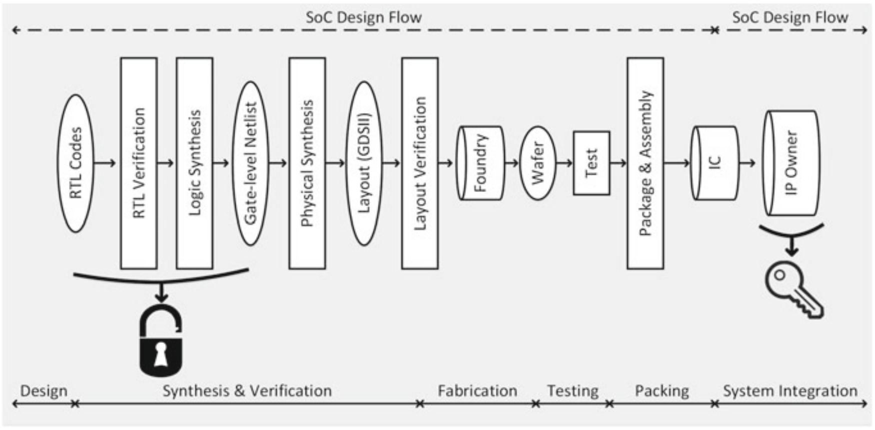

Fig. 4.2 Different stages at which hardware locking/encryption methods are applied

locking [14–18], (ii) finite state machine (FSM) locking [19–28], and (iii) locking using reconfigurable components [29–31]. The combinational logic locking method includes cryptographic algorithm for locking and logic encryption. The FSM locking methods include hardware obfuscation techniques and active hardware metering. The FPGA protection methods focus on securing the bitstream. In the rest of this section we will describe the threat model and each prevention method in details.

4.2.1 Threat Model

Security threats to an IP/IC vary depending on the location of the adversary in the supply chain (see Fig. 4.3). Below we briefly explain these threats to an IP/IC:

- *Cloning*: Adversary creates an exact copy or clone of the original product and sell it under a different label. To carry out cloning of ICs, an attacker should be either a manufacturer, system integrator, or a competing company equipped with necessary tools.
- *Counterfeiting*: When cloned products are sold under the label of the original vendor, without their authorization, it is called counterfeiting. This can be performed by an attacker in a manufacturing facility or by companies having capability to manufacture replicas of the original chip.
- *IC Overbuilding*: Another threat to an IC designer is overbuilding. In overbuilding, manufacturer or system integrator fabricates more IC than authorized.
- *IP Piracy*: In IP piracy, a system integrator steals the IP to claim its ownership or sell it illegally.
- *Reverse Engineering*: By analyzing an existing IC, manufacturers, system integrators, or companies having reverse engineering capabilities, can bypass security

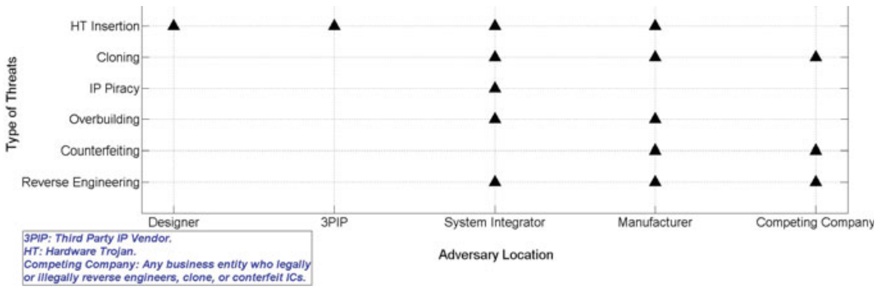

Fig. 4.3 Security threats at different stages of the supply chain

measures to learn and reuse the elements of IP blocks, such as implementation strategies, optimization algorithms, and design details. Consequently, adversaries can recreate the IP cores and sell them at much lower cost. Note that reverse engineering can also be used for IP piracy detection but it is out of the scope of this chapter.

- *Malicious Tampering*: Third-party IP providers can perform malicious tampering of IP by inserting hardware Trojans into the IP cores. Such attacks can also be performed by system integrators who have access to the IP core, or by manufacturer who can manipulate the lithographic masks by adding, deleting, or modifying gates/wires.

To protect an IP/IC from such threats, many prevention methods have been proposed, which are described in Sects. 4.2.2 and 4.2.3.

4.2.2 Combinational Logic Locking/Encryption

Combinational logic locking augments a combinational logic network by adding a group of lock inputs. Only if the correct key (generated at random) to the lock inputs is applied, the augmented network can produce the correct functionality [14, 15]. The locking mechanism is carried out by adding XOR gates on non-critical paths of the IC and by providing control to the key registers. Combinational locking and public-key cryptography was used in [14] for protecting the IC design. In [15], an attack was demonstrated to break the IP/IC protection method of [14] by "sensitizing" the key values to the output. The attacker on obtaining the modified netlist identified the unknown key by observing the output, provided the other key bits did not interfere with the "sensitized" path. In order to prevent key-leakage by "key-sensitization," key gates were inserted in such a way that propagation of a key value was possible only if certain conditions were forced on other key inputs. As these key inputs were not accessible to an attacker, they could not "sensitize" key values, thereby leaving the attacker the only option of brute force attacks. An algorithm was presented in [15], which used key gates interference graphs for insertion of key gates in such a way that attackers needed exponential number of brute force attempts to decipher the key. Compared to random insertion, this procedure incurred less area overhead as it required less number of XOR/XNOR as key gate. Another limitation of the combinational logic locking method of [14] was that inappropriate key input did not affect the output of the circuit.

Logic encryption was proposed in [16, 17], which used conventional fault simulation techniques and tools to guide the XOR/XNOR gate insertions and produced wrong output with 50 % Hamming distance between the correct and wrong outputs for an invalid key. This method masked the functionality and the implementation of a design by inserting key gates into the original design. To prevent collusion attack, physical unclonable functions (PUFs) were used to produce unique user keys for encrypting each IC. Instead of encrypting the design file by a cryptographic algorithm, the *logic encryption* method encrypted the hardware functionality. The performance overhead of this method was smaller than random key gate insertion method as it used smaller number of XOR/XNOR gates to achieve the 50 % Hamming distance.

Another combinational locking method was proposed in [18], which protected ICs from illegal overproduction and potential hardware Trojans insertion by minimizing rare signals of a circuit. The method made it harder for an attacker to exploit rare signals of a circuit to incorporate a hardware Trojan. An encryption algorithm modified the circuit, but preserved its functionality. The algorithm uses a probability-based method [32] to identify signals with low controllability. Among the identified signals, candidate signals for encryption were the ones with an unbalanced probability (signals with probability below 0.1 or above 0.9). For encryption, AND/OR gates were inserted in paths with large slack time and unbalanced probability. The type of gates to be inserted depended on the value of probability on the signal. When the probability of the signal was close to 0, an OR gate was included and the corresponding key value was 0 and when the probability was close to 1, an AND gate was included and the corresponding key value was 1. However, this method could not create multiple encryption key for the same design and hence, all the IP consumers of the design used the same key. Due to this limitation, it was not effective for preventing IP piracy.

4.2.3 Finite State Machine Locking/Encryption

Finite state machine (FSM) locking obfuscates a design by augmenting its state machine with a set of states. The modified FSM transit from the obfuscated states to the normal operating states after applying the specific input sequence (aka obfuscation key). The obfuscation method approximately transform the hardware design by preserving its functionality.

The FSM-based obfuscation method protects an IP/IC from reverse engineering, IP piracy, IC overproduction, and hardware Trojan insertion. Several variations of this method have been proposed in the literature [14–31]. In this chapter, we discuss three of its variations: (i) obfuscation by modifying gate-level netlist, (ii) obfuscation by modifying RTL code, and (iii) obfuscation using reconfigurable logic.

Obfuscation by Modifying Gate-Level Netlist

One of the methods for obfuscating a hardware design is to insert an FSM in the gate-level netlist [19–23]. The method in [19] obfuscated and authenticated an IP core by incorporating structural modification in the netlist. Along with the state transition function, large fan-in and fan-out nodes of the netlist were modified such that the design produces undesired output until a specific vector was given at the primary inputs. The circuit was re-synthesized after the modification to hide the structural changes. The FSM, which was inserted into the netlist to modify the state transition graph (STG), was connected to the primary inputs of the circuit. Depending on an initialization key sequence, the FSM operated the IP core in either the normal mode or the obfuscated mode. The maximum number of unique fan-out nodes (N_{max}) of the netlist were identified using an *iterative ranking algorithm*. The output of the FSM and the modified (N_{max}) nodes were given to an XOR gate. When the FSM output was 0, the design produced correct behavior. Although the method required less area and power overheads, it did not analyze security of the design. New methods were proposed to overcome this limitation [20–22]. A metric was developed to quantify the mismatch between the obfuscated design and the original design [20]. Also, the effect of obfuscation on security of the design was evaluated. Certain modifications were made to the methodology of [19] such as embedding "modification kernel function" for modifying the nodes selected by the *iterative ranking algorithm* and adding an authentication FSM, which acted as a digital watermark. These modifications prevented attacks from untrusted parties in the design flow with knowledge of the initialization sequence.

The obfuscation scheme of [20] was extended in [21, 22] to prevent insertion of trigger-activated hardware Trojans. The new method also ensured that such Trojans, when activated in the obfuscated mode, did not affect normal operation of the circuit. To incorporate these changes, the obfuscation state space was divided into (i) initialization state space and (ii) isolation state space [21]. On applying the correct key at power on, the circuit transitioned from the initialization state space to normal state space. However, an incorrect key transitioned the circuit to isolation state space from which the circuit could not return to the normal state space. Due to the extreme rareness of the transition condition for normal state space, it was assumed that the attacker was stuck in the isolation state space. Also, the insertion of a Trojan with wrong observability/controllability in the obfuscation mode would increase its detection probability at post-manufacturing testing. To further increase the probability of Trojan detection, the state space of the obfuscated mode was made larger than the normal mode. The proposed methodology was robust enough to prevent reverse engineering of modified netlist with large sequential circuits. Also, the area and power overheads of the method were relatively low.

However, the methodology of [20] could not protect an evaluation version of firm IP core. In [23], this problem was overcome by embedding a FSM in the IP netlist. The FSM disrupted normal functional behavior of the IP after its evaluation period.

The number of cycles required to activate the FSM depended on the number of bits in the state machine. Also, the activation probability decreased if the number of trigger nodes were increased. This method helped in putting an expiry date on the evaluation copy of the hardware IP. To distinguish between legally sold version of an IP and its evaluation version containing the FSM, IP vendors either (i) used disabling key to deactivate the FSM or (ii) provided a FSM-free version. The method structurally and functionally obfuscated the FSM to conceal it during reverse engineering. Area overhead of this method was directly proportional to the size of the FSM. However, the overhead decreased with an increase in size of the original IP.

Obfuscation by Modifying RTL Code of Design

Apart from netlist, obfuscation can also be carried out in RTL code of the design [24–27]. A key-based obfuscation approach for protecting synthesizable RTL cores was developed in [27]. The RTL design was first synthesized into a technology independent gate-level netlist and then obfuscated using the method of [19]. Then, the obfuscated netlist was decompiled into RTL in such a way that the modifications made on the netlist were hidden and the high-level HDL constructs preserved. A simple metric was presented to quantify the level of such structural and semantic obfuscation. This approach incurred minimal design overhead and did not affect adversely the automatic synthesis process of the resultant RTL code. However, decompilation removed some preferred RTL constructs and hence made the design undesirable for certain preferred design constraints. This limitation was overcome in [26], where the RTL code was first converted into a control and data flow graph (CDFG) and then a key-activated "Mode-Control FSM" was inserted to obfuscate the design. The CDFG was built by parsing and transforming concurrent blocks of RTL code. Small CDFGs were merged to build larger ones with more nodes. This helped in better obfuscation of the "Mode-Control FSM," which operated the design either in normal or obfuscated mode. This FSM was realized in the RTL by modifying a set of host registers. To further increase the level of obfuscation, state elements of the FSM were distributed in a non-contiguous manner inside one or more registers. After hosting the FSM in a set of selected host registers, several CDFG nodes were modified using the control signals generated from the FSM. The nodes with large fan-out cones were selected for modification, as they ensured maximum change in functional behavior at minimal design overhead. At the end of modifications on the CDFG, the obfuscated RTL was generated, which on powering up, initialized the design at the obfuscated mode. Only on the application of a correct input key, the "mode-control" FSM transited the system through a sequence of states to the normal mode. This method incurred low area and power overheads.

Another approach obfuscates the RTL core by dividing the overall functionality of the design into two modes, "Entry/Obfuscated mode" and "Functional mode," and encoding the path from the *entry/obfuscated mode* to the *functional mode* with a "code-word" [24]. The functionality of the circuit was divided by modifying the FSM representing the core logic. To modify the FSM, its states were extended and divided into *entry/obfuscated mode* and *functional mode* states. Only on the application of the right input sequence at the *entry mode*, the correct *Code-Word* was formed, which produced correct transitions in the *functional mode*. Unlike those methods where an invalid key disallowed entry to the *normal mode*, this method always allowed entry to the *functional mode*. However, the behavior in the functional mode depended on the value of the *Code-Word*. This *Code-Word* was not stored anywhere on chip, but it was formed dynamically during the *entry mode*. It was integrated into the transition logic to make it invisible to the attacker. In this method, the length of the *Code-Word* was directly related to the area overhead and security level required by the designer. A longer *Code-Word* meant higher level of security against brute force attacks, but at the cost of higher area overhead.

In [25], a key-based obfuscation method having two modes of operation, *normal mode* and *slow mode*, was developed to prevent IP piracy on sequential circuits. This method modified the state transition graph (STG) in such a way that the design operated in either mode depending on whether it was initialized with the correct key state. The key state was embedded in the power-up states of the IC and was known only to the IP owner. When the IP owner received the fabricated chip, power-up states were reset from the fixed initial state to the key state. As the number of power-up states was less in the design, chances of the IC being operational in the *normal mode* on random initialization were significantly reduced. Moreover, powering up the design with an incorrect initial state operated the IC in the *slow mode*, where it functioned slower than the *normal mode* without causing significant performance difference. This functionality prevented IP pirates from suspecting the performance degradation in the IC and the presence of *key state* in the design. To modify the STG, four structural operations were performed: (i) *retiming*, (ii) *resynthesis*, (iii) *sweep*, and (iv) *conditional stuttering*. In *retiming*, registers were moved in the sequential circuit using any one of the two operations. These operations included (i) adding a register to all outputs and deleting a register from each input of a combinational node and (ii) deleting a register from all outputs and adding a register to each input of a combinational node. *Resynthesis* restructured the netlist within the register boundaries whereas removal of redundant registers and logic that did not affect output was done using *sweep*. Both the *resynthesis* and the *retiming* operations preserved logical functionality of the design. *Conditional stuttering* involved addition of control logic to the circuit to stutter the registers under a given logic condition. On the other hand, inverse *conditional stuttering* removed certain control logic. *Stuttering* operations were done to obtain circuits which were cycle–accurate–equivalent. This method mainly focused on those real-time applications which were very sensitive to throughput. Unlike existing IC metering techniques, the secret key in this method was implicit, thus making it act as a hidden watermark. However, the area and power overheads of this method were higher than previous approaches.

An *active hardware metering* approach prevents overproduction of IC by equipping designers with the ability to lock each IC and unlock it remotely [28]. In this method, new states and transitions were added to the original finite state machine (FSM) of the design to create a *boosted finite state machine* (BFSM). This structural manipulation preserved the behavioral specification of the design. Upon activation, the BFSM was placed in the power-up state using an unique ID which was generated by the IC. To bring the BFSM into the functional initial state, the designer used an input sequence generated from the transition table. Black hole states were integrated with the BFSM to make the *active metering* method highly resilient against the brute force attacks. This method incurred low overhead and was applicable to industrial-size design.

Obfuscation Using Reconfigurable Logic

Reconfigurable logic was used in [29, 30] for obfuscation of ASIC design. In [29], reconfigurable logic modules were embedded in the design and their final implementation was determined by the end user after the design and the manufacturing process. This method assumed that the supply chain adversary has knowledge of the entire design except the exact implementation of the function and the internal structure of reconfigurable logic modules. The lack of knowledge prohibited an adversary from tampering the logic blocks. Combining this method with other security techniques provided data confidentiality, design obfuscation, and prevention from hardware Trojans. In the demonstration, a code injection Trojan was considered which, when triggered by a specific event, changed input–output behavior or leaked confidential information. The Trojan was assumed to be injected at the instruction decoded unit (IDU) of a processor during runtime and it was not detected by non-lock stepping concurrent checking methods, code integrity checker, and testing. To prevent such a Trojan attack, instruction set randomization (ISR) was done by obfuscating the IDU. For obfuscation, reconfigurable logic was used, which concealed opcode check logic, instruction bit permutation, or XOR logic. This method prevented the Trojan from monitoring an authentic computation. Moreover, Trojans which were designed to circumvent this method no longer remained stealthy and those trying to duplicate the IDU or modify it caused significant performance degradation. It was shown that the minimum code injection Trojan with a 1 KB ROM resulted in an area increase of 2.38 % for every 1 % increase in the area of the LEON2 processor.

To hide operations within the design and preserve its functionality, the original circuit was replaced with PUF-based logic and FPGA in [30]. PUF was also used to obfuscate signal paths of the circuit. The architecture for signal path obfuscation was placed in a location where most flip-flops were affected most number of times. To prevent this technique from affecting critical paths, wire swapping components (MUX'es with PUF as select input) were placed between gates with positive slack. The PUF-based logic and signal path obfuscation techniques were used simultaneously to minimize delay constraints of the circuit and maximize its security under user-specified area and power overheads. Two types of attacks were considered: (i) adversary can read all flip-flops, but can only write to primary inputs; (ii) adversary can read and write to all flip-flops of the circuit [30]. It was assumed that the adversary has complete knowledge of circuit netlist, but not of input–output mapping of any PUFs. For preventing the first type of attack, FPGA was used after the PUF and PUF were made large to accept a large challenge. To prevent the second attack, a PUF was placed in a location that was difficult to control directly using the primary inputs of the circuit. By preventing this attack, reverse engineering was made difficult for an adversary. These two methods were demonstrated on ISCAS 89 and ITC 99 benchmarks and functionality of circuits were obfuscated with area overhead upto 10 %.

In [31], obfuscation of DSP circuit was done using high-level transformation, key-based FSM, and a reconfigurator. High-level transformation does structural obfuscation of the DSP circuit at HDL or netlist level by preserving its functionality. This transformation was chosen based on the DSP application and performance requirements (e.g., area, speed, power, or energy). For performing high-level transformation on the circuit, reconfigurable secure switches were designed in [31]. These switches were implemented as multiplexers, whose control signals were obtained from a FSM designed using ring counters. Securities of these switches were directly related to the design of the ring counters. Another FSM, called the *obfuscated FSM*, was incorporated in the DSP circuit along with the reconfigurator. A configuration key was given to the *obfuscated FSM* for operating the circuit correctly. This key consisted of two parts: an *L-bit* initialization key and a *K-bit* configure data. The initialization key was used to activate reconfigurator via the *obfuscated FSM*, whereas *configure data* was used by the reconfigurator to control the operation of the switches. As configuration of the switches required correct initialization key and configure data, attacks targeting either of them could not affect the design. An adversary attempting to attack a DSP circuit, obfuscated with this method, had to consider the length of the configuration key and the number of input vectors required for learning the functionality of each variation mode. Structural obfuscation degree (SOD) and functional obfuscation degree (FOD) were used as metrics for measuring simulation-based attack and manual attack (visual inspection and structural analysis). SOD was estimated for manual attacks, whereas FOD estimated obfuscation degree of simulation-based attacks. A higher value of SOD and FOD indicated a more secure design. The area and the power overheads of this method were low.

4.2.4 Protection Methods for FPGA IP

Field-programmable gate arrays (FPGAs) have been widely used for many applications since 1980s. They provide considerable advantage in regards to design cost and flexibility. Due to accelerated time-to-market, designing a complete system on FPGA is becoming a daunting task. To meet the demands, designers have started using/reusing third-party intellectual property (IP) modules, rather than developing a system from scratch. However, this has raised the risk of FPGA IP piracy. Protection methods [33–36] have been proposed to mitigate this issue. In [35], a protection scheme was proposed which used both public-key and symmetric-key cryptography. To reduce area overhead, the public-key functionality was moved to a temporary configuration bitstream. Using five basic steps, the protection scheme enabled licensing of FPGA IP to multiple IP customers. However, this scheme restricted system integrators to the use of IP from a single vendor. The scope of the FPGA IP protection method of [35] was extended in [36], where system integrators could use cores from multiple sources. In [33], implementation of the protection methods of [35, 36] was carried out on commercially available devices. For securely transporting the key,

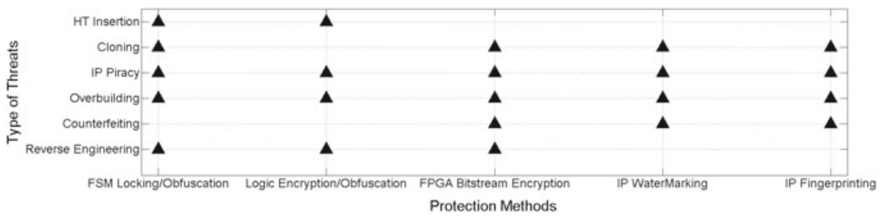

Fig. 4.4 Different protection methods

[33] used symmetric cryptography and trusted third-party provider. Use of symmetric cryptography also reduced the size of temporarily occupied reconfigurable logic for building the IP decryption key.

A practical and feasible protection method for SRAM-based FPGA was given in [34]. This approach allowed licensing of IP cores on a per-device basis and it did not require contractual agreement with trusted third-parties, large bandwidth, and complicated communication processes. The IP instance was encrypted for each system integrator and decryption key was generated using the license for the chips. This procedure ensured that the licensed IP core was used only on the contracted devices. Moreover, it helped to prevent IP core counterfeiting by tracking the unique fingerprint embedded in the licensed IP instance of the vendor. The proposed scheme did not require an external trusted third-party (TTP) and was applicable on IP cores designed for commercial purposes. It also helped in secure transaction of IP cores and prevented sophisticated attackers from cloning, reverse engineering, or tampering contents of the IP core.

A summary of all the above-described protection methods is shown in Fig. 4.4.

4.3 IP Certification

Recently, pre-silicon trust evaluation approaches have been developed to counter the threat of untrusted third-party resources [37–39]. Most of these methods try to trigger the malicious logic by enhancing functional testing with extra test vectors. Toward this end, authors in [37] proposed a method for generating "Trojan Vectors" to activate hardware Trojans during functional testing. To identify suspicious circuitry, unused circuit identification (UCI) [39] method analyzed the RTL code to find lines of code that were never used. However, these methods assume that the attacker uses rarely-occurring events as Trojan triggers. This assumption was voided in [40], where hardware Trojans were designed using "less-rare" trigger events.

Due to the limitations of enhanced functional testing methods in security evaluation, researchers started looking into formal solutions. Although at its early stage, formal methods have already shown their benefits over testing methods in exhaustive security verification [41–44]. A multi-stage approach was used in [41] for detection of hardware Trojans by identifying suspicious signals. The stages of this method included assertion-based verification, code coverage analysis, redundant circuit removal, equivalence analysis, and use of sequential automatic test pattern generation (ATPG). In [42–44], the PCH framework ensured the trustworthiness of soft IP cores by verifying security properties. With the help of the Coq proof assistant [45], formal security properties were proved in PCH. A review of currently existing formal methods for hardware security is given in [46].

Interactive Theorem Prover

Theorem provers are used to prove or disprove properties of systems expressed as logical statements. Since 1960s, several automated and interactive theorem provers have been developed and used for proving properties of hardware and software systems. However, verifying large and complex systems using theorem provers require excessive effort and time. Moreover, automated theorem provers require more developmental effort than proof assistants. Despite these limitations, theorem provers have currently drawn a lot of attentions in verification of security properties on hardware designs. Among all the formal methods, they have emerged as the most prominent solution for providing high-level protection of the underlying designs.

The proof-carrying hardware (PCH) framework uses interactive theorem prover and SAT solvers for verifying security properties on soft IP cores. This approach was used for ensuring the trustworthiness of RTL and firm cores [42, 47–49]. It is inspired from the proof-carrying code (PCC), which was proposed in [50]. Using the PCC mechanism, untrusted software developers/vendors certify their software code. During the certification process, software vendor develops *safety proof* for the safety policies provided by software customers. The vendor then provides the user with a PCC binary file, which includes the formal proof of the safety properties encoded with the executable code of the software. The customer becomes assured of the safety of the software code by quickly validating the PCC binary file in a proof checker. Efficiency of this approach in reducing validation time at the customer end led to its adoption in different applications.

Using the concept of PCC, authors in [48, 49, 51, 52] developed the PCH framework for dynamically reconfigurable hardware platforms. In this framework, authors used runtime combinational equivalence checking (CEC) for verifying equivalence between the design specification and the design implementation. A Boolean satisfiability (SAT) solver was used to generate resolution proof for unsatisfiability of the combinational miter circuit, represented in a conjunctive normal form (CNF). The proof traces were combined with the bitstream into a proof-carrying bitstream by the vendor and given to the customer for validation. However, the approach did not consider exchange of a set of security properties between the customer and the vendor. Rather it considers safety policy, which included agreements on a specific bitstream format, on a CNF to represent combinational functions, and the propositional calculus for proof construction and verification.

Another PCH framework was proposed in [42, 47], which overcame the limitations of the previous framework and expanded it for verification of security properties on soft IP cores. The new PCH framework was used for security property

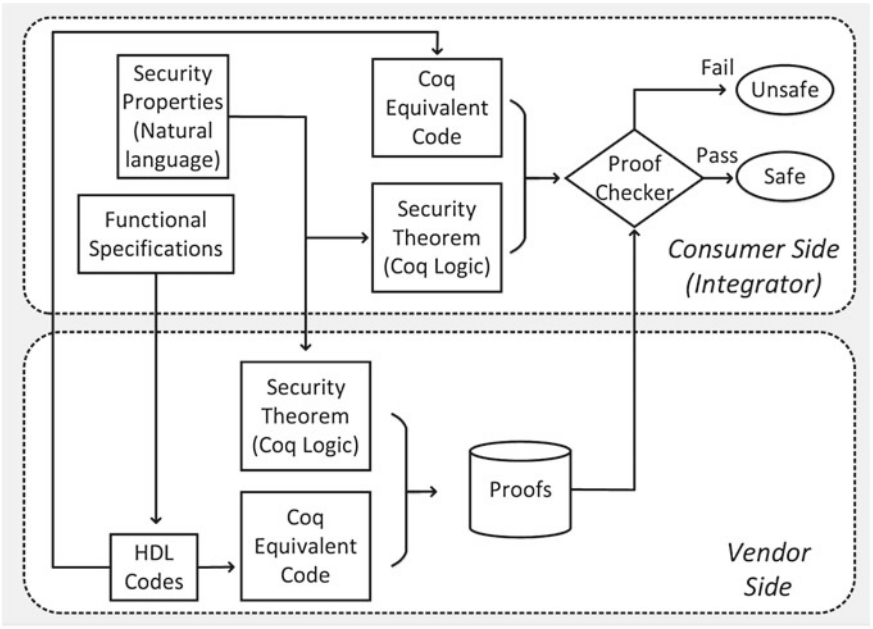
Security Properties (Natural language)
Coq Equivalent Code
Fail
Unsafe
Pass
Safe
Proof Checker
Functional Specifications
Security Theorem (Coq Logic)
Consumer Side (Integrator)
Security Theorem (Coq Logic)
Proofs
Coq Equivalent Code
HDL Codes
Vendor Side

Fig. 4.5 Working procedure of the PCH framework [46]

verification on synthesizable IP cores. In the framework, Hoare-logic style reasoning was used to prove the correctness of the RTL code and implementation was carried out using the Coq proof assistant [45]. As Coq supported automatic proof checking, the security proof validation effort of IP customers was reduced. Moreover, usage of the Coq platform by both IP vendors and IP consumers ensures that same deductive rules could be used for validating the proof. However, Coq does not recognize commercial hardware description languages (HDLs) and security properties expressed in a natural language. To solve this problem, semantic translation of HDLs and informal security specifications to calculus of inductive construction (CIC) was done. Based on this PCH framework, a new trusted IP acquisition and delivery protocol was proposed (see Fig. 4.5), in which IP consumers provided both functional specifications and a set of security properties to IP vendors. IP vendors then developed the HDL code based on the functional specifications. The HDL code and security properties were then translated to CIC. Subsequently, proofs were constructed for security theorems and the transformed HDL code. The HDL code and proof for security properties were combined into a trusted bundle and delivered to the consumer. Upon receiving the trusted bundle, IP consumers first generate the formal representation of the design and security properties in CIC. The translated code, combined with formal theorems and proofs, was quickly validated using the proof checker in Coq platform.

The PCH framework was also extended to support verification of gate-level circuit netlist [44]. With the help of the new gate-level framework, authors in [44] formally analyzed the security of design-for-test (DFT) scan chains, which is the industrial standard testing method, and formally proved that a circuit with scan chain can violate data secrecy property. Although various attack and defense methods have been developed to thwart the security concerns raised by DFT scan chains [53–58], methods for formally proving the vulnerability of scan chain inserted designs did not exist. For the first-time vulnerability of such a design was proved using the PCH framework of [44]. The same framework was also applied in built-in-self-test (BIST) structure to prove that BIST structure can also leak internal sensitive information [44].

Equivalence Checking

Orthogonal to the theorem proving approach is equivalence checking, which ensure that the specification and the implementation of a circuit are equivalent. The traditional equivalence checking approach uses a SAT solver for proving functional equivalence between two representations of a circuit. In this approach, if the specification and the implementation were equivalent, the output of the "xor" gate was always zero (false). If the output was true for any input sequence, it implied that the specification and the implementation produced different outputs for the same input sequence. Following the equivalence checking approach, [38] proposed a four-step procedure to filter and locate suspicious logic in third-party IPs. In the first step, easy-to-detect signals were removed using functional vectors generated by a sequential ATPG. In the next step, hard-to-excite and/or propagate signals were identified using a full-scan N-detect ATPG. To narrow down the list of suspected signals and identify the actual gates associated with the Trojan, a SAT solver was used in the third step for equivalence checking of the suspicious netlist containing the rarely triggered signals against the netlist of the circuit exhibiting correct behavior. At the final step, clusters of untestable gates in the circuit were determined using the region isolation approach on the suspected signals list.

However, traditional equivalence checking techniques could result in state space explosion when large IP blocks were involved with significantly different specifications and implementations. They also could not be used on complex arithmetic circuits with larger bus widths. An alternative approach was to use computer symbolic algebra for equivalence checking of arithmetic circuit. These circuits constituted a significant portion of datapath in signal processing, cryptography, multimedia applications, error root causing codes, etc. Due to this, their chances of malfunctioning were very high. The new equivalence checking approach allowed verification of such large circuits and it did not cause state space explosion.

4.4 Conclusion

In this chapter, we analyzed existing prevention and certification methods for soft/firm hardware IP cores. The prevention methods largely consisted of various hardware locking/encryption schemes. These methods protected IP cores from piracy, overbuilding, reverse engineering, cloning, and malicious modifications. On the other hand, formal methods, such as theorem proving and equivalence checking, helped validate the trustworthiness of IP cores. These methods can help certify the trustworthiness of IP cores. Meanwhile, after a thorough analysis of all these proposed IP validation/protection methods, we realized that a single method is not sufficient to eliminate all the threats to IP cores. Combination of these methods becomes a necessity in order to ensure the security of IP cores and further secure the modern semiconductor supply chain.

Acknowledgements This work was supported in part by the National Science Foundation (CNS-1319105).

References

1. Kocher, P.: Advances in Cryptology (CRYPTO'96). Lecture Notes in Computer Science, vol. 1109, pp. 104–113 (1996)
2. Kocher, P., Jaffe, J., Jun, B.: Advances in Cryptology–CRYPTO'99, pp. 789–789 (1999)
3. Quisquater, J.J., Samyde, D.: Smart Card Programming and Security. Lecture Notes in Computer Science, vol. 2140, pp. 200–210 (2001)
4. Gandolfi, K., Mourtel, C., Olivier, F.: Cryptographic Hardware and Embedded Systems (CHES) 2001. Lecture Notes in Computer Science, vol. 2162, pp. 251–261 (2001)
5. Chari, S., Rao, J.R., Rohatgi, P.: Cryptographic Hardware and Embedded Systems—Ches 2002. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer, Berlin (2002)
6. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: IEEE Trans. Comput. **51**(5), 541 (2002)
7. Tiri, K., Akmal, M., Verbauwhede, I.: Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European, pp. 403–406 (2002)
8. Fan, Y.C., Tsao, H.W.: Electr. Lett. **39**(18), 1316 (2003)
9. Torunoglu, I., Charbon, E.: IEEE J. Solid-State Circuits **35**(3), 434 (2000)
10. Kahng, A.B., Lach, J., Mangione-Smith, W.H., Mantik, S., Markov, I.L., Potkonjak, M., Tucker, P., Wang, H., Wolfe, G.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **20**(10), 1236 (2001)
11. Lach, J., Mangione-Smith, W.H., Potkonjak, M.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **20**(10), 1253 (2001)
12. Qu, G., Potkonjak, M.: Proceedings of the 37th Annual Design Automation Conference, pp. 587–592 (2000)
13. Chang, C.H., Zhang, L.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **33**(1), 76 (2014)
14. Roy, F.K.J.A., Markov, I.L.: Design, Automation and Test in Europe (DATE), vol. 1 (2008)
15. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, pp. 83–89 (2012)
16. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Design, Automation Test in Europe Conference Exhibition (DATE), vol. 2012, pp. 953–958 (2012). doi:10.1109/DATE.2012.6176634

17. Rajendran, J., Zhang, H., Zhang, C., Rose, G., Pino, Y., Sinanoglu, O., Karri, R.: IEEE Trans. Comput. **99** (2013)
18. Dupuis, S., Ba, P.S., Natale, G.D., Flottes, M.L., Rouzeyre, B.: Conference on IEEE 20th International On-Line Testing Symposium (IOLTS), IOLTS '14, pp. 49–54 (2014)
19. Chakraborty, R., Bhunia, S.: IEEE/ACM International Conference on Computer-Aided Design 2008. ICCAD 2008, pp. 674–677 (2008). doi:10.1109/ICCAD.2008.4681649
20. Chakraborty, R., Bhunia, S.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **28**(10), 1493 (2009). doi:10.1109/TCAD.2009.2028166
21. Chakraborty, R.S., Bhunia, S.: J. Electr. Test. **27**(6), 767 (2011). doi:10.1007/s10836-011-5255-2
22. Chakraborty, R., Bhunia, S.: IEEE/ACM International Conference on Computer-Aided Design—Digest of Technical Papers, 2009. ICCAD 2009, pp. 113–116 (2009)
23. Narasimhan, S., Chakraborty, R., Bhunia, S.: IEEE Des. Test Comput. **99**(PrePrints) (2011). http://doi.ieeecomputersociety.org/10.1109/MDT.2011.70
24. Desai, A.R., Hsiao, M.S., Wang, C., Nazhandali, L., Hall, S.: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, CSIIRW '13, pp. 8:1–8:4. ACM, New York (2013). doi:10.1145/2459976.2459985
25. Li, L., Zhou, H.: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2–3, pp. 55–60 (2013). doi:10.1109/HST.2013.6581566
26. Chakraborty, R., Bhunia, S.: 23rd International Conference on VLSI Design, 2010. VLSID'10, pp. 405–410 (2010). doi:10.1109/VLSI.Design.2010.54
27. Chakraborty, R., Bhunia, S.: IEEE International Workshop on Hardware-Oriented Security and Trust, 2009. HOST '09, pp. 96–99 (2009). doi:10.1109/HST.2009.5224963
28. Alkabani, Y., Koushanfar, E.: USENIX Security, pp. 291–306 (2007)
29. Liu, B., Wang, B.: Design. Automation and Test in Europe Conference and Exhibition (DATE), pp. 1–6 (2014). doi:10.7873/DATE.2014.256
30. Wendt, J.B., Potkonjak, M.: Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design, ICCAD'14, pp. 270–277. IEEE Press, Piscataway (2014). http://dl.acm.org/citation.cfm?id=2691365.2691419
31. Lao, Y., Parhi, K.: IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **99**, 1 (2014). doi:10.1109/TVLSI.2014.2323976
32. Natale, G.D., Dupuis, S., Flottes, M.L., Rouzeyre, B.: Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE13) (2013)
33. Maes, R., Schellekens, D., Verbauwhede, I.: IEEE Trans. Inf. Forensics Secur. **7**(1), 98 (2012)
34. Zhang, L., Chang, C.H.: IEEE Trans. Inf. Forensics Secur. **9**(11), 1893 (2014)
35. Guneysu, T., Moller, B., Paar, C.: IEEE International Conference on Field-Programmable Technology, ICFPT, pp. 169–176 (2007)
36. Drimer, S., Güneysu, T., Kuhn, M.G., Paar, C.: (2008). http://www.cl.cam.ac.uk/sd410/
37. Wolff, E., Papachristou, C., Bhunia, S., Chakraborty, R.S.: IEEE Design Automation and Test in Europe, pp. 1362–1365 (2008)
38. Banga, M., Hsiao, M.: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 56–59 (2010)
39. Hicks, M., Finnicum, M., King, S.T., Martin, M.M.K., Smith, J.M.: Proceedings of IEEE Symposium on Security and Privacy, pp. 159–172 (2010)
40. Sturton, C., Hicks, M., Wagner, D., King, S.: 2011 IEEE Symposium on Security and Privacy (SP), pp. 64–77 (2011)
41. Zhang, X., Tehranipoor, M.: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 67–70 (2011)
42. Love, E., Jin, Y., Makris, Y.: IEEE Trans. Inf. Forensics Secur. **7**(1), 25 (2012)
43. Jin, Y., Yang, B., Makris, Y.: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 99–106 (2013)
44. Jin, Y.: IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (2014)
45. INRIA: The coq proof assistant (2010). http://coq.inria.fr/

46. Guo, X., Dutta, R.G., Jin, Y., Farahmandi, F., Mishra, P.: Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE (2015) (To appear)
47. Love, E., Jin, Y., Makris, Y.: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 12–17 (2011)
48. Drzevitzky, S., Kastens, U., Platzner, M.: International Conference on Reconfigurable Computing and FPGAs, pp. 189–194 (2009)
49. Drzevitzky, S.: International Conference on Field Programmable Logic and Applications, pp. 255–258 (2010)
50. Necula, G.C.: POPL'97: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 106–119 (1997)
51. Drzevitzky, S., Platzner, M.: 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip, pp. 1–8 (2011)
52. Drzevitzky, S., Kastens, U., Platzner, M.: Int. J. Reconfig. Comput. **2010** (2010)
53. Yang, B., Wu, K., Karri, R.: Test Conference, 2004. Proceedings. ITC 2004. International, pp. 339–344 (2004)
54. Nara, R., Togawa, N., Yanagisawa, M., Ohtsuki, T.: Proceedings of the 2010 Asia and South Pacific Design Automation Conference, pp. 407–412 (2010)
55. Yang, B., Wu, K., Karri, R.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **25**(10), 2287 (2006)
56. Sengar, G., Mukhopadhyay, D., Chowdhury, D.: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **26**(11), 2080 (2007)
57. Da Rolt, J., Di Natale, G., Flottes, M.L., Rouzeyre, B.: 2012 IEEE 30th VLSI Test Symposium (VTS), pp. 246–251 (2012)
58. Rolt, J., Das, A., Natale, G., Flottes, M.L., Rouzeyre, B., Verbauwhede, I.: Constructive Side-Channel Analysis and Secure Design. In: Schindler, W., Huss, S. (eds.) Lecture Notes on Computer Science, vol. 7275, pp. 89–104. Springer, Berlin (2012)