Chapter 4
Code Coverage Analysis for IP Trust Verification

Adib Nahiyan and Mark Tehranipoor

4.1 Introduction

Due to globalization of the semiconductor design and fabrication process, integrated circuits (ICs) are becoming increasingly vulnerable to malicious activities and alterations. Today's system-on-chips (SoCs) usually contain tens of IP cores (digital and analog) performing various functions. In practice, very seldom IPs are developed by the SoC integrator; in fact most of them are currently being designed offshore by 3PIP vendors. This raises a major concern towards the trustworthiness of 3PIPs. These concerns have been documented in various reports and technical events [1].

IP trust problem is defined as the possibility of inserting Trojan into 3PIPs by IP vendors during SoC design. This issue has gained significant attention as a Trojan-inserted by 3PIP vendor can create backdoors in the design through which sensitive information can be leaked and other possible attacks (e.g., denial of service, reduction in reliability, etc.) can be performed [2, 3].

Detection of Trojans in 3PIP cores is extremely difficult as there is no golden version against which to compare a given IP core during verification. In theory, an effective way to detect a Trojan in an IP core is to activate the Trojan and observe its effects, but the Trojan's type, size, and location are unknown, and its activation condition is most likely a rare event [4]. The conventional side-channel techniques for IC trust are not applicable to IP trust. When a Trojan exists in an IP core, all the fabricated ICs will contain Trojans. The only trusted component would be the specification from the SoC designer which defines the function, primary input and output, and other information about the 3PIP that they intend to use in their systems. A Trojan can be very well hidden during the normal functional operation of the 3PIP

A. Nahiyan (✉) • M. Tehranipoor
University of Florida, Gainesville, FL, USA
e-mail: adib1991@ufl.edu; tehranipoor@ece.ufl.edu

P. Mishra et al. (eds.), *Hardware IP Security and Trust*,
DOI 10.1007/978-3-319-49025-0_4

supplied as register transfer level (RTL) code. A large industrial-scale IP core can include thousands of lines of code. Identifying the few lines of RTL code in an IP core that represents a Trojan is an extremely challenging task.

Given the complexity of the problem, there is no silver bullet available to detect hardware Trojans in 3PIPs. An IP trust verification technique that tries to address the IP trust problem is presented in [5]. This chapter will present this technique in details. Several concepts such as formal verification, code coverage analysis, and ATPG methods have been employed in this technique to achieve high confidence in whether the circuit is Trojan-free or Trojan-inserted. This IP trust verification technique is based on identification of suspicious signals. Suspicious signals are identified first by coverage analysis with improved test bench. Removing redundant circuit and equivalence theorems are then applied to reduce the number of suspicious signals. Sequential ATPG is used to generate patterns to activate suspicious signals which can trigger the Trojan.

4.2 SoC Design Flow

A typically SoC design flow is shown in Fig. 4.1. Design specification by the SoC integrator is generally the first step. The SoC integrator then identifies a list of IPs necessary to implement the given specification. These IP cores are either developed in-house or purchased from 3PIP vendors. These 3PIP cores can be procured from the vendors in one of the following three ways [6]:

- soft IP cores are delivered as synthesizable register transfer level (RTL) hardware description language (HDL)
- hard IP cores are delivered as GDSII representations of a fully placed and routed core design
- firm IP cores are optimized in structure and topology for performance and area, possibly using a generic library.

After developing/procuring all the necessary soft IPs, the SoC design house integrates them to generate the RTL specification of the whole system. SoC integrator then synthesizes the RTL description into a gate-level netlist based on the logic cells and I/Os of a target technology library, then they may integrate gate-level IP cores from a vendor into this netlist. They also add design-for-test (DFT) structures to improve the design's testability. The next step is to translate the gate-level netlist into a physical layout based on logic cells and I/O geometries. It is also possible to import IP cores from vendors in GDSII layout file format. After performing static timing analysis (STA) and power closure, developers generate the final layout in GDSII format and send it out for fabrication.

Today's advanced semiconductor technology requires prohibitive investment for each stage of the SoC development procedure. As an example, the estimated cost of owning a foundry was five billion dollar in 2015 [7]. As a result, most semiconductor companies cannot afford maintaining such a long supply chain from

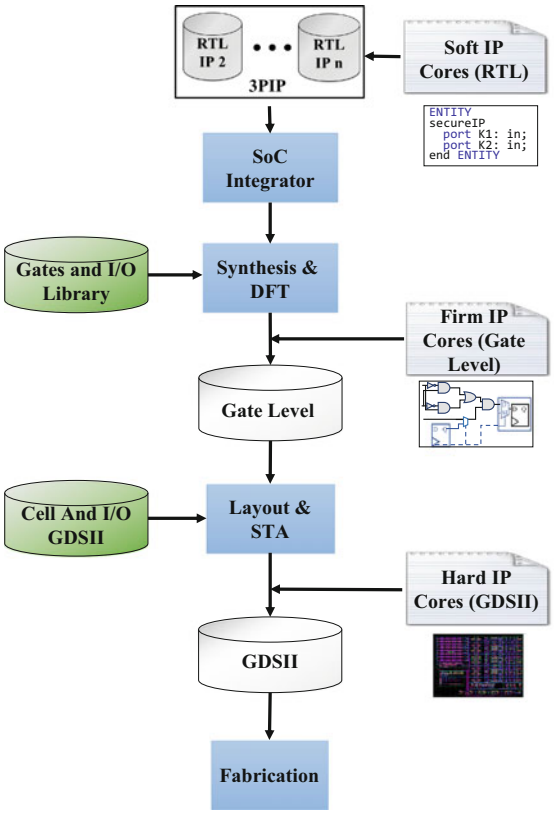

Fig. 4.1 System-on-chip (SoC) design flow consists of IP-core-based design, system integration, and manufacturing

design to packaging. In order to lower R&D cost and speed up the development cycle, the SoC design houses typically outsource fabrication to a third-party foundry, purchase third-party intellectual property (IP) cores, and/or use Electronic Design Automation (EDA) tools from third-party vendors. The use of untrusted (and potentially malicious) third parties increases the security concerns. Thus, the supply chain is now considered susceptible to various attacks, such as hardware Trojan insertion, reverse engineering, IP piracy, IC tampering, IC cloning, IC overproduction, and so forth. Among these, hardware Trojans are arguably one of the biggest concerns and have garnered considerable attention [9].

Trojans can be inserted in SoCs at the RTL, at the gate level during synthesis and DFT insertion, at the layout level during placement and routing, or during IC manufacturing [8]. An attacker can also insert a Trojan through IP cores provided by external vendors. Designers must verify the trustworthiness of IP cores to ensure that they perform as intended, nothing more and nothing less.

4.3 Hardware Trojan Structure

A hardware Trojan is defined as a malicious, intentional modification of a circuit design that results in undesired behavior when the circuit is deployed [8]. The basic structure of a Trojan in a 3PIP can include two main parts, trigger and payload. A Trojan trigger is an optional part that monitors various signals and/or a series of events in the circuit. The payload usually taps signals from the original (Trojan-free) circuit and the output of the trigger. Once the trigger detects an expected event or condition, the payload is activated to perform malicious behavior. Typically, the trigger is expected to be activated under extremely rare conditions, so the payload remains inactive most of the time. When the payload is inactive, the IC acts like a Trojan-free circuit, making it difficult to detect the Trojan.

Figure 4.2 shows the basic structure of the Trojan at gate level. The trigger inputs $(T_1, T_2, \ldots, T_k)$ come from various nets in the circuit. The payload taps the original signal Net_i from the original (Trojan-free) circuit and the output of the trigger. Since the trigger is expected to be activated under rare condition, the payload output stays at the same value as Net_i most of the time. However, when trigger is active (i.e., *TriggerEnable* is "0") the payload output will be different from Net_i; this could result in injecting an erroneous value into the circuit and causing error at the output. Note that Trojans at RTL would have similar functionality to the one shown in Fig. 4.2.

4.4 Related Work

IP trust validation focuses on verifying that an IP does not perform any malicious function, i.e., an IP does not contain any Trojan. Existing IP trust validation techniques can be broadly classified into code/structural analysis, functional verification, logic testing, formal verification, and runtime validation.

Code Coverage Analysis Code coverage is defined as the percentage of lines of code that has been executed during functional verification of the design. This metrics

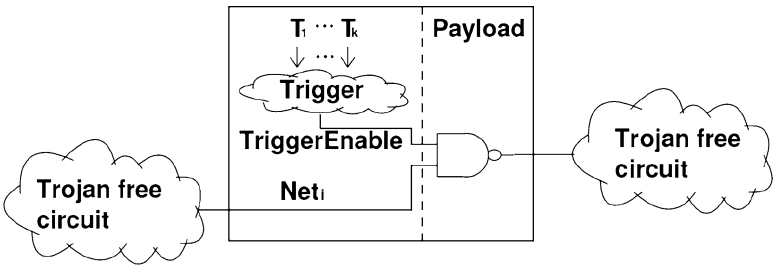

Fig. 4.2 Trojan structure

gives a quantitative measure of the completeness of the functional simulation of the design. Code coverage analysis can also be applied to identify suspicious signals that may be a part of a Trojan and validate the trustworthiness of a 3PIP. In [9], authors have proposed a technique named unused circuit identification (UCI) to find the lines of RTL code that have not been executed during simulation. These unused lines of codes can be considered to be part of a malicious circuit. Authors in [9] proposed to remove these suspicious lines of RTL code from the hardware design and emulate it in the software level. In [10], authors have proposed similar code coverage analysis in combination with hardware assertion checker to identify malicious circuitry in a 3PIP. However, these techniques do not guarantee the trustworthiness of a 3PIP. Authors in [11] have demonstrated that hardware Trojans can be designed to defeat UCI technique. This type of Trojans derives their triggering circuits from less likely events to evade detection from code coverage analysis.

Formal Verification Formal methods such as symbolic execution [12], model checking [13], and information flow [14] have been traditionally applied to software systems for finding security bugs and improving test coverage. Formal verification has also shown to be effective in verifying the trustworthiness of 3PIP [15–17]. These approaches are based on the concept of proof-carrying code (PCC) to formally validate the security-related properties of an IP. In these proposed approaches, an SoC integrator provides a set of security properties in addition to the standard functional specification to the IP vendor. A formal proof of these properties alongside with the hardware IP is then provided by the third-party vendor. SoC integrator then validates the proof by using the PCC. Any malicious modification of the IP would violate this proof indicating the presence of hardware Trojan. However, these approaches cannot ensure complete trust in an IP because the third-party vendor crafts the formal proof of these security-related properties [18].

Authors in [19] have proposed a technique to formally verify malicious modification of critical data in 3PIP by hardware Trojans. The proposed technique is based on bounded model checking (BMC). Here, the BMC checks for the property—"does critical information get corrupted?" and outputs if the property is being violated in the given IP. Also BMC reports the sequence of input patterns which violates this property. Form the reported input patterns it is possible to extract the triggering condition of the Trojan. Another similar approach has been proposed in [20] which formally verifies unauthorized information leakage in 3PIPs. This technique checks for the property—"does the design leak any sensitive information?" The limitation of these approaches is that the processing ability of the model checking is relatively limited, due to the problem of space explosion.

Structural Analysis Structural analysis employs quantitative metrics to mark signals or gates with low activation probability as suspicious. In [21], authors have presented a metric named "Statement Hardness" to evaluate the difficulty of executing a statement in the RTL code. Areas in a circuit with large value of "Statement Hardness" are more vulnerable to Trojan insertion. At gate level, an attacker would most likely target *hard-to-detect* areas of the gate-level netlist to

insert Trojan. *Hard-to-detect* nets are defined as nets which have low transition probability and are not testable through well-known fault testing techniques (stuck-at, transition delay, path delay, and bridging faults) [22]. Inserting a Trojan in *hard-to-detect* areas would reduce the probability to trigger the Trojan and thereby, reduce the probability of being detected during verification and validation testing. In [23], authors have proposed metrics to evaluate *hard-to-detect* areas in the gate-level netlist. The limitations of code/structural analysis techniques are that they do not guarantee Trojan detection, and manual post-processing is required to analyze suspicious signals or gates and determine if they are a part of a Trojan.

Logic Testing Logic testing aims to activate Trojans by applying test vectors and comparing the responses with the correct results. While at first glance this is similar in spirit to manufacturing tests for detecting manufacturing defects, conventional manufacturing tests using functional/structural/random patterns perform poorly to reliably detect hardware Trojans [24]. Intelligent adversaries can design Trojans that are activated under very rare conditions, so they can go undetected under structural and functional tests during the manufacturing test process. In [25] authors have developed a test pattern generation methods to trigger such rarely activated nets and improve the possibility of observing Trojan's effects from primary outputs. However, this technique does not guarantee to trigger the Trojan and it is infeasible to apply this technique in an industrial-scale design. Additionally, there could be Trojans which do not functionally affect the circuit but leaks confidential information through side channel. This type of Trojans cannot be identified through logic testing.

Functional Analysis Functional analysis applies random input patterns and performs functional simulation of the IP to find suspicious regions of the IP which have similar characteristics of a hardware Trojan. The basic difference between functional analysis and logic testing is that logic testing aims to apply specific patterns to activate a Trojan, whereas functional analysis applies random patterns and these patterns are not directed to trigger the Trojan. Authors in [26] have proposed a technique named Functional Analysis for Nearly unused Circuit Identification (FANCI) which flags nets having weak input-to-output dependency as suspicious. This approach is based on the observation that a hardware Trojan is triggered under very rare condition. Therefore the logic implementing the trigger circuit of a Trojan is nearly unused or dormant during normal functional operation. Here, the authors have proposed a metric called "Control value" to find "nearly unused logic" by quantifying the degree of controllability of each input net has on its output function. "Control value" is computed by applying random input patterns and measuring the number of output transitions. If the control value of a net is lower than a predefined threshold, then the net is flagged as suspicious. For example, for the RSA-T100 (http://trust-hub.org/resources/benchmarks) Trojan, the triggering condition is *32'h44444444*. The "Control value" for the triggering net is 2^{-32} which is expected to be lower than the predefined threshold. The major limitations of FANCI are, this approach produces a large number of false positive results and this approach does not specify any method to verify if the suspicious

signals are performing any malicious operation. Also, authors in [27] have shown to design Trojans to defeat FANCI. Here, they design Trojan circuits whose trigger vector arrives over multiple clock cycles. For example, for the RSA-T100 Trojan, the triggering sequence can be derived over four cycles making the "Control value" of the triggering net 2^{-8}. Furthermore, FANCI cannot identify "Always On" Trojans which remains active during their lifetime and do not have any triggering circuitry.

Authors in [28] have proposed a technique called VeriTrust to identify potential triggering inputs of a hardware Trojan. The proposed technique is based on the observation that input ports of the triggering circuit of a hardware Trojan keep dormant during normal operation and therefore, are redundant to the normal logic function of the circuit. VeriTrust works as follows, first it performs functional simulation of the IP with random input patterns and traces the activation history of the inputs ports in the form of sums-of-products (SOP) and product-of-sums (POS). Veritrust then identifies redundant inputs by analyzing the SOPs and POSs which are unactivated during functional simulation. These redundant input signals are potential triggering inputs of a hardware Trojan. VeriTrust technique aims to be independent of the implementation style of hardware Trojan. However, this technique also produces a large number of false positive results because of incomplete functional simulation and unactivated entries belonging to normal function. Also, authors in [27] have designed Trojans which can defeat VeriTrust by ensuring that Trojan triggering circuit is driven by a subset of functional inputs in addition to triggering inputs. VeriTrust also shares the same limitation of FANCI as not being able to identify "Always On" Trojans.

Runtime Validation Runtime validation approaches are generally based on dual modular redundancy-based approaches [29]. These techniques rely on procuring IP cores of same functionality from different IP vendors. The basic assumption is that, it is highly unlikely that different Trojans in different 3PIPs will produce identical wrong outputs. Therefore, by comparing the outputs of IP cores of same functionality but obtained from different vendors, one can detect any malicious activity triggered by a Trojan. The main disadvantage of this approach is that the area overhead is prohibitively high ($\tilde{1}00\,\%$ area overhead) and the SoC integrator needs to purchase same functional IP from different vendors (economically infeasible).

4.5 A Case Study for IP Trust Verification

It is evident from the discussion on Sect. 4.4 that it is not possible to verify with high confidence the trustworthiness of a 3PIP using one single approach. Authors in [5] have proposed an IP trust validation technique which involves formal verification, coverage analysis, redundant circuit removal, sequential ATPG, and equivalence theorems. This technique first utilizes formal verification and code coverage analysis to identify any suspicious signals and components corresponding to Trojans in a

3PIP core. Next, the proposed technique applies suspicious signal analysis to reduce the number of false positive suspicious signals. This technique focuses on ensuring trust in soft IP cores, as they are the most dominant cores in the market today due to the flexibility they offer to the SoC designers.

4.5.1 Formal Verification and Coverage Analysis

One of the important concepts in this proposed technique is formal verification, an algorithmic-based approach to logic verification that exhaustively proves the functional properties of a design. It contains three types of verification methods that are not commonly used in the traditional verification, namely model checking, equivalence checking, and property checking. All functions in the specification are defined as properties. The specific corner cases in the test suite as they monitor particular objects in a 3PIP could also be represented by properties, such as worry cases, inter-block interfaces, and complex RTL structures. They can be represented as properties wherever the protocols may be misused, assumptions violated, or design intent incorrectly implemented. Formal verification uses property checking to check whether the IP satisfies those properties. With property checking, every corner of the design can be explored. For example, in benchmark RS232, there are two main functionalities in the specification: (1) transmitter and (2) receiver. Figure 4.3 shows the waveform of the transmitter. Take the *startbit* as an example; with $Rst == 1'b1$, *clk* positive edge, and $xmitH == 1'b1$, the output signal *Uart_xmit* will start to transmit *startbit* "0". This functionality is described using the SystemVerilog property shown in Fig. 4.4, and the corresponding assertion is defined simultaneously. The remaining items in the specification are also translated to properties during formal verification. Once all the functionalities in the specification are translated to properties, coverage metrics can help identify suspicious parts in the 3PIP under authentication. Those suspicious parts may be Trojans (or part of Trojans).

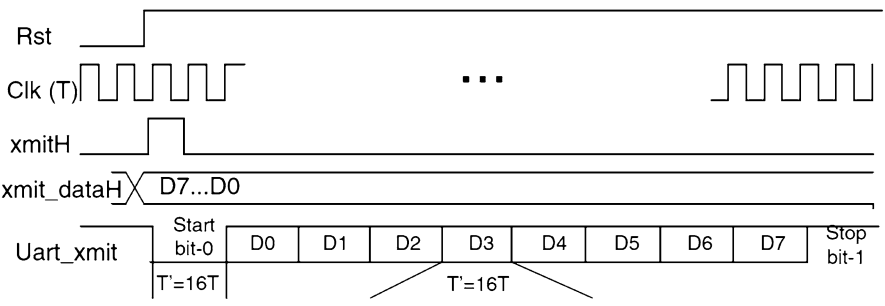

Fig. 4.3 Transmitter property in the specification stage

```
01:   property e1;
02:     @(posedge uart_clk) disable iff (Rst)
03:     $rose(xmitH) |− > ##1 (uart_XMIT_dataH==0);
04:     endproperty
05:
06:     a1: assert property( e1 );
```

Fig. 4.4 One of the properties and assertions definitions for RS232

01: Line No	Coverage	Block Type
02: 69	1	ALWAYS
03: 70	1	CASEITEM
04: 71	1	CASEITEM
05: 72	0	CASEITEM
06: 73	1	CASEITEM
07: 74	0	CASEITEM
08: 82	1	ALWAYS
09: 82.1	1	IF
... ...	...	...

Fig. 4.5 Part of the line coverage report

Coverage metrics include code coverage and functional coverage. Code coverage analysis is a metric that evaluates the effectiveness of a test bench in exercising the design [30, 31]. There are many different types of code coverage analysis, but only a few of them are helpful for IP trust, namely line, statement, toggle, and finite state machine (FSM) coverage. Toggle coverage reports whether signals switch in the gate-level netlist while the other three coverage metrics show which line(s) and statement(s) are executed, and whether states in FSM are reached in RTL code during verification. Figure 4.5 shows parts of line coverage report during the simulation with RS232. This report shows that lines 72 and 74 are not executed, which helps improve the test bench by checking the source code. If the RTL code is easily readable, special patterns that can activate those lines will be added to the test bench. Otherwise, random patterns will be added to verify the 3PIP.

Functional coverage is the determination of how much functionality of the design has been exercised by the verification environment. The functional requirements are imposed on both the design inputs and outputs and on their interrelationships by the design specifications from SoC designer (i.e., IP buyers). All the functional requirements can be translated as different types of assertion, as in Fig. 4.4. Functional coverage checks those assertions to see whether they are successful or not. Table 4.1 shows part of the assertions coverage report (Assertion a1 is defined in Fig. 4.4). The number of attempts in the table means that there are 500,003 positive edge clocks during the simulation time when the tool tries to check the assertion.

Table 4.1 Part of the assertion report with RS232

Assertion	Attempts	Real success	Failure	Incomplete
test.uart1.uart_checker.a1	500,003	1953	0	0
test.uart1.uart_checker.a2	1953	1952	0	1
...	...	...	...	...

In Table 4.1, the Real Success column represents the assertion success rate while Failure/Incomplete column denotes the frequency of assertion failure/incomplete. When there are zero Failures, the property is always satisfied.

If all the assertions generated from the specification of the 3PIP are successful and all the coverage metrics such as line, statement, and FSM are 100 %, then it can be assumed with high confidence that the 3PIP is Trojan-free. Otherwise, the uncovered lines, statements, states in FSM, and signals are considered suspicious. All the suspicious parts constitute the suspicious list.

4.5.2 Techniques for Suspicious Signals Reduction

Based on the formal verification and coverage metric, a flow is proposed to verify the trustworthiness of 3PIP in [1]. The basic idea of the proposed solution is that without redundant circuit and Trojans in a 3PIP, all the signals/components are expected to change their states during verification and 3PIP should function perfectly. Thus, the signals/components that stay stable during toggle coverage analysis are considered suspicious, as Trojan circuits do not change their states frequently. Each suspicious signal is then considered as the *TriggercEnablex*. Figure 4.6 shows the flow to identify and minimize the suspicious parts, including test pattern generation, suspicious signal identification, and suspicious signal analysis. Each step in the figure will be discussed in detail in the following.

4.5.2.1 Phase 1: Test Bench Generation and Suspicious Signal Identification

In order to verify the trustworthiness of 3PIPs, 100 % coverage of the test bench is best. However, it is very difficult to achieve 100 % coverage for every 3PIP, especially those with tens of thousands of lines of code. In the flow, the first step is to improve the test bench to obtain a higher code coverage with an acceptable simulation runtime. With each property in the specification and basic functional test vectors, formal verification reports line, statement, and FSM coverage for the RTL code. If one of the assertions fails even once during verification, the 3PIP is considered untrustworthy, containing Trojans or bugs. If all the assertions are successful and the code coverage is 100 %, the 3PIP can be trusted. If at least one

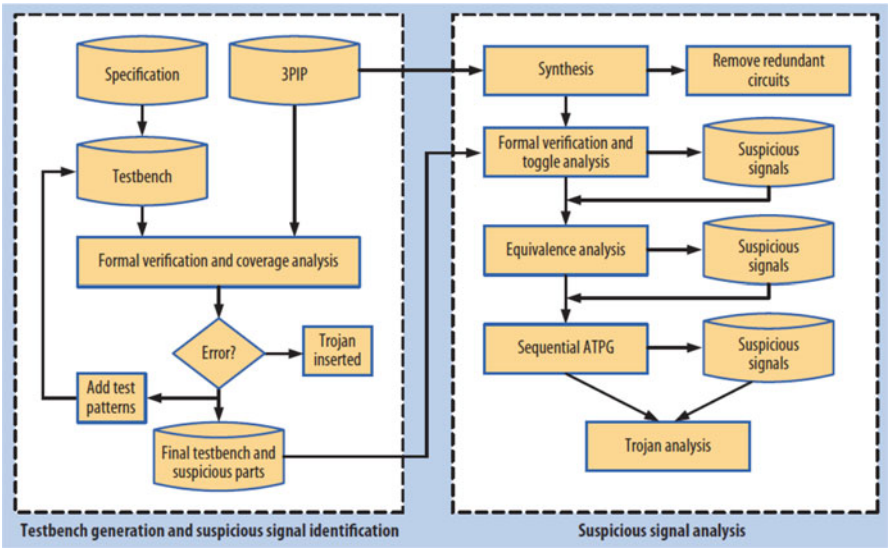

Fig. 4.6 The proposed flow for identifying and minimizing suspicious signals

assertion fails or the code coverage is less than 100 %, more test vectors need to be added to the test bench. The basic purpose of adding new vectors is to activate the uncovered parts as much as possible. But the verification time will increase as the number of test vectors increases. With the acceptable verification time and certain coverage percentage, both defined by the IP buyer, the final test bench will be generated and the RTL source code will be synthesized for further analysis.

4.5.2.2 Phase 2: Suspicious Signals Analysis

Redundant Circuit Removal (RCR) Redundant circuits must be removed from the suspicious list since they also tend to stay at the same logic value during verification, and input patterns cannot activate them. Removing a redundant circuit involves sequential reasoning, SAT-sweeping, conflict analysis, and data mining. The SAT method integrated in the Synopsys Design Compiler (DC) is used in this flow.

Another method to remove redundant circuits is developed in [5]. Scan chains are inserted into the gate-level netlist after synthesis for design testability, and ATPG generates patterns for all the stuck-at faults. The untestable stuck-at faults during ATPG are likely to be redundant logic. The reason is that if the stuck-at fault is untestable, the output responses of the faulty circuit will be identical to the output of the fault-free circuit for all possible input patterns. Thus, when ATPG identifies a stuck-at-1/0 fault as untestable, the faulty net can be replaced by logic 1/0 in the

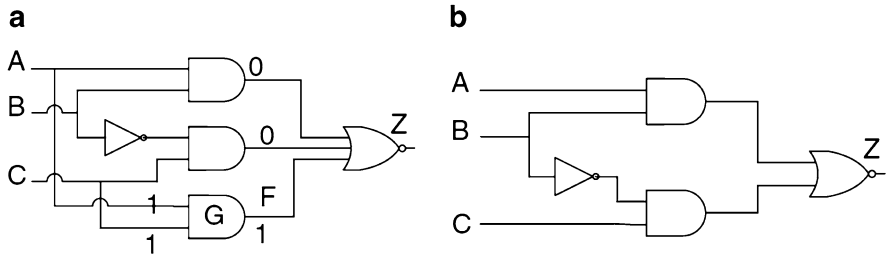

Fig. 4.7 (**a**) Before removing the redundant circuit with untestable F stuck-at-0 fault and (**b**) After removing the redundant circuit

gate-level netlist without scan-chain. All the circuits driving the faulty net will be removed as well. Figure 4.7a shows the circuit before redundant circuit removal.

The stuck-at-0 fault of net F is untestable when generating patterns. Net F will be replaced by 0 and the gate G driving it will be removed from the original circuit, as shown in Fig. 4.7b.

After redundant circuit removal, toggle coverage analysis for the gate-level netlist without scan chain will identify which signals do not toggle (also called quiet signal) during verification with the test bench generated in Phase 1. These signals will be considered suspicious and added to the suspicious list. By monitoring these suspicious signals during verification, the authors obtain the logic value those signal are stuck at.

Equivalence Analysis Fault equivalence theorems are known to reduce the number of faults during ATPG [32]. Similarly, the authors develop suspicious signal equivalence theorems to reduce the number of suspicious signals in [5].

- **Theorem 1.** *If signal A is the D pin of a flip-flop (FF) while signal B is the Q pin of the same FF, the quiet signal A makes signal B quiet. Thus signal A is considered equal to B, which means if the pattern that can activate A is found, it will activate B as well. Then signal B will be removed from the suspicious signal list. As the QN port of an FF is the inversion of the Q port, they will stay quiet or switch at the same time. Thus the suspicious signal B would be considered equal to A and should be removed from the suspicious list.*
- **Theorem 2.** *If signal A is the output pin of an inverter while signal B is its input, they will stay quiet or switch at the same time. Thus the suspicious signal B would be considered equal to A and should be removed from the suspicious list.*
- **Theorem 3.** *One of the inputs of AND gate A stuck-at-0 will cause the output B to stay quiet and one of the inputs of OR gate C stuck-at-1 will make the output D high all along. Thus, for AND gate, B stuck-at-0 is identical to A stuck-at-0, while for OR gate, D is identical to C stuck-at-1.*

Sequential ATPG After reducing the number of suspicious signals by applying the above equivalence theorems, the authors use sequential ATPG to generate special patterns to change the value of certain signals during simulation in [5]. Stuck-at

faults are targeted by the sequential ATPG to generate a sequential pattern to activate the suspicious signals when applied to the 3PIP. If the 3PIP functions perfectly with this pattern, the activated suspicious signals are considered part of the original circuit. Otherwise, there must be malicious inclusion in the 3PIP.

4.6 Simulation Results

The flow is applied to the RS232 circuit. Nine Trojans designed by authors in [5] and ten Trojans from (http://trust-hub.org/resources/benchmarks) are inserted into the 3PIP (RS232). In total, there are 19 RS232 benchmarks with one Trojan in each IP. The following section presents the simulation setup and test bench analysis for the 19 Trojan-inserted benchmarks. Next, the results of redundant circuit removal and the reduction of suspicious signals will be presented. Finally, Trojan coverage analysis will be discussed.

4.6.1 Benchmark Setup

The specification, structure, and functionality of the Trojans designed by authors in [5] are discussed below.

Trojan 1 The trigger of Trojan 1 is a special input sequence *8'ha6–8'h75–8'hc0–8'hff* . The payload changes the FSM in the transmitter of RS232 from state *Start* to *Stop*, which means that once the Trojan is triggered, RS232 will stop transmitting data ($outputdata = 8'h0$). Since the trigger of the Trojan is a sequence of four special inputs, the probability of detecting the Trojan during verification is $1/2^{32}$. If the baud rate is 2400 and RS232 transmits 240 words in one second, it will take 207.2 days to activate the Trojan and detect the error. In other words, it would be practically impossible to detect it by conventional verification. When this Trojan is inserted into RS232, an FSM is used to describe the Trojan input sequence. A three-bit variable state represents the FSM.

Trojan 2 This Trojan only adds four lines to the original RTL code. If the transmitting word is odd and the receiving word is *8'haa*, RS232 will stop receiving words. This Trojan is less complex compared to Trojan 1, however, it provides opportunities to demonstrate the effectiveness of each step of the proposed flow.

Trojan 3 The trigger of Trojan 3 is the same as that of Trojan 1, but the payload is different. Trojan 1 changes the state machine while Trojan 3 changes the shift process. The eighth bit of the transmitting word will be replaced by a Trojan bit during transmission. The Trojan bit could be authentication information, the special key to enable the system, or other important information.

Trojan 4 Trojan 4 is designed to act like a time bomb. A counter is inserted into RS232 to count the number of words that have been sent out. After sending *10'h3ff* words, the Trojan will be activated. The sixth bit of the transmitting word will be replaced by a Trojan bit.

Trojan 5 After *24'hffffff* positive edge clock, this Trojan's enable signal will become high. The sixth bit of the transmitting word will be replaced by a Trojan bit.

Trojan 6 If RS232 receives "0" when the system is reset, the Trojan will be activated. The eighth bit of the transmitting word will be replaced by a Trojan bit.

Trojan 7 When the transmitter sends a word *8'h01* and the receiver receives a word *8'hef* at the same time, the Trojan will be activated. A Trojan bit will replace the first bit of the transmitting word.

Trojans 8 and 9 These Trojans do not tamper the original function of RS232 but add extra one stage (Trojan 8) and three stage (Trojan 9) ring oscillator to the RTL, which will increase the temperature of the chip quickly if they get activated.

4.6.2 Impact of Test Bench on Coverage Analysis

All the items in the specification are translated into properties and defined as assertions in the test bench. Assertion checkers will verify the correctness of assertions by SystemVerilog. Another important feature of a test bench is the input patterns. Some test corners need special input patterns. The more input patterns in the test bench, the more, for example, lines will be covered during verification. Table 4.2 shows five test benches with different test patterns and verification times for various coverage metric reports for the RS232 benchmark with Trojan 1. Generally, the verification time will increase with more test patterns and the code

Table 4.2 Analyzing the impact of the test bench on coverage metrics (a benchmark with Trojan 1 is used)

Test bench #	Test bench 1	Test bench 2	Test bench 3	Test bench 4	Test bench 5
Test patterns #	2000	10,000	20,000	100,000	1,000,000
Verification time	1 min	6 min	11 min	56 min	10 h
Line coverage (%)	89.5	95.2	98.0	98.7	100
FSM state coverage (%)	87.5	87.5	93.75	93.75	100
FSM transition coverage (%)	86.2	89.65	93.1	96.5	100
Path coverage (%)	77.94	80.8	87.93	97.34	100
Assertion	Successful	Successful	Successful	Successful	Failure

coverage will be higher as well. For Test Bench 1 to Test Bench 4, all the coverage reports are less than 100 % and all the assertions are successful, which indicates that the Trojan is dormant during the entire verification. The special test patterns added in Test Bench 5 increase the pattern count significantly and can activate the Trojans inserted in the benchmark. Hundred percent code coverage could be achieved with these additional test patterns. If one of the assertion experiences a failure, it signifies Trojan activation and the RS232 will give an erroneous output. One can conclude that the IP is Trojan-inserted. However, it is not easy to generate a test bench with 100 % code coverage for large IPs, and the verification time will be extremely long.

This phase of the flow can help improve the quality of the test bench. Given the time-coverage trade off, Test Bench 4 is selected for further analysis.

4.6.3 Reducing the Suspicious Signals

All the 19 benchmarks with different Trojans are synthesized to generate the gate-level netlist. The removal of redundant circuits is done during the synthesis process with special constrains using the Design Compiler. The simulation results are shown in Table 4.3. The second column in the table shows the area overhead of each Trojan after generating the final layout. As the table shows, Trojans are composed of different sizes, gates, and structures, as well as different triggers and payloads as previously mentioned. The smallest Trojan has only 1.15 % area overhead. The percentage of Trojan area covered by suspicious signals *SS-Overlap-Trojan* is obtained by,

$$\text{SS-Overlap-Trojan} = \frac{N_{\text{SS}}}{N_{\text{TS}}}$$

where N_{SS} is the number of suspicious signals and N_{TS} is the number of Trojan signals. The results in Table 4.3 show that *SS-Overlap-Trojan* is between 67.7 % and 100 %, as shown in seventh column.

If all the suspicious signals are part of the Trojan, the *SS-Overlap-Trojan* would be 100 %. This indicates that the number of signals in the final suspicious list fully overlapped with those from Trojan. This is an indicator of how successful the flow is at identifying Trojan signals. In addition, if the Trojan is removed or detected by sequential ATPG, the *SS-Overlap-Trojan* would also be 100 %.

Test Bench 4 is used to verify which signals in each Trojan-inserted circuit are not covered by the simulation with all the successful assertions. Those quiet signals are identified as suspicious. The number of suspicious signals of each benchmark is shown in the third column of Table 4.3. Different benchmarks have a different number of suspicious signals based on the size of its Trojans. The larger the Trojan is, the more suspicious signals it has. On the other hand, the suspicious signals' stuck-at values are monitored by verification. All stuck-at-faults are simulated by the ATPG tool with scan chain in the netlist. If the fault is untestable, the suspicious

Table 4.3 Suspicious signal analysis

Benchmark (RS232)	Trojan area overhead (%)	Step 1: Number of SS after RCR with synthesis	Step 2: Number of SS after RCR with ATPG	Step 3: Number of SS after equivalence analysis	Step 4: Number of SS after sequential ATPG	SS-Overlap-Trojan (%)
With Trojan 1	11.18	22	20	17	12	100
With Trojan 2	20.35	17	16	3	Trojan is identified	100
With Trojan 3	10.48	20	15	15	10	97.3
With Trojan 4	20.35	3	3	3	2	87.6
With Trojan 5	4.59	9	8	8	7	100
With Trojan 6	1.15	1	1	1	Trojan is identified	100
With Trojan 7	3.79	3	3	3	2	100
With Trojan 8	1.15	1	Trojan is removed	–	–	100
With Trojan 9	3.79	3	Trojan is removed	–	–	100
TR04C13PI0	1.6	8	3	3	3	100
TR06C13PI0	1.8	9	3	3	3	100
TR0AS10PI0	2.09	8	1	1	1	100
TR0CS02PI0	25.3	59	55	39	39	67.7
TR0ES12PI0	2.09	8	1	1	1	100
TR0FS02PI0	25.0	30	28	20	20	73.3
TR2AS0API0	11.9	19	18	11	11	100
TR2ES0API0	12.0	20	18	11	11	100
TR30S0API0	12.4	22	20	13	13	93.6
TR30S0API1	12.3	25	22	14	14	87.3

circuit is a redundant circuit and will be removed from the original gate level netlist, in addition to the gates that drive the net. The number of suspicious nets after redundant circuit removal is shown in the fourth column of Table 4.3. As can be seen in the table, the suspicious nets of benchmarks with Trojan 8 and Trojan 9 are zero, which means that if the redundant circuits are removed in the two benchmarks, the benchmarks will be Trojan-free. The reason that redundant circuit removal can distinguish Trojans is that some Trojans are designed without payload and have no impact on circuit functionality. Thus it can be concluded that such Trojans can be removed by redundant circuit removal.

The remaining suspicious nets of each benchmark are needed to be processed by equivalence analysis and sequential ATPG. The fifth and sixth columns in Table 4.3 show the number of suspicious signals after the first two steps. It can be concluded that equivalence analysis can reduce a large number of suspicious signals, and sequential ATPG can be effective as well. For benchmarks with Trojan 2 and Trojan 6, the sequential ATPG can generate sequential patterns for the stuck-at faults in the suspicious signal. The sequential test patterns improve the test bench and increase its coverage percentage. Even though the coverage percentage is not 100 %, some assertions experience failure during simulation. Thus, the benchmarks with Trojan 2 and Trojan 6 are identified as Trojan-inserted.

The flow is implemented on ten trust benchmarks from the Trust-Hub (http://trust-hub.org/resources/benchmarks) and the results reported in rows 11–20 in Table 4.3 show that the presented flow can effectively reduce the total number of suspicious signals. In addition, as shown in seventh column, there is a good overlap between the number of suspicious signals and the actual Trojan signals inserted into each benchmark. However, some benchmarks experience low *SS-Overlap-Trojan*, such as RS232-TR0CS02PI0, since only part of this Trojan was activated during simulation.

4.6.4 Trojan Coverage Analysis

In the suspicious list, not all of signals are a result of Trojans. However, the *TriggerEnable* signal must be in the suspicious list if the IP contains a Trojan. Once one net is identified as part of a Trojan, it can be concluded that the 3PIP is Trojan-inserted. All the gates driving this net are considered to be Trojan gates. Figure 4.8 shows that the percentage of Trojan signals in the suspicious list increases significantly with the flow. As the authors apply different steps (step 1–4) to the benchmarks, 72 %, on average, of the suspicious signals are of the result of Trojans after redundant circuit removal with synthesis and ATPG in the 19 benchmarks. However, the percentage increases to 85.2 % when equivalence analysis is done and 93.6 % of signals in the suspicious signal list come from Trojans after sequential ATPG is applied to these benchmarks.

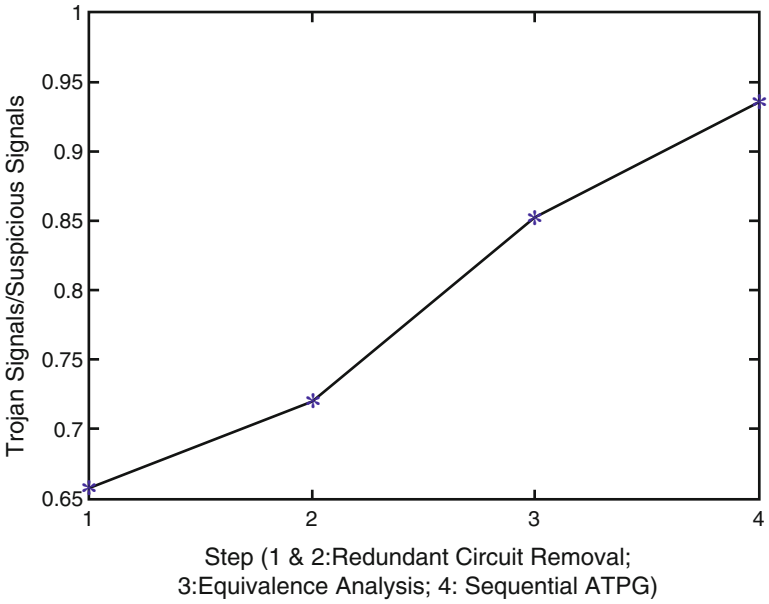

Fig. 4.8 Average Trojan signals/suspicious signals in 19 benchmarks

4.7 Conclusion

In this chapter, we have presented a technique to verify the trustworthiness of 3PIPs. This technique involves formal verification, coverage analysis, redundant circuit removal, sequential ATPG, and equivalence theorems. The code coverage generates the suspicious signals list. Redundant circuit is removed to reduce the number of suspicious signals. Equivalence theorems are developed for the same purpose. Sequential ATPG is used to activate these suspicious signals and some Trojans will be detected. However, more work is needed to get 100 % hardware Trojan detection rates in 3PIPs.

References

1. Report of the Defense Science Board Task Force on High Performance Microchip Supply, Defense Science Board, US DoD (2005), http://www.acq.osd.mil/dsb/reports/2005-02-HPMSi_Report_Final.pdf
2. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection. IEEE Des. Test Comput. **27**(1), 10–25 (2010)
3. M. Tehranipoor, C. Wang, *Introduction to Hardware Security and Trust* (Springer, New York, 2011)

4. H. Salmani, X. Zhang, M. Tehranipoor, *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection* (Springer, Cham, 2013)
5. X. Zhang, M. Tehranipoor, Case study: detecting hardware Trojans in third-party digital IP cores, in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (2011)
6. VSI Alliance, VSI Alliance Architecture Document: Version 1.0 (1997)
7. DIGITIMES. Trends in the global IC design service market (2012). Retrieved from http://www.digitimes.com/news/a20120313RS400.html?chid=2
8. M. Tehranipoor, et al., Trustworthy hardware: Trojan detection and design-for-trust challenges. Computer **44**(7), 66–74 (2011)
9. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research. ACM Trans. Des. Autom. Electron. Syst. **22**(1), Article 6 (2016)
10. M. Bilzor, T. Huffmire, C. Irvine, T. Levin, Evaluating security requirements in a general-purpose processor by combining assertion checkers with code coverage, in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (2012)
11. C. Sturton, M. Hicks, D. Wagner, S. King, Defeating UCI: building stealthy and malicious hardware, in *2011 IEEE Symposium on Security and Privacy (SP)* (2011), pp. 64–77
12. C. Cadar, D. Dunbar, D.R. Engler, Klee: unassisted and automatic generation of high-coverage tests for complex systems programs, in *Proceedings of the 2008 USENIX Symposium on Operating Systems Design and Implementation* (2008)
13. A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu, Symbolic model checking using SAT procedures instead of BDDs, in *Proceedings of the ACM/IEEE Annual Design Automation Conference* (1999), pp. 317–320
14. A.C. Myers, B. Liskov, A decentralized model for information flow control, in *Proceedings of the 1997 Symposium on Operating Systems Principles* (1997)
15. E. Love, Y. Jin, Y. Makris, Proof-carrying hardware intellectual property: a pathway to trusted module acquisition. IEEE Trans. Inf. Forensics Secur. **7**(1), 25–40 (2012)
16. Y. Jin, B. Yang, 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)
17. G. Xiaolong, R.G. Dutta, Y. Jin, F. Farahmandi, P. Mishra, Pre-silicon security verification and validation: a formal perspective, in *Proceedings of the 52nd Annual Design Automation Conference* (ACM, New York, 2015), p. 145
18. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
19. J. Rajendran, V. Vedula, R. Karri, Detecting malicious modifications of data in third-party intellectual property cores, in *Design Automation Conference (DAC)* (2015)
20. J. Rajendran, A.M. Dhandayuthapany, V. Vedula, R. Karri, Formal security verification of third party intellectual property cores for information leakage, in *29th International Conference on VLSI Design* (2016)
21. H. Salmani, M. Tehranipoor, Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level, in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (2013), pp. 190–195
22. H. Salmani, R. Karri, M. Tehranipoor, On design vulnerability analysis and trust benchmarks development, in *Proceedings of IEEE 31st International Conference on Computer Design (ICCD)* (2013), pp. 471–474
23. M. Tehranipoor, H. Salmani, X. Zhang, *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection* (Springer, Cham, 2013)
24. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
25. R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: a statistical approach for hardware Trojan detection, in *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'09)* (2009)

26. A. Waksman, M. Suozzo, S. Sethumadhavan, FANCI: identification of stealthy malicious logic using Boolean functional analysis, in *Proceedings of the ACM Conference on Computer and Communications Security* (2013), pp. 697–708
27. J. Zhang, F. Yuan, L. Wei, Z. Sun, Q. Xu, VeriTrust: verification for hardware trust, in *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference* (2013), pp. 1–8
28. J. Zhang, F. Yuan, Q. Xu, DeTrust: defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans, in *Proceedings of the ACM Conference on Computer and Communications Security* (2014), pp. 153–166
29. J. Rajendran, O. Sinanoglu, R. Karri, Building trustworthy systems using untrusted components: a high-level synthesis approach. IEEE Trans. Very Large Scale Integr. VLSI Syst. **24**(9), 2946–2959 (2016)
30. Synopsys, The Synopsys Verification Avenue Technical Bulletin, vol. 4, issue 4 (2004)
31. I. Ugarte, P. Sanchez, Formal meaning of coverage metrics in simulation-based hardware design verification, in *IEEE International High Level Design Validation and Test Workshop (HLDVT)* (IEEE, Napa Valley, 2005)
32. M. Bushnell, V. Vishwani, *Essentials of Electronic Testing for Digital, Memory and Mixed Signal VLSI Circuits*, vol. 17 (Springer Science & Business Media, 2000)