Chapter 16
Design for Hardware Trust

Yier Jin, Eric Love, and Yiorgos Makris

16.1 Overview

Toward further enhancing the effectiveness of postfabrication hardware Trojan detection solutions and alleviating their limitations, as discussed in previous chapters, several methods which rely on modifying the current IC design flow have been developed by the hardware security and trust community. Collectively termed *design for hardware trust* [1], these Trojan prevention methods aim to prevent insertion and facilitate simple detection of hardware Trojans. In contrast to Trojan detection methods which passively test chips anticipating that the inserted Trojans will be identified based on their abnormal behavior, Trojan prevention methods take a proactive step by changing the circuit structure itself in order to prevent the insertion of Trojans. In order to achieve this goal, the entire IC supply chain needs to be revisited. The resulting modified IC supply chain emphasizes design security to counter Trojan threats and provide a solution for trusted IC design.

Among the proposed Trojan prevention methods, the majority aim to complement side-channel-based Trojan detection methods, by inserting non-functional circuitry during the chip design stage, to facilitate the construction of side-channel signatures. Hereafter, we refer to these methods as side-channel fingerprint-based hardware prevention methods. Some of them can only play an auxiliary role to Trojan detection methods and help in measuring internal side-channel information which is otherwise untestable using off-chip testing equipment. Others go further to not only measure internal side-channel information but also to compare the measured information with predefined threshold values using a special-purpose module which is embedded on-chip and which is responsible for deciding whether abnormality exists within the side-channel signals. Design overhead is the major concern of side-channel fingerprint-based Trojan prevention methods because

Y. Jin · E. Love · Y. Makris (✉)
Department of Electrical Engineering, Yale University, New Haven, CT 06520, USA
e-mail: yier.jin@yale.edu; eric.love@yale.edu; yiorgos.makris@yale.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_16,

side-channel measurement and analysis circuits can be rather complex and consume large on-chip area. Interestingly, for this reason, Trojan prevention methods of this kind typically select path delay as the basis for constructing a side-channel fingerprint of a chip, because delay measurement modules are light-weight in terms of overhead.

Other methods rely on the assumption that attackers will only use rare events to trigger the inserted Trojans. Such methods attempt to enhance traditional functional/structural testing methods by increasing the probability of fully activating inserted Trojans during the test stage. Among them, design obfuscation [2] conceals the circuit so that attackers cannot calculate the probabilities of real events. Alternatively, dummy scan flip-flops [3] and the inverted voltage scheme [4] aim to balance internal signal transition frequencies in order to remove rare events. Along a different direction and in order to avoid relying on such assumptions, a Design for Trojan Test (DFTT) methodology [5] is proposed as a general design hardening scheme.

Lastly, methods that target protection of Intellectual Property (IP) have recently started to appear in the literature. Realizing the importance of protecting third party IP, which constitute a significant portion of the design in most contemporary systems on a chip, the authors in [6, 7] proposed the concept of proof-carrying hardware (PCH), which is based on a well-developed formal software protection methodology, namely proof-carrying code (PCC).

16.2 Delay-Based Methods

16.2.1 Shadow Registers

Shadow registers constitute one of the path delay-based Trojan prevention methods which was first proposed in [8] and then carefully evaluated in [9]. Trojan detection based on path-delay fingerprints was first proposed in [10] where the authors showed that with the help of statistical data analysis, this method could effectively detect hardware Trojans as small as 0.2% of the total on-chip area, even when considering $\pm 7.5\%$ process variation. An obstacle preventing wide usage of this path delay-based Trojan detection method is the difficulty of measuring/observing delays of internal paths (i.e. from register to register but not connected to primary input/output). Without internal path delay information, it is almost impossible to construct the full path-delay fingerprint to which chips-under-test can then be compared. Shadow-register provides a possible solution to this problem by enabling a mechanism for measuring internal path delay.

Figure 16.1 shows the basic architecture of the shadow register Trojan prevention scheme [8]. From this architecture, it can be seen that the basic unit contains one *shadow register* one *comparator* and one *result register*. The *shadow register* is located in parallel with the *destination register* at which a path ends. Different from the *system clock*, shadow registers are controlled by a negative-skewed clock signal

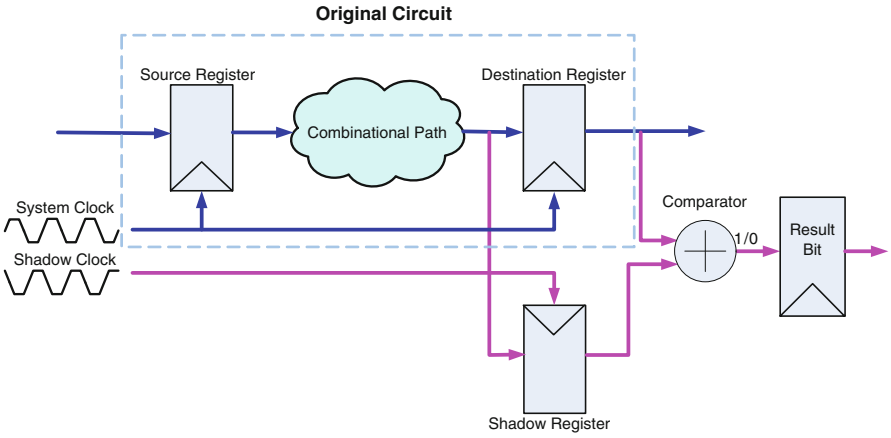

Fig. 16.1 Trojan prevention architecture with shadow registers [8]

the *shadow clock*. The frequency and phase of the *system clock* are kept constant in order to maintain the functionality of the original circuit even when in delay testing mode. The *shadow clock* has the same frequency as the *system clock* but its phase is adjustable. In order to perform internal path delay measurement, an off-chip clock signal generator or an on-chip Digital Clock Manager (DCM) is required. The precision of the signal generator or DCM decides the accuracy of the measured path delays. At the beginning of each internal path delay test, the phase of the *shadow clock* is reset to the same as that of the *system clock* so that the value in the *shadow register* is the same as the *destination register* and the output of the comparator is "0." Then the *shadow clock* is negatively skewed step by step according to the precision of the signal generator or DCM. The adjustment continues until the output of the comparator is "1" indicating that the value in the *shadow register* is now different from the value in the *destination register*. The output "1" is then stored in the *result register* and finally read out through a scan chain (not shown in the figure).

The shadow register architecture significantly improves the observability inside a chip to make it easier to construct path delay fingerprints both for genuine chips and for chips-under-test. Although the authors in [9] demonstrated effectiveness of this Trojan prevention method on a sample target circuit, i.e., an 8x8 Braun multiplier, this prevention scheme suffers from several inherent limitations. These limitations and possible solutions are discussed next.

1. The phase adjustment step size of the signal generator or DCM is critical to the accuracy of the measured delay and also related to the effectiveness of this method. A high-resolution on-chip DCM is preferable but it would consume large area and power.
2. The existence of process variation and measurement noise can easily deteriorate testing results. A popular way of solving this problem is to leverage statistical data analysis methods to deal with the measured delay information.

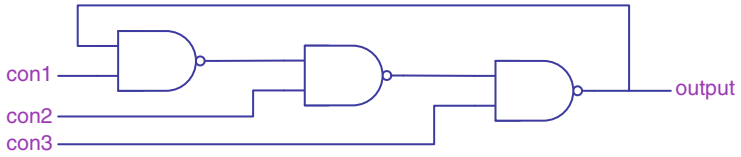

Fig. 16.2 Ring oscillator constructed by three NAND gates

3. As the scale of the original circuit increases, the total number of paths to be measured also increases. More test vectors should be applied to the design in order to cover a large number of paths and improve the testing coverage rate, but at higher testing cost. The continued increase of circuit size and complexity also means that more shadow registers are required to be inserted in the original design. These shadow registers are supported by the *shadow clock*. From a layout point of view, a design with two clock trees would cost large on-chip area and the chip performance may deteriorate due to inefficiencies in clock signal distribution.
4. The question of reading out the information in result registers triggers another concern regarding this Trojan prevention method. Embedding result registers into the original scan chain to reuse the scan chain controller is an efficient way to reduce the area of the prevention scheme, but it forces the chip into scan mode frequently by stopping the normal operation.

16.2.2 Ring Oscillators

In order to lower the testing cost raised by shadow registers but still leverage path delays to prevent the insertion of hardware Trojans, some researchers tried not to measure the existing path delays but insert new paths and measure the delay of these paths instead. For the following reasons, ring oscillators are among the most popular choices to construct the extra internal paths:

1. *Small area*: the area of ring oscillators is much smaller than that of other Trojan prevention architectures, so the total overhead is lower.
2. *Easy insertion*: As the architecture of the ring oscillator is very simple, it is easy for designers to insert ring oscillators inside circuit designs with low impact to the original design. For the same reason, it is easy to predict the behavior of the inserted ring oscillators even when considering large process variation.

Figure 16.2 shows a sample ring oscillator constructed with three NAND gates. Using NAND gates instead of inverters can improve the controllability of the ring oscillator. For this example, there are three control signals, *con1*, *con2*, and *con3*, to control oscillation. Only if all three signals are of high voltage could the ring oscillator start oscillating. Under normal operation, all the inserted ring

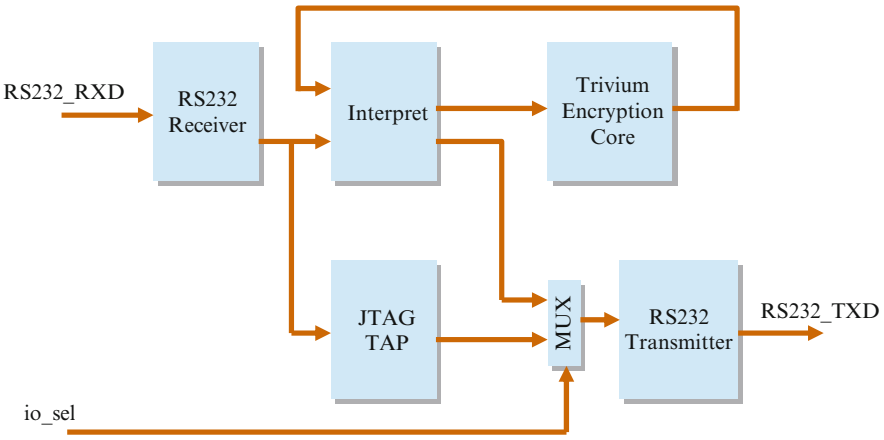

Fig. 16.3 Trivium-based encryption platform

oscillators will be muted to avoid power consumption; while in testing mode, switching the value of controlling signals allows the testers to turn on the selected ring oscillators and then read out the oscillation frequency. In addition to ensuring low power consumption, the added control signals can also prevent attackers from understanding the details of the inserted ring oscillators.

The main idea behind ring oscillator-based Trojan prevention methods is that any malicious modifications to the original design would also change parameters of pre-inserted ring oscillators. These changes include lower power supply voltage, higher input current, longer delay between NAND gates due to rewiring, etc. One question raised by this method is how many ring oscillators are needed and where they should be located inside the chip. The author in [11] uses an example to answer both questions where the target system contains a Trivium encryption core, an interpret module, an RS232 transceiver and a JTAG TAP, as shown in Fig. 16.3.

Three ring oscillators are inserted into the encryption platform: one in the RS232 Transceiver, one in the interpret module and a third in the JTAG TAP. The location selection covers both the datapath and the control logic and emphasizes the security weakness of input/output modules in the target platform. The expectation is that by carefully selecting insertion locations, one can lower the number of ring oscillators needed and increase the efficiency of the Trojan prevention scheme. However, this kind of location selection is solely based on designers' experience and their understanding of the target circuit, and could have limitations that attackers may use to circumvent the entire protection scheme. For example, the insertion of ring oscillators in the RS232 transceiver and JTAG TAP may be able to detect any modifications trying to leak information through the RS232 channel and prevent any malicious change in the control logic of scan chain. Yet malicious addition of a simple shortcut between the plaintext input and the RS232 output channel can easily evade this prevention method and leak the plaintext, as the latter is not protected by any of the three ring oscillators [12].

Another Trojan prevention scheme using ring oscillators, different from the method mentioned earlier, is to construct ring oscillators from gates of the original design by inserting multiplexors (MUXs), NAND gates, and inverters. Rather than inserting ring oscillators into the design in order to detect additional malicious circuitry in an indirect way, this method highly improves the sensitivity of the constructed ring oscillators because any modifications will directly change the internal architecture and result in a significant frequency change for the constructed ring oscillators. At worst, the insertion of malicious circuitry may mute the ring oscillators. The designers can also adjust the coverage rate (the percentage of the on-chip area which is part of the constructed ring oscillators) to control area overhead of the proposed method. To fully present the tradeoff between hardware security level and area overhead, a 4-bit carry look-ahead adder is chosen as the sample circuit in the 2010 CSAW Embedded System Challenge hosted by the Polytechnic University of NYU [13]. Contest organizers proposed three circuit hardening levels by constructing two, four, and six ring oscillators inside the adder. Among them, the low protection level (two ring oscillators) with two MUXs and two inverters covers 62% of the original design's gates (16 of 26 gates). The medium protection level (four ring oscillators) with four MUXs and four inverters covers 85% of the original design's gates (22 of 26 gates). The hard protection level (six ring oscillators) with six MUXs and six inverters covers 92% of the original design's gates (24 over 26 gates). Figure 16.4 shows the gate level architecture of the 4-bit carry look-ahead circuit and one sample ring oscillator constructed by an additional Inverter, an additional MUX and three gates from the original design (an XOR, an AND, and an OR gate). When the ring oscillator control signal *RO* is "0," the circuit performs its normal functionality. When it is set to "1," however, the ring oscillator starts oscillating. Testers can then measure the frequency of the constructed ring oscillators to decide whether the chip is genuine or not.

It is easy to construct internal ring oscillators in small-scale circuits such as the one shown in Fig. 16.4, but for more complex circuits, designers will have to rely on algorithms to automate the insertion process, hence such automation tools should also be developed. Research in this domain is still in progress and more efficient algorithms may be proposed in the future.

Although both ring oscillator insertion and ring oscillator construction are lightweight in terms of overhead, one concern with both methods is the ability to read out the frequency of these internal ring oscillators. Adding pins for each ring oscillator could be a straightforward solution, which can leverage the high precision of external test equipment to measure frequency. Considering the large number of ring oscillators, however, it is not an economic way to use or reuse valuable pin resources. In contrast, most current solutions use on-chip frequency measuring modules to accomplish this task. Counters are among the most popular frequency measuring modules. Figure 16.5 shows a sample measuring module which includes two separate counters, *Clock_Counter* and *RO_Counter* [14]. The global *RST* signal is used to reset both counters. The *Clock_Counter* has its clock pin connected to the system clock and counts the cycles of the system clock while the *RO_Counter* is connected to the output of the ring oscillator. The output of

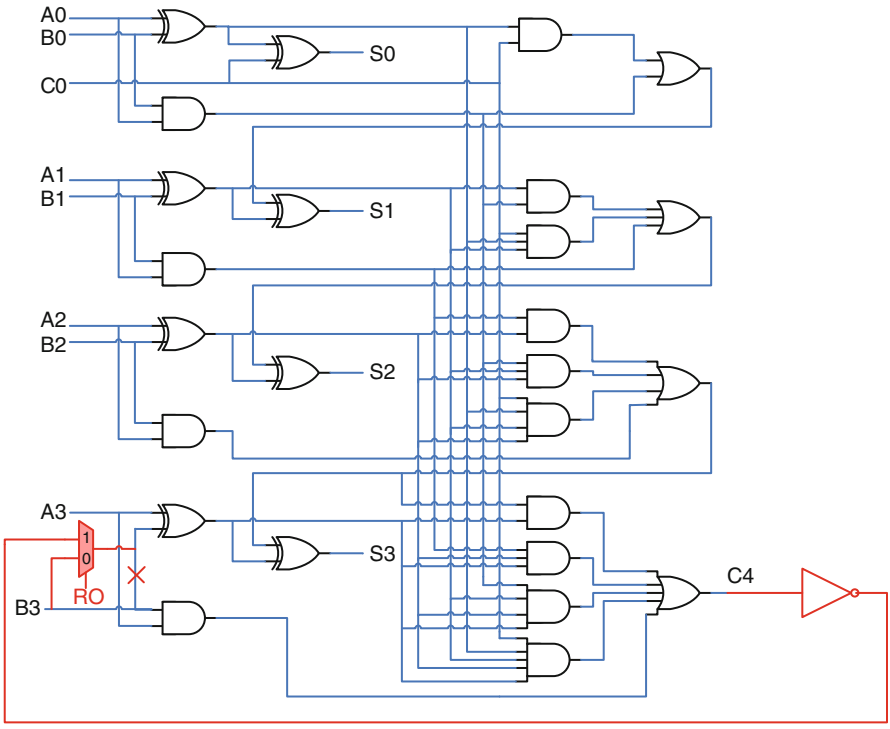

Fig. 16.4 Gate level circuit of 4-bit carry look-ahead adder and an inserted ring oscillator

Clock_Counter is compared to a predefined threshold value N to decide the ON/OFF state of *RO_Counter*. At the testing stage, the ring oscillator is enabled. Both counters start to count, but at different frequencies; one based on the system clock frequency and the other based on the frequency of the connected ring oscillator. When the output of *Clock_Counter* is equal to N, the *RO_Counter* stops counting and its output over N is a measure of a relative frequency to the system clock. Though quite compact, the added frequency measuring modules still increase the overhead of this Trojan prevention method.

All previously introduced hardware Trojan prevention methods try to prevent Trojan insertion by measuring internal path delays. None of them, however, pays attention to the security of the inserted testing circuit itself. As a result, the overall effectiveness of these methods may be lower than expected. The limitations of unsecure protection circuits have already been pointed out by various researchers, who have demonstrated the weakness of these methods by presenting successful hardware Trojan attacks that these prevention methods are unable to detect [11, 12, 14–17]. In fact, the focus of the embedded systems challenge at CSAW 2010 [13] was to find possible security limitations of the ring-oscillator Trojan prevention method. Reports from this competition conclude that currently proposed

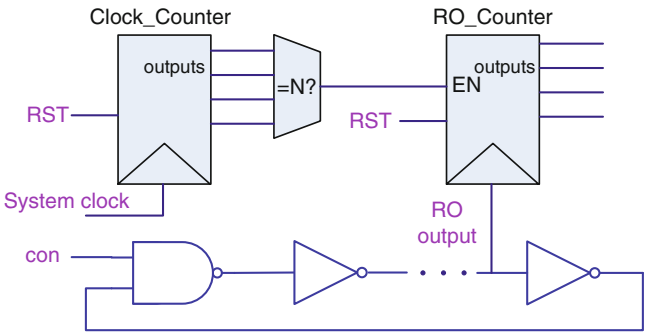

Fig. 16.5 Frequency measuring module for ring oscillator [14]

hardware prevention methods are far less secure than initially believed. More than 200 Trojan designs were submitted to this competition, and many of them succeeded by tampering with the Trojan prevention method.

16.3 Rare Event Removal

Besides the side-channel fingerprint-based Trojan prevention method, other researchers are trying to develop functional/structural Trojan prevention methods. In [18], the author conjectures that attackers will only choose rarely occurring events as triggers and proposes the idea of *Trojan vectors* which can trigger low-frequency events to enhance the detectability of traditional structural testing. Here the basis for launching Trojan attacks is the event probability because attackers will choose low-frequency events to trigger the inserted Trojans. Consequently, if a design hardening scheme increases the difficulty of calculating the event frequency and/or makes rare events scarce, attackers will have to randomly pick trigger events and the probability of the inserted Trojans being activated during the testing phase will increase, thereby deterring attackers from inserting Trojans inside the design. Design obfuscation [2], dummy scan flip-flops [3], and the inverted voltage scheme [4] are three representatives of functional/structural Trojan prevention methods.

Design obfuscation, by definition, means that a design will be transformed to another one which is functionally equivalent to the original, but in which it is much harder for attackers to obtain complete understanding of the internal logic, making reverse engineering much more difficult to perform. In [2, 15], the authors propose a Trojan prevention method that obfuscates the state transition function to add an *obfuscated mode* on top of the original functionality (called *normal mode*). Figure 16.6 shows the obfuscated functionality and normal functionality after the state transition function of the original design is obfuscated. As shown in the figure, the transition arc $K3$ is the only way the design can enter normal operation mode from the obfuscated mode. Thus, only one input pattern is able to guide the circuit

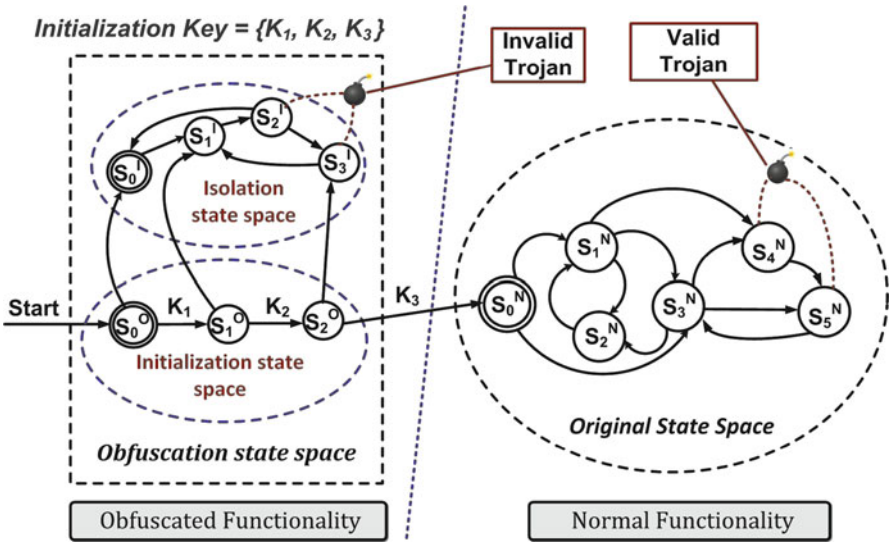

Fig. 16.6 Trojan prevention by design obfuscation [15]

into its normal mode. This special input pattern is called the *initialization key sequence*. Without knowing this key sequence, attackers are unlikely to get into normal mode by randomly picking input patterns, and the event probabilities derived from simulations cannot reflect their real probabilities if only running in normal operation (in order to prevent attackers from finding the initialization key sequence, the size of the obfuscation state space is designed to be very large). After design obfuscation, any inserted hardware Trojans can be divided into two groups, one with all (or part) of the trigger states from obfuscation mode and the other with all trigger states from normal mode. For the former group of Trojans, because the state space in obfuscated mode is unreachable in normal operation, these Trojans will never be triggered at all. For the later group of Trojans, while they are valid, the real event probabilities are likely different from what the attackers expect based on simulation, so it is not necessarily true that these events indeed occur rarely. These two groups of Trojans are labeled as *invalid Trojans* and *valid Trojans* in Fig. 16.6.

The design obfuscation method is easy to implement because it can leverage commercial EDA tools. An algorithm is proposed to automate the whole obfuscation process [2]. However, in many cases, the underlying assumptions of this hardware Trojan prevention method based on design obfuscation may not hold entirely true. One assumption here is that attackers will only use rare events to trigger the Trojan. This assumption omits the always-on hardware Trojans, and the definition of "rare" is rather arbitrary. Note that the other two Trojan prevention methods which we introduce later, namely dummy flip-flop insertion and inverted voltage, also suffer from this problem. Another assumption is that attackers have no knowledge of the obfuscation mechanism but only try to analyze the obfuscated version of the code to

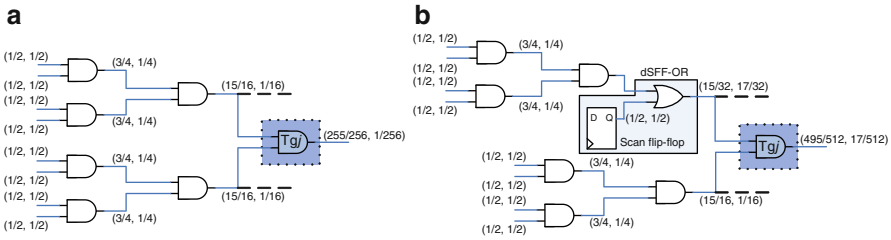

Fig. 16.7 Transition probability analysis on original circuit (**a**) and with dummy flip-flop (**b**) [3]

find rare events. This method assumes that attackers do not choose triggers from the space between exhaustive input patterns and ATPG patterns (plus rare events), even though this space is very large. If any of these assumptions are not valid in reality, the effectiveness of this entire Trojan prevention method will be diminished.

In [3], the authors first model the internal net transition probability using geometric distribution and calculate the value based on the number of clock cycles needed to generate a transition on a net. A Trojan prevention (or more accurately, a Trojan activation) methodology is then proposed which can increase the probability of generating a transition in functional Trojan circuits, if any exist, and analyze the transition generation time. In order to increase the activity of nets with low transition probability, extra flip-flops (called *dummy flip-flops*) are inserted into the original design. These dummy flip-flops are inserted in a way that does not change the design's original functionality. Figure 16.7 shows the procedure to calculate the net transition probability with and without dummy flip-flops. In Fig. 16.7a, the sample circuit contains seven AND gates located in three levels with eight inputs and one output. If it is assumed that, for each input, the probabilities of that input being "1" and "0" are equal, then due to the characteristics of AND gates, the probability of a "0" output is 255/256 while the probability of a "1" output is only 1/256, which is rather low. In Fig. 16.7b, a dummy flip-flop and an OR gate are inserted in one branch of the final AND gate and the probability of a "1" output is increased to 17/512, 8.5 times that of the original circuit.

This dummy flip-flop-based Trojan prevention method can help Trojan detection and Trojan prevention in two ways:

1. *Power-based side-channel analysis*: Due to the insertion of dummy flip-flops and related AND/OR gates, the activity of Trojans under ATPG-generated testing patterns will increase to consume more power during the testing stage. The increased ratio of Trojan power consumption to total power consumption will lead to an easy detection of inserted Trojans and will finally prevent attackers from inserting Trojans even if they know the design is protected by the dummy flip-flop protection scheme.
2. *Functional testing*: The inserted dummy flip-flops can balance the transition probabilities of internal nets so that the probability that the inserted Trojan is fully activated increases. The rare occurrence assumption, which attackers rely on to evade traditional functional testing, can be invalidated and erroneous responses may be observed at a primary output.

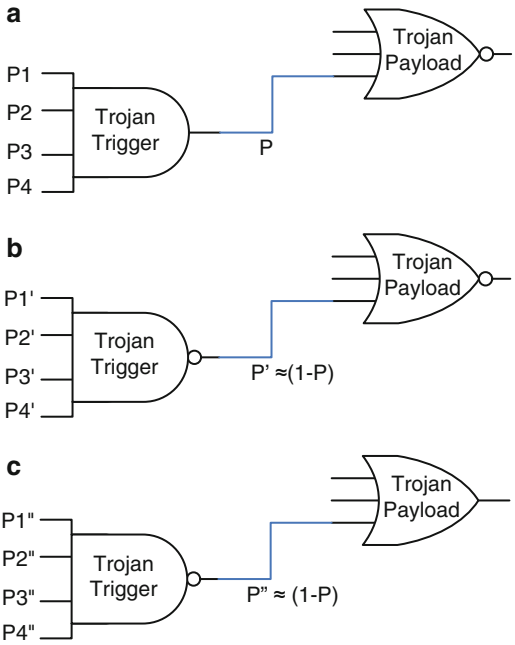

Fig. 16.8 Trojan circuit under normal voltage supply (**a**), only Trojan trigger is affected by inverted voltage supply (**b**), both Trojan trigger and payload are affected by inverted voltage supply (**c**) [4]

In [4], the authors proposed a supply voltage inversion method to magnify Trojan activity without inserting extra gates. The key idea here is that reversing a gate's power supply and ground changes the function of the gate to make low probability outputs occur more frequently. Figure 16.8 shows how this method can help to detect the inserted Trojan by switching the *majority value* and *minority value* of any internal gate [4]. The trigger of the sample Trojan is a four-input AND gate and $P1, P2, P3, P4$ are the probabilities of the four inputs being set to the non-controlling value 1, respectively. So the probability that output equals to its *minority value* 1 is approximately $P = P1 \times P2 \times P3 \times P4$. If the supply voltage to the Trojan trigger is inverted while the supply of the Trojan payload remains the same, the Trojan structure will be changed to that of Fig. 16.8b where the AND gate is converted into a NAND gate with the *majority value* being 1. Now the probability of triggering the Trojan is $P' \approx (1 - P)$. Figure 16.8c shows the case when the supply voltage of both the Trojan trigger and payload are inverted and the payload gate is converted to OR gate but the Trojan activation probability does not change.

One problem of supply voltage inversion in CMOS logic is that the degraded gate potential of one stage of CMOS gates degrades the gate potential on the next stage and the signal might stop propagating after a few stages. To avoid this degradation problem, the authors in [4] modified this method to apply inverted voltage to alternate stages in the original circuit.

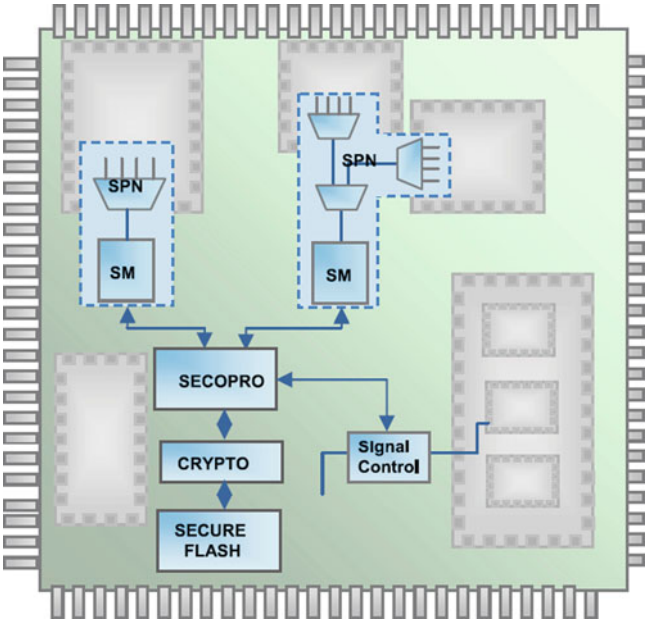

Fig. 16.9 SoC design with DEFENSE logic [19]

In [19], the authors proposed infrastructure logic to perform online security checks during normal operation, focusing on the System-on-Chip (SoC) domain. A reconfigurable logic called design-for-enabling-security (DEFENSE) is added to the SoC platform to implement real-time abnormality monitoring. Figure 16.9 shows the architecture of a hardened SoC chip with DEFENSE. The basic module of DEFENSE logic is the Signal Probe Networks (SPN) – Security Monitor (SM) pair. Here a signal probe network is a distributed pipelined MUX network configured to select a subset of user defined important signals and transport them to Security Monitors, programmable transaction engines configured to implement an FSM to check user-specified behavioral properties of the signals from SPNs. The Security and Control Processor (SECOPRO) reconfigures SPNs to dynamically reselect monitoring signals. All the configurations are encrypted and stored in secure flash memory so that their function is not visible to reverse engineering.

When signal abnormality is detected, the signal control module enables SECOPRO to override the value of any suspicious signals and restore the whole system back to normal operation. The core exhibiting illegal behavior may also be isolated. While effective, the overhead of this method remains is a concern because high coverage of on-chip signals requires a large number of nets to be monitored. Fairly sizeable on-chip area will also be occupied by the DEFENSE logic.

16.4 Design for Trojan Test

The authors in [5] developed a general hardening scheme which follows the paradigm of the widely accepted structural fault testing methodology, Design for Test (DFT). As the target of this method is to prevent attackers from inserting hardware Trojans into circuit designs, it has been termed DFTT. Despite the naming similarity, the DFTT methodology has some key differences from DFT. For DFT methods, test vectors are generated based on the assumption that the CUT (circuit-under-test) is genuine, with no inserted malicious circuits, because the purpose of DFT is to detect manufacturing faults. However, for DFTT, the goal is to generate test vectors with the objective of detecting maliciously inserted circuits.

The central idea of DFTT methods is that any effective hardware Trojans must impose a specific structure on the infected circuit, which the attacker leverages to leak internal information. While this structure is not known to circuit designers, scrutiny of all active logic on-chip with the help of local probing cells may be sufficient to reveal its existence and, thereby, expose the hardware Trojan. This method is robust because even if attackers have a complete picture of how this scrutiny works, it is still difficult to evade. Performing DFTT on the original design is known as *hardening* a design. Figure 16.10 shows the three basic steps of DFTT, which are explained later.

16.4.1 Step I: Code Evaluation

DFTT coding rules are first developed through which the original Hardware Description Language (HDL) code is converted to DFTT-compliant code.

16.4.2 Step II: Sensitive Path Selection

An assumption is made here that the attacker's purpose is to insert an additional structure in the original design to reliably steal internal sensitive information. For this purpose they would attempt to evaluate the relative merit of internal signals (such as the encryption key in a cryptographic chip) before inserting Trojans. Hence, the DFTT tool (developed in parallel with the DFTT methodology to automate the whole DFTT procedure) isolates paths in which sensitive signals, or other signals which are auxiliary to sensitive signals, flow from primary inputs to primary outputs.

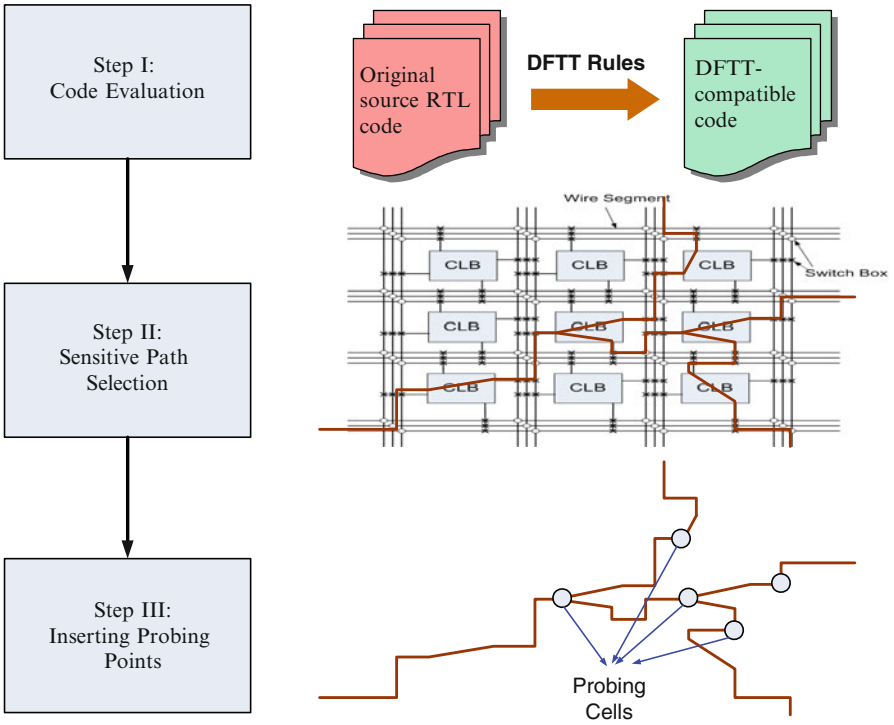

Fig. 16.10 The basic procedure of hardening a design via the DFTT methodology [5]

16.4.3 Step III: Inserting Probing Points

Based on the sensitive paths chosen in Step II, probe cells are inserted into the DFTT-compliant code. This step is similar to the insertion of scan flip-flops (SFFs) when performing DFT, but the probe cell is slightly different from a normal SFF because of the emphasis on two key characteristics: *genuineness* and *integrity* [5].

After the design is hardened using the DFTT methodology, the subsequent testing process is similar to that used in DFT. DFT-style trigger-response pairs generated by DFTT tools are loaded into the CUT. Assuming that there are no manufacturing faults within the design, any mismatch between the CUT's response sequence and the genuine response sequence reveals that internal logic has been modified. Reverse engineering or other related testing methods can then be performed to further analyze the suspicious chips.

16.5 Proof-Carrying Hardware

In [20], the authors raised a limitation of existing Trojan detection methods, namely the fact that they all try to scrutinize ICs for the presence of Trojans in the post-silicon stage. Far less is known or has been researched regarding this problem in pre-silicon stages. Unfortunately, the same problem exists in the design for hardware trust domain. All the previously introduced Trojan prevention methodologies are based on the assumption that any inserted Trojans will be detected relatively easily in the post-silicon stage if it is simple for designers to measure side-channel signals or hard for attackers to find true probability of internal events. None of these methods have tried to protect third-party hardware IP from hardware Trojan threats.

Recently, research on how to combat the threats of hardware Trojan on third party IP has emerged. Pertinent efforts attempt to exploit a well-developed body of work in the software domain, known as PCC. Originally developed by Necula et al. [21] in 1996, PCC provided a new way of determining whether code from a potentially untrusted source is safe to execute. This was accomplished by establishing a formal, automatically verifiable proof that the questionable code obeys a set of formalized properties. Such a proof may demonstrate, for example, that a given set of machine instructions follows the predefined type-safety rules of a particular programming language. The proof is then combined with the code, which allows the recipient to automatically check the code against the proof. Only if the check process finishes with no errors can the recipient know that the code is safe to run.

The idea of expanding this methodology to the hardware trust domain first appeared in [6]. The authors made a case for the necessity of PCH based on the increasing prominence of FPGAs and reconfigurable devices; if hardware itself becomes just as instantaneously reprogrammable as software, then the capability to establish the trustworthiness of unknown circuitry is inherently desirable. The authors further elucidate the novelty of their approach by contrasting it with other, more common security practices such as formal verification or model-checking. They argued that PCH, in contrast to these, is unique in that it requires very little of end users, who need only perform a straightforward validation check of the proof. The burden of demonstrating security instead falls on the producers, who must construct the proof when they provide the original IP core. This new approach was also differentiated from simple checksum-hashing of FPGA bitstreams because the latter does not take into account the actual functionality of the code being transmitted.

As a first step toward provable hardware security, the authors proposed a technique for presenting proofs of combinational equivalence for digital logic functions implemented in FPGA bitstreams. This approach required an agreed-upon specification function $S(x)$ for each logic function and an implementation $I(x)$ extracted from the FPGA netlist. From these two inputs, a proof would automatically be generated to show that the implementation $I(x)$ was combinationally equivalent to the specification $S(x)$. The consumer could then check this proof against $I(x)$ and $S(x)$ to quickly see that the implemented function agrees with its specification.

The specific technologies employed to achieve this result include standard FPGA CAD tools, a satisfiability (SAT) solver, and a SAT trace checker. From the netlist output of the CAD synthesis tools, each logic function must be proven against its specification. For each such function, then, the authors propose to create a structure called a "miter" which is formed by taking the XOR of $S(x)$ and $I(x)$ as $M(S(x), I(x))$. As the output of the miter can only be *true* if there exists some boolean vector x for which $S(x) \neq I(x)$, a proof that $M(S(x), I(x))$ is unsatisfiable therefore demonstrates that $S(x)$ and $I(x)$ are equivalent.

When the SAT solver finds that $M(S(x), I(x))$ is indeed unsatisfiable, a trace is output to show how this unsatisfiability result was achieved. It is precisely this trace that constitutes the correctness proof under the PCH system. When traces for all relevant functions have been extracted, they are sent along with the bitstream to the consumer. The consumer may then regenerate a miter for each logic function from its netlist implementation and check this against the corresponding proof trace. If all functions check against their traces, then the hardware is accepted as safe.

This work represented a first step toward proof-based security. It is clear that many types of Trojans could be prevented under such a system because modifications to the combinational behavior of an FPGA bitstream's logic functions would be immediately detectable. Nevertheless, the expressiveness of this approach is limited by the need to specify exact Boolean functionality. In software PCC, security policies have generally specified a broader definition of "safe" behavior without necessarily stipulating precisely what a program must compute.

In response to this limitation, other researchers have sought ways of expanding PCH to encompass a more abstract notion of security-related properties. The authors in [7] present Proof-Carrying Hardware Intellectual Property (PCHIP) to guarantee proofs about a circuit's HDL representation, rather than the FPGA bitstream. PCHIP bears a superficial resemblance to more traditional formal verification methods in that it uses a domain-specific temporal logic to codify properties, but its ultimate goal is the transmission and efficient validation of proofs of these properties, which standard formal verification cannot accomplish.

PCHIP introduces a new protocol, shown in Fig. 16.11, for the design and acquisition of hardware intellectual property IP cores. Under this system, a hardware IP consumer commissions a vendor to construct a module according to both a standard functional specification and a set of formal security properties stated in a temporal logic. These properties are markedly different from the specification in that they do not necessarily describe the functional behavior of the module; rather, they delimit the acceptable boundaries of this behavior so that the consumer can rest assured no undesired functionality exists. It is then the IP vendor's task to construct a formal proof that his module complies with these properties. Just as in PCC and PCH, the resulting proof is then transmitted back to the consumer along with the IP cores.

Unlike PCH, however, the properties allowed in PCHIP are much more abstract. In [7], the authors described a formal semantics for a carefully codified syntax of an HDL using the Coq proof assistant [22]. In the context of this semantic model they were able to then craft a temporal logic describing the behavior of signals in

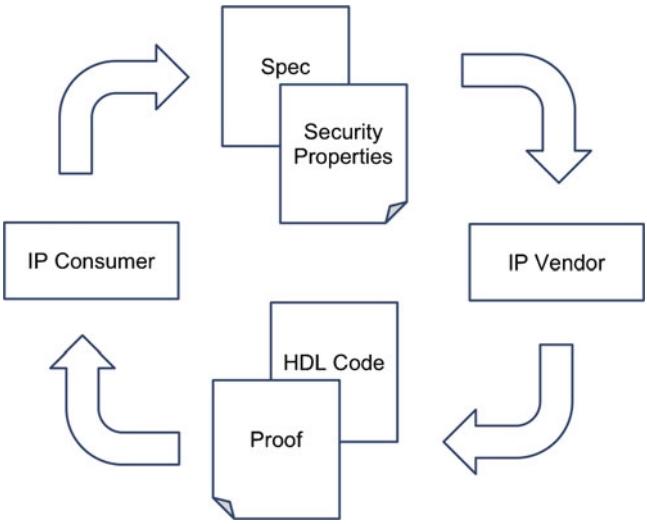

Fig. 16.11 IP core design and aciqusition protocol [7]

synchronous circuits. Security-related properties could be specified in this logic as trees of complex predicates and quantifiers, as opposed to simple Boolean function specifications allowed in PCH. PCHIP provides a set of rules to be applied to HDL code in order to generate a set of propositions in Coq that represent that code's behavior in the already established semantic model. These propositions are then referred to in the proofs that must be constructed for each security-related property.

Figure 16.12 shows how proof-checking proceeds when the consumer receives a circuit's IP code and its corresponding Coq proofs. The code is first passed through a "verification generator" to regenerate all the propositions describing the circuit's behavior, since it is not known whether those included in the received proof actually match the vendor's untrusted code. However, because they are generated according to the same set of rules on both the vendor's side and the client's, it is guaranteed that the resulting propositions will also be the same. Were this not so, then proofs would not be able to refer to them consistently. The regenerated semantic description is then recombined with proofs and security properties, and is checked by the Coq interpreter. If the proofs are found to be valid, then the module is accepted as trustworthy.

The authors of PCHIP argue that the security-related properties expressible in their system can effectively prevent certain types of Trojans by prohibiting the kinds of malicious behaviors Trojans might engage in. Nevertheless, their work leaves a number of important questions that future research might address, especially with regard to the generation of security-related properties. These questions also ask how many of these properties need be standardized, whether there is significant overlap in the kinds of guarantees that consumers of different modules would like to have proven, to what extent proof construction and management can be automated, etc.

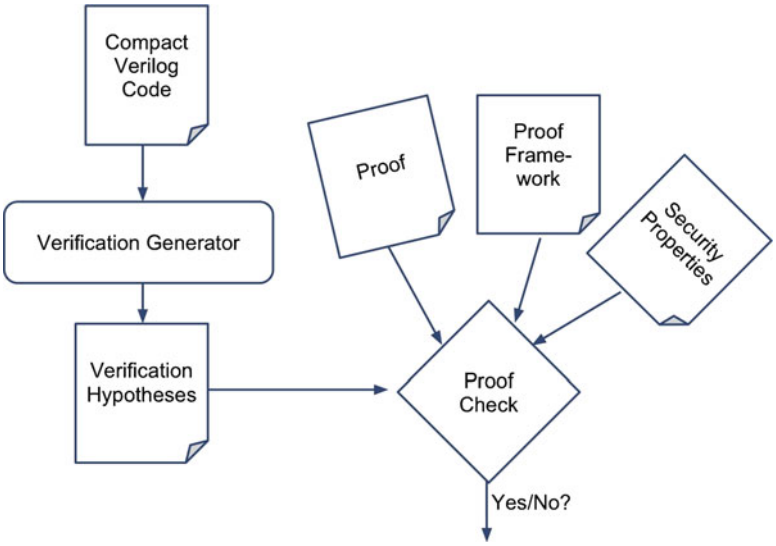

Fig. 16.12 Automated design verification [7]

16.6 Summary

This chapter introduced Trojan prevention methodologies which have been proposed in recent years in the field of *design for hardware trust*. While these methods have proven effective either in simplifying detection of inserted hardware Trojans or in preventing the insertion of Trojans themselves, they do have limitations which one should also be aware of and which point the directions of interest for future research in this area. For side-channel fingerprint-based Trojan prevention methods, area overhead is a key concern because of the complex control logic needed for propagating internal measurements to primary outputs. Other methods, including design obfuscation, dummy flip-flops, and inverted voltage, can only be used if the fundamental assumption that attackers will only choose rare events to trigger the Trojan is valid. Furthermore, a concern with all these methods is the security of the hardening scheme itself, as it is likely that attackers will first compromise the hardening scheme before tampering with the original circuit. In this direction, DFTT is the first method that provides an excellent mechanism for protecting the hardening scheme.

Formal methods such as the PCH and PCHIP paradigms constitute an excellent starting point for future research in the field of pre-silicon Trojan prevention in third party IP. Both methods move the burden of demonstrating IP core security onto the producer in order to accelerate the IP security validation process during its implementation. In the foreseeable future, it is likely that such protocols will become an industry standard in the area of trusted IP acquisition.

Finally, in the domain of hardware trust, contemporary research has almost exclusively focused on digital circuits. However, the extensive use of analog electronics in sensors and actuators, as well as Radio-Frequency (RF) electronics in telecommunications, will certainly attract the interest of both potential attackers and the hardware security and trust community. The first signs of activity in this area have appeared in [23], where the problem of Trojans in wireless cryptographic ICs was studied, so it is very likely that Trojan prevention methods for analog/RF circuits will also be necessitated in the near future.

References

1. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Design Test Comput 27: 10–25
2. Chakraborty RS, Bhunia S (2009) Security against hardware Trojan through a novel application of design obfuscation. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design pp 113–116
3. Salmani H, Tehranipoor M, Plusquellic J (2009) New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 66–73
4. Banga M, Hsiao M (2009) VITAMIN: voltage inversion technique to ascertain malicious insertion in ICs. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 104–107
5. Jin Y, Kupp N, Makris Y (2010) DFTT: design for Trojan test. In: Proceedings of the IEEE International Conference on Electronics Circuits and Systems, pp 1175–1178
6. Drzevitzky S, Kastens U, Platzner M (2009) Proof-carrying hardware: towards runtime verification of reconfigurable modules. In: Proceedings of the Reconfigurable Computing and FPGAs, pp 189–194
7. Love E, Jin Y, Makris Y (2011) Enhancing security via provably trustworthy hardware intellectual property. In: Proceedings of the IEEE Symposium on Hardware-Oriented Security and Trust, pp 12–17
8. Li J, Lach J (2008) At-speed delay characterization for IC authentication and Trojan horse detection. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 8–14
9. Rai D, Lach J (2009) Performance of delay-based Trojan detection techniques under parameter variations. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 58–65
10. Jin Y, Makris Y (2008) Hardware Trojan detection using path delay fingerprint. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 51–57
11. Reece T (2009) Implementation of a ring oscillator to detect Trojans (Vanderbilt University). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Vanderbilt.pdf. Accessed 17 July 2011
12. Rajendran J, Kanuparthi AK, Somasekharan R, Dhandapani A, Xu X (2009) Securing FPGA design using PUF-chain and exploitation of other Trojan detection circuits (Polytechnic Institute of NYU). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Polytechnic_JV.pdf. Accessed 17 July 2011
13. Embedded Systems Challenge (CSAW – Cyber Security Awareness Week) (2009) Polytechnic Institute of New York University.http://www.poly.edu/csaw-embedded
14. Guo R, Kannan S, Liu W, Wang X (2009) CSAW 2009 team report (Polytechnic Institute of NYU). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Polytechnic_Rex.pdf. Accessed 17 July 2011

15. Narasimhan S, Du D, Wang X, Chakraborty RS (2009) CSAW 2009 team report (Case Western Reserve University) CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/CaseWestern.df. Accessed 17 July 2011
16. Yin C-ED, Gu J, Qu G (2009) Hardware Trojan attack and hardening (University of Maryland. College Park). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Maryland.pdf. Accessed 17 July 2011
17. Jin Y, Kupp N (2009) CSAW 2009 team report (Yale University). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Yale.pdf. Accessed 17 July 2011
18. Wolff F, Papachristou C, Bhunia S, Chakraborty RS (2008) Towards Trojan-free trusted ICs: problem analysis and detection scheme. In: Proceedings of the IEEE Design Automation and Test in Europe, pp 1362–1365
19. Abramovici M, Bradley P (2009) Integrated circuit security: new threats and solutions. In: Proceedings of the 5th Annual Workshop Cyber Security and Information Intelligence Research: Cyber Security and Information Challenges and Strategies, article 55
20. Hsiao MS, Tehranipoor M (2010) On trust in third-party hardware IPs. Trust-HUB.org, 2010. http://trust-hub.org/resources/133/download/trust_hub_sept2010-v03.pdf. Accessed 17 July 2011
21. Necula GC (1997) Proof-carrying code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp 106–119
22. INRIA (2010) The coq proof assistant. http://coq.inria.fr/. Accessed 17 July 2011
23. Jin Y, Makris Y (2010) Hardware Trojans in wireless cryptographic ICs. IEEE Design Test Comput 27: 26–36