CHAPTER

HARDWARE TROJANS 5

CONTENTS

Hardware Security. https://doi.org/10.1016/B978-0-12-812477-2.00010-1

5.1 INTRODUCTION

The complexity of modern system on chip (SoCs), amplified by time-to-market pressure, makes it infeasible for a single design house to complete an entire SoC without outside support. Also, the cost to build and maintain a fabrication unit (or foundry) for modern technology nodes makes it infeasible for the majority of SoC design houses to afford their own fabs. Given these factors, and added pressure for lowering time-to-market, the semiconductor industry has shifted to a horizontal business model over the past two decades, where time-to-market and manufacturing costs are lowered through outsourcing and design reuse. In this model, SoC designers often obtain licenses for third party intellectual property (3PIP) cores, design an SoC by integrating the various 3PIPs with their own IPs, and outsource the SoC design to contract foundries and assemblies for fabrication, testing, and packaging.

Due to the emerging trend of outsourcing the design and fabrication services to external facilities and increasing reliance on third-party Intellectual Property (IP) cores, SoCs are becoming increasingly vulnerable to malicious activities and alterations referred to as hardware Trojans (HT). Hardware Trojans are malicious modifications to original circuitry inserted by adversaries to exploit hardware or to use hardware mechanisms to create backdoors in the design. These backdoors can leak sensitive (or private) information as well as enable launching other possible attacks, for example, denial of service and reduction in reliability. They have raised serious concerns regarding possible threats to military systems, financial infrastructures, transportation security, and household appliances. Hardware Trojans have reportedly been used as ‘kill switches’ and backdoors in foreign military weapon systems [1]. Current and former U.S. military and intelligence executives agree that the dangers of hardware Trojans hidden in a chip are among the most severe threats the nation faces in the event of a war [2].

Detection of hardware Trojans is extremely difficult, for several reasons. First, given the large number of soft, firm, and hard IP cores used in SoCs, and the high complexity of today’s IP blocks, detecting a small malicious alteration is extremely difficult. Second, nanometer SoC feature sizes make detection by physical inspection and destructive reverse engineering very difficult, time consuming, and costly. Moreover, destructive reverse engineering does not guarantee that the remaining SoCs will be Trojan-free, especially when Trojans are selectively inserted into a portion of the chip population. Third, Trojan circuits, by design, are typically activated under very specific conditions (for example, connected to low-transition probability nets or sensing a specific design signal, such as power or temperature), which makes them unlikely to be activated and detected using random or functional stimuli. Fourth, tests used to detect manufacturing faults, such as stuck-at and delay faults cannot guarantee detection of Trojans. Such tests operate on the netlist of a Trojan-free circuit and, therefore, cannot activate and detect Trojans. Even when 100% fault coverage for all types of manufacturing faults is possible, there are no guarantees as far as Trojans are concerned. Finally, as physical feature sizes decrease because of improvements in lithography, process and environmental variations have an increasingly greater impact on the integrity of the circuit parametric behavior. Thus, detection of Trojans using simple analysis of these parametric signals would be ineffective. All these factors make the detection of Trojan in an SoC a very challenging task.

The following section discusses the modern SoC design flow and how the rogue entities in this flow can insert hardware Trojans. Thereafter, a comprehensive Trojan taxonomy is presented. This taxonomy systematically categorizes hardware Trojans, which facilitate the development of Trojan mitigation, detection, and protection techniques. Further, a detailed description of the state-of-the-art countermeasures for Trojan threat both in terms of detection and prevention is presented.

5.2 SoC DESIGN FLOW

A typical SoC design flow is shown in Fig. 5.1. Design specification by the SoC integrator is generally the first step. For example, the SoC integrator first identifies what functionalities need to be incorporated in the SoC, and what will be the targeted performance. The integrator then identifies a list of functional blocks to implement the SoC. These functional blocks have intellectual property values and are commonly referred to as IPs. These IP cores are either developed in-house or purchased from 3PIP developers. These 3PIP cores can be procured either as soft IP (register-transfer level (RTL)), firm IP (gate level), and hard IP (layout) [3].

After developing/procuring all the necessary soft IPs, the SoC design house integrates them to generate RTL specification of the whole SoC. The RTL design goes through extensive functional testing to verify the functional correctness of the SoC and also to find any design bugs. SoC integrator synthesizes the RTL description into a gate-level netlist based on a target technology library. Synthesis is a process by which an RTL code is transformed into a hardware implementation consisting of logic gates with the help of a computer aided design (CAD) tool, for example, Design Compiler of Synop-

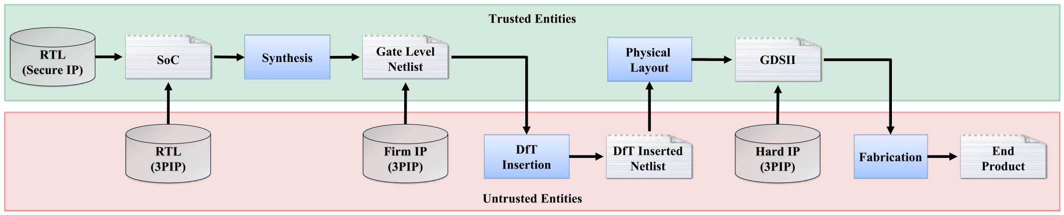

FIGURE 5.1

System on chip (SoC) design flow.

sys. The CAD tools also optimize the design with the target to minimize area, timing or power. The gate-level netlist then goes through formal equivalence checking to verify that the netlist is equivalent to the RTL representation. The SoC designer may also integrate a firm IP core from a vendor into the SoC netlist at this stage. The SoC integrator then integrates design-for-test (DFT) structures to improve the design's testability. However, in many cases, the DFT insertion is outsourced to third party vendors who specialize in designing test and debug structures, such as built-in self-test (BIST) and compression structures. In the next step, the gate-level netlist is translated into a physical layout design. It is also possible to import hard IP cores from vendors and integrate them at this stage. After performing static timing analysis (STA) and power closure, the SoC integrator generates the final layout in GDSII format and sends it out to a foundry for fabrication. The generated GDSII file contains layer-by-layer information needed to fabricate the SoC on a silicon wafer. The foundry fabricates the SoC and performs structural tests on the die and wafer to find manufacturing defects. After fabrication, the foundry sends tested wafers to the assembly line to cut the wafers into die, and package the die to produce chips [4].

5.2.1 HARDWARE TROJAN INSERTION: POTENTIAL ADVERSARIES

In the hardware supply chain, there are potential adversaries who have the capability and access to insert Hardware Trojans into the IC design. A brief discussion about these adversaries follows.

3PIP Vendor: Due to short time-to-market constraints, design houses are increasingly becoming dependent on third party vendors to procure IPs. These IPs are designed by hundreds of IP vendors distributed across the world. Such IPs cannot be assumed trusted as hardware Trojans can be maliciously inserted into them. The 3PIP vendors have full control over their IP and can insert stealthy Trojans, which would be extremely difficult, if not impossible, to detect using traditional test and verification techniques.

Detection of Trojans in 3PIPs is challenging as there is no golden reference against which to compare a given IP core during verification. Also, the system integrator only has the black-box knowledge of the IP, that is, the system integrator only knows high-level functionality of the design. However, he/she does not know the low-level implementation design of the Trojan. A large industrial-scale IP core can include thousands of lines of code. It is an extremely challenging task to identify the few lines of RTL code that represents a Trojan in the IP core [5].

DFT Vendor: In the current hardware design flow, the DFT insertion process is generally outsourced to third party vendors who specialize in designing test and debug structures. These vendors have access to the whole design and have the capability to incorporate stealthy malicious circuitry in the design.

Also, DFT vendors incorporate additional test and debug hardware into the original design to improve test coverage of the design. These additional hardware can also include hardware Trojans.
Fabrication: Due to the complexity and cost of semiconductor chip fabrication, most design houses have become fabless, that is, they fabricate their products in third party foundries offshore to lower cost. In this process, the foundry has access to the whole SoC design and can maliciously tamper with the design. Therefore, a rogue foundry has a unique opportunity to insert hardware Trojan in the SoC. It can exploit the empty spaces in the SoC to introduce malicious functionality in addition to the actual functionality. Trojan inserted by rogue foundry is extremely difficult, if not impossible to detect. The reason is that unlike the pre-silicon design stage, a fabricated SoC offers very limited verification and validation options.

5.3 HARDWARE TROJANS

A hardware Trojan (HT) is defined as a malicious, intentional modification of a circuit design that results in undesired behavior when the circuit is deployed [6]. SoCs that are ‘infected’ by a hardware Trojan may experience changes in their functionality or specification, may leak sensitive information, or may experience degraded or unreliable performance. Hardware Trojan poses a serious threat to any hardware design being deployed in a critical operation.

As the hardware Trojans are inserted at the hardware level, software-level countermeasures may be inadequate to address the threat posed by HT. Also, detection of Trojans in a hardware design is challenging as there is no golden version against which to compare a given design during verification. In theory, an effective way to detect a Trojan is to activate the Trojan and observe its effects, but a Trojan’s type, size, and location are unknown, and its activation is, most likely, a rare event. A Trojan can be, therefore, well hidden during the normal functional operation of the chip and activated only when the triggering condition is applied.

5.3.1 HARDWARE TROJAN STRUCTURE

The basic structure of a Trojan in a 3PIP can include two main parts, trigger and payload [3]. 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 5.2 shows the basic structure of the Trojan at gate level. The trigger inputs ($T_1, T_2, ..., T_k$) 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 most of the time, Net_i. However, when the trigger is active, that is, *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. 5.2.

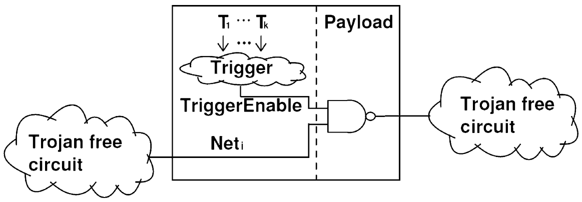

FIGURE 5.2

Trojan structure.

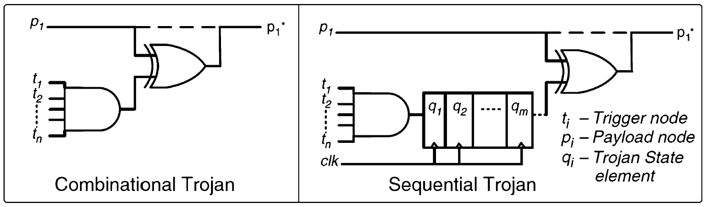

FIGURE 5.3

Example of combinational and sequential Trojan models.

5.3.2 TROJAN MODELING

Many different Trojans have been developed over the years by researchers using different Trojan models to demonstrate the capability and efficacy of their proposed Trojan countermeasure approaches. This section discusses the digitally triggered and a digital payload Trojan model that has been most commonly used. In this model, it is assumed that a Trojan will be activated by rare circuit node conditions and will have its payload as a critical node in terms of functionality, but low observable node in terms of testing, to evade detection during normal functional testing. If the Trojan includes sequential elements, such as rare-event triggered counters, then the Trojan may be even harder to detect.

Figure 5.3 shows generic models for combinational and sequential Trojans [7]. The trigger condition is an n-bit value at internal nodes, which is assumed to be rare enough to evade normal functional testing. The payload is defined as a node that is inverted when the Trojan is activated. To make it more difficult to detect, one might consider a sequential Trojan, which requires the rare event to repeat 2^m times before the Trojan gets activated and inverts the payload node. The sequential state machine is considered in its simplest form to be a counter, and the effect of the output on the payload is considered to be an XOR function to have maximal impact. In more generic models, the counter can be replaced by any Finite State Machine (FSM) and the circuit can be modified as a function of Trojan output and the payload node. In like manner, the Trojan trigger condition is modeled as an AND function of several rare nodes in order to make the combined event highly rare. However, the attacker may choose to reuse existing logic within the circuit to implement the trigger condition, without adding too many extra gates.

It is worth noting—when choosing rare values at internal nodes as inputs for the Trojan—that the combined rare event must be excitable; otherwise, the Trojan will never be triggered in real life.

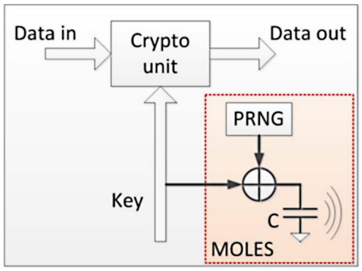

FIGURE 5.4

An example Trojan design with capability of leaking secret information from inside a cryptochip through power side-channels.

5.3.3 HARDWARE TROJAN EXAMPLES

This section provides some examples of hardware Trojans. It describes what will be the potential Trojans in a cryptomodule and in a general purpose processor.

5.3.3.1 Trojans in Cryptographic Engines

A possible Trojan attack in a cryptoengine can try to subvert the security mechanisms. The payload could range from a mechanism that presents dummy keys, predefined by the attacker, instead of the actual cryptographic keys used for sensitive encryption or signature verification operations, to leaking the secret hardware keys via covert side-channels, for example, information leaked through a power trace. Figure 5.4 provides an example of such a Trojan, which attempts to leak a secret key from inside a cryptographic module through power side-channels using a technique called malicious off-chip leakage, enabled by side-channels (MOLES) [8]. Other targets could be a random number generator used for deriving random session keys for a particular operation, or the debug passwords used for unlocking test mode access to security-sensitive signals. Researchers have also proposed leaking such secret information over wireless channels [9] by using low-bandwidth modulation of the transmitted signal.

5.3.3.2 Trojans in General-Purpose Processors

In general-purpose processors, an attacker at the fabrication facility can implement a backdoor, which can be exploited in the field by a software adversary [10–13]. For example, modern processors implement a hardware chain of trust to ensure that malware cannot compromise the hardware assets, such as secret keys and memory range protections. By using different stages of firmware and boot code authentication, one can ensure that the operating system (OS) kernel and lower levels, such as hypervisor, are not corrupted. However, in such systems, the attacker at an untrusted fabrication facility could implement a backdoor, which disables the secure booting mechanism under certain rare conditions or when presented with a unique rare input condition in the hands of an end-user adversary [10]. Likewise, other objectives that could be realized with the help of hardware Trojans would be able to bypass memory range protections using buffer overflow attacks, or gain access to privileged assets by evading the access control protection mechanisms implemented in the hardware.

This section discusses the Trojan threat from application-specific integrated circuit (ASIC) perspective. The next section will discuss the Trojan threat for FPGA design flow and supply chain.

5.4 HARDWARE TROJANS IN FPGA DESIGNS

FPGAs are widely used today in an array of embedded applications, ranging from telecommunications and data centers to missile guidance systems. Unfortunately, the outsourcing of FPGA production and the use of untrusted third party IPs has also given rise to the threat of Trojan insertion in them. FPGA-based Trojans can be in the form of IP blocks (hard, soft or firm), which get loaded onto a generic FPGA fabric and cause malicious activity (such as denial-of-service and leakage) on the system in which the FPGA is deployed. Such FPGA IP-based Trojans are more or less similar to their counterparts in an ASIC design flow, with the exception of layout-based Trojans, which are not applicable to FPGAs. However, Trojans that 'pre-exist' in an FPGA fabric and could potentially be inserted by an untrusted foundry or vendor pose unique threats and challenges of their own. FPGAs contain a large volume of reconfigurable logic in the form of lookup tables, block RAM, and programmable interconnects, which can be used to realize any arbitrary sequential or combinational design. However, there might be a significant amount of reconfigurable logic open to a malicious party (for example, the FPGA foundry or even the FPGA vendor) who can load a hardware Trojan and affect the FPGA-integrated system or compromise the IP loaded onto the FPGA. These FPGA device-specific hardware Trojans and their effects are explained in [14] and summarized below.

5.4.1 ACTIVATION CHARACTERISTIC

Hardware Trojans in FPGAs can have activation characteristics similar to the ones described in Section 5.3.1 and 5.3.2, such as always-on or triggered. However, a unique characteristic of FPGA device-based hardware Trojans is that they can either be IP-dependent or IP-independent.

5.4.1.1 IP-Dependent Trojans

A malicious foundry or FPGA vendor may implement a hardware Trojan that can monitor the logic values of several lookup tables (LUTs) in the FPGA fabric. Once triggered, such Trojans can corrupt other LUT values, load incorrect values into block RAMs (BRAMs), or sabotage configuration cells. Since any arbitrary IP may be loaded onto the FPGA, the malicious foundry or vendor could distribute trigger LUTs throughout the FPGA so that the probability of the Trojan triggering and causing malfunction may increase.

5.4.1.2 IP-Independent Trojans

A malicious foundry or vendor may also implement a Trojan into an FPGA chip that is completely independent of the IP loaded onto it. Such Trojans can occupy a small portion of FPGA resources and malfunction IP-independent but critical FPGA resources, such as digital clock managers (DCM). One potential mode of attack would be a Trojan increasing or decreasing the design clock frequency by manipulating the configuring SRAM cells of the DCM unit, which can cause failure in sequential circuits.

5.4.2 PAYLOAD CHARACTERISTICS

FPGA device-based Trojans can also bring about unique malicious effects, such as causing malfunction of FPGA resources or leakage of the IP loaded onto the FPGA.

5.4.2.1 Malfunction

Hardware Trojans in FPGA devices can either cause logical malfunction by corrupting LUT(s) or SRAM values, thereby affecting the functionality of the implemented IP, or by causing physical damage to the FPGA device. For example, a triggered hardware Trojan could reprogram an I/O port set as an input to become an output I/O port, while suppressing the configuration cells that prevent it from being programmed as such. This would cause a high short-circuit current to flow between the FPGA and the system it is connected to, thereby leading to physical device failure.

5.4.2.2 IP Leakage

FPGAs today offer bitstream encryption capabilities in order to protect the IP loaded onto an FPGA device. However, such encryption only prevents a direct or unauthorized readback by software. A hardware Trojan may circumvent such protection by either leaking the decryption key, or even the entire IP. The Trojan may tap the decryption key as it comes out of non-volatile memory, or the actual decrypted IP, which could then be exfiltrated either via covert side-channels (for example, power traces) or through JTAG, USB or I/O ports.

5.5 HARDWARE TROJANS TAXONOMY

Developing a better understanding of hardware Trojans and creating effective defenses require a framework that groups similar Trojans together to enable a systematic study of their characteristics. Detection, mitigation, and protection techniques can then be developed for each Trojan class along with benchmarks to serve as the basis for comparing countermeasures. In addition, experimental implementations can be created for Trojan classes yet to be observed, thereby fostering proactive defense.

Taxonomy is developed based on hardware Trojans' physical, activation, and functional characteristics. In this regard, hardware Trojans are classified based on five attributes: (1) insertion phase, (2) abstraction level, (3) activation mechanism, (4) payload, and (5) location (shown in Fig. 5.5) [15].

5.5.1 INSERTION PHASE

Hardware Trojans can be inserted throughout SoC design flow. As the abovementioned Trojan's attribute suggests, Trojans can also be classified based on the phases in which they are inserted.

5.5.1.1 Specification Phase

In this phase, chip designers define the system's characteristics: the target environment, expected function, size, power, and delay. While the SoC development is in this phase, functional specifications or

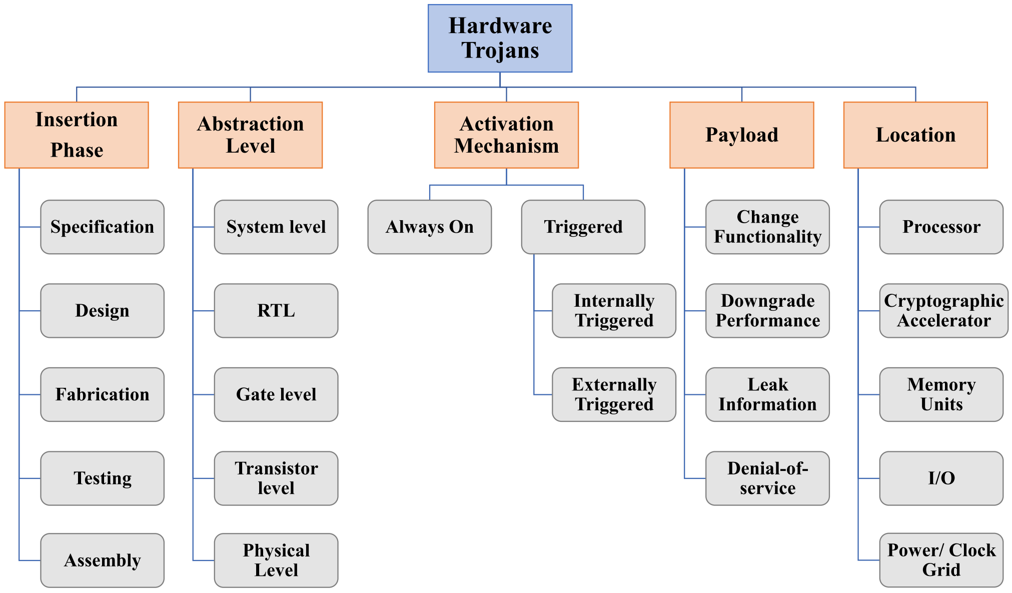

FIGURE 5.5

Taxonomy of hardware Trojans.

other design constraints can be altered. For example, a Trojan at the specification phase might change the hardware's timing requirements.

5.5.1.2 Design Phase

Developers consider functional, logical, timing, and physical constraints as they map the design onto the target technology. At this point, they can use third-party IP blocks and standard cells. Trojans might be in any of the components that aid the design. For example, a standard cell library can be tampered with Trojans.

5.5.1.3 Fabrication Phase

During this phase, developers create a mask set and use wafers to produce the masks. Subtle mask changes can have serious effects. In an extreme case, an adversary can substitute a different mask set. Alternatively, chemical compositions might be altered during fabrication to increase the electromigration in critical circuitry, such as power supplies and clock grids, which would accelerate failures.

5.5.1.4 Testing Phase

The IC testing phase is important for hardware trust, not only because it is a likely phase for Trojan insertion, but also because it provides an opportunity for Trojan detection. Testing is only useful for detection when done in a trustworthy manner. For example, an adversary who inserted a Trojan in the fabrication phase would want to have control over the test vectors to ensure that the Trojan is not detected during test. Trustworthy testing ensures that the test vectors will be kept secret and faithfully applied, and that the specified actions accept/reject, and binning will be faithfully followed. An attacker can take advantage of not-detected (ND) faults by automatic test pattern generation (ATGP) tools. By doing so, a Trojan that uses these ND faults will never be activated.

5.5.1.5 Assembly Phase

Developers assemble the tested chip and other hardware components on a printed circuit board (PCB). Every interface in a system, where two or more components interact is a potential Trojan insertion site. Even if all the ICs in a system are trustworthy, malicious assembly can introduce security flaws in the system. For example, an unshielded wire connected to a node on the PCB can introduce unintended electromagnetic coupling between the signal on the board and its electromagnetic surroundings. An adversary can exploit this for information leakage and fault injection.

5.5.2 ABSTRACTION LEVEL

Trojan circuits can be inserted at various hardware abstraction levels. Their functionality and the structure are dependent on the abstraction level at which they are inserted.

5.5.2.1 System Level

At the system level, the different hardware modules, interconnections, and communication protocols used are defined. At this level, the Trojans may be triggered by the modules in the target hardware. For example, the ASCII values of the inputs from the keyboard can be interchanged.

5.5.2.2 Register-Transfer Level

At the RTL, chip designers describe each functional module in terms of registers, signals, and Boolean functions. A Trojan can be easily designed and inserted at the RTL because the attacker has full control over the design's functionality. For example, a Trojan implemented at this level might halve the rounds of a cryptographic algorithm by making a round counter to advance in two steps instead of one.

5.5.2.3 Gate Level

At this level, an SoC is represented as an interconnection of logic gates. An attacker can carefully control all aspects of the inserted Trojan, including its size and location. For example, a Trojan might be a simple comparator consisting of basic gates (AND, OR, XOR gates) that monitor the chip's internal signals.

5.5.2.4 Transistor Level

Chip designers use transistors to build logic gates. This level gives the Trojan designer control over circuit characteristics, such as power and timing. The attacker can insert or remove individual transistors, altering the circuit functionality, or modify transistor sizes to alter circuit parameters. For example, a transistor-level Trojan might be a transistor with low gate width that can cause more delay in the critical path.

5.5.2.5 Physical Level

This level describes all circuit components and their dimensions and locations, and is the design's physical level, where a Trojan can be inserted. An attacker can insert Trojans by modifying the size of the wires and distances between circuit elements and reassigning metal layers. For example, changing the width of the clock wires, timing critical nets or metal wires in the chip can cause clock skew.

5.5.3 ACTIVATION MECHANISM

Some Trojans are designed to be always on, whereas others remain dormant until triggered. A triggered Trojan needs an internal or external event to be activated. Once the trigger activates a Trojan, it can remain active forever or return to a dormant state after a specified time.

5.5.3.1 Internally Triggered

An internally triggered Trojan is activated by an event within the target device. The event might be either time-based or physical-condition-based. A counter in the design can trigger a Trojan at a predetermined time, resulting in a silicon timebomb. Similarly, a Trojan can trigger when the chip temperature exceeds a certain threshold.

5.5.3.2 Externally Triggered

An externally-triggered Trojan requires external input to the target module to activate. The external trigger can be user input or component output. User-input triggers include push buttons, switches, keyboards, or keywords and phrases in the input data stream in a system. Component-output triggers might be from any of the components that interact with the target device. For example, a Trojan in a cryptomodule can derive its trigger condition from the applied plaintext input and triggers when it observes a specific plaintext, or a combination of plaintext and operating conditions.

One can also classify the triggered Trojans into two categories: (1) Analog triggered and (2) Digital triggered. Analog-triggered Trojans are triggered analog signals, such as temperature and voltage. Digitally triggered Trojans are triggered by logic-conditions, such as state of the flip-flops, state of a logic net, counter, clock signal, data, instruction, and/or interrupts.

5.5.4 PAYLOAD

Trojans can also be characterized by their payload, that is, the malicious effects caused by them when they become activated. The severity of these effects on target hardware or systems can range from subtle disturbances to catastrophic system failures.

5.5.4.1 Change Functionality

Trojan can change the functionality of the target device, and cause subtle errors that might be difficult to detect during manufacturing test. For example, a Trojan might cause an error detection module to accept inputs that should be rejected.

5.5.4.2 Downgrade Performance

A Trojan can downgrade performance by intentionally changing device parameters. These include functional, interface, or parametric characteristics, such as power and delay. For example, a Trojan might insert more buffers in the chip's interconnections and, hence, consume more power, which in turn could drain the battery quickly.

5.5.4.3 Leak Information

A Trojan can also leak information through both covert and overt channels. Sensitive data can be leaked via radio frequency, optical or thermal power, timing side-channels, and interfaces, such as RS-232 and JTAG (Joint Test Action Group). For example, a Trojan might leak a cryptographic algorithm's secret key through unused RS-232 ports.

5.5.4.4 Denial-of-Service

A denial-of-service (DoS) Trojan can cause the target module to exhaust scarce resources, such as bandwidth, computation, and battery power. It could also physically destroy, disable, or alter the device's configuration, for example, causing the processor to ignore the interrupt from a specific peripheral. DoS can be either temporary or permanent.

5.5.5 LOCATION

A hardware Trojan can be inserted in a single component or spread across multiple component, for example, processor, memory, input/output, power supply, or clock grid. Trojans distributed across multiple components can act independently of one another or together as a group to accomplish their attack objectives.

5.5.5.1 Random Logic

A Trojan can be inserted into the random logic portion of an SoC. Detection of such Trojans are extremely challenging since understanding functionality of random logic is difficult, hence making it limited to generate effective test stimuli. Size of such logic in an SoC can be quite large.

5.5.5.2 Processing Unit

Any Trojan embedded into the logic units that are part of the processor can be grouped under this category. A Trojan in the processor might, for example, change the instructions' execution order.

5.5.5.3 Cryptographic Accelerator

Cryptomodules are likely target for Trojan insertion as these modules work with assets, such as private keys and sensitive plaintext. A Trojan in a cryptomodule can leak the secret key (attack on confidentiality) or replace the key (attack on integrity), and compromise the security of the overall system.

5.5.5.4 Memory Units

Trojans in the memory blocks and their interface units fall in this category. These Trojans might alter the value stored in the memory and also blockread or blockwrite access to certain memory locations, for example, change the contents of a programmable read-only memory in an SoC.

5.5.5.5 Input/Output Port

Trojans can reside in a chip's peripherals or within the PCB. These peripherals interface with the external components and can give the Trojan control over data communication between the processor and the system's external components. For example, a Trojan might alter the data coming through a JTAG port.

5.5.5.6 Power Supply

Modern SoCs include many voltage islands, a large number of locally distributed voltage regulators, and dynamic voltage/frequency systems, where the chip frequency is adjusted in the field by changing VDD as the chip ages in the field. Trojans could be inserted by an adversary to alter the voltage and current supplied to the chip, causing failures.

5.5.5.7 Clock Grid

Trojans in the clock grid can change the clock's frequency, insert glitches in the clock supplied to the chip, and launch fault attacks. These Trojans can also freeze the clock signal supplied to the rest of the chip's functional modules. For example, a Trojan might increase the clock signal skew supplied to specific parts of a chip, causing hold-time violation on short paths.

5.6 TRUST BENCHMARKS

A "trust benchmark" is a benchmark circuit (generic circuits at the RTL, gate or layout level), which has Trojan(s) deliberately added to it at hard-to-detect, impactful, and/or opportunistic locations (for example, rare nodes and layout white-space), for the purpose of comparing impacts of Trojans and the effectiveness of different Trojan detection techniques [16]. The current trust benchmarks are available at http://www.trust-hub.org/benchmarks.php.

Each benchmark comes ready with documentation, that provides important features of the trust benchmark, such as trigger probability (for gate/layout level Trojans), exact effect of the Trojan, input combination required to trigger Trojan (for RTL/gate-level), Trojan-induced delay or capacitance, and size of Trojan/overall circuit. Additionally, for some benchmarks, a "golden model" is provided, that is, a version of the same circuit without Trojans, which is essential for analyzing trust benchmarks against different attack models. Finally, for most of the trust benchmarks, two test benches are included, one of which can be used with the golden model (for debugging and test purposes), and the other can be used to trigger the Trojan. For RTL trust benchmarks, the test bench is in the form of Verilog/VHDL format that have the Trojan trigger specified. For netlist/gate-level benchmarks, exact test patterns to trigger the Trojan are provided. Finally, the documentation for each trust benchmark contains the exact form and location of the inserted Trojan. For example, for RTL Trojans, the part of the RTL code that implements the Trojan has been documented. For gate-level circuits, a snippet of the Trojan netlist has also been provided. The exact location and implementation of the Trojan have also been provided to make it easier for researchers to present results in terms of detection accuracy. However, it should be noted that such information must only be used a posteriori, as taking into account the Trojan implementation and location beforehand might unfairly bias detection techniques. Lastly, it is relevant to note that, generally, Trojan benchmarking is an ongoing effort, and more trust benchmarks are being developed to cover the Trojan taxonomy and improve on existing ones.

The following are some of the representative benchmarks from approximately a hundred benchmarks developed so far:

5.6.1 BENCHMARK NAMING CONVENTION

Trust benchmark follows the following naming convention to assign a unique name to each Trojan benchmark in a trust benchmark circuit: **DesignName-Tn#$**, where,

- **DesignName:** The name of the main design without a Trojan.
- **Tn (Trojan number):** It is of a maximum two digits. The same Trojan number in different designs does not represent the same Trojan.
- **# (Placement number):** The second to last digit indicates the different placement of the same Trojan in a circuit and ranges from 0 to 9.
- **$ (Version number):** The last digit in a benchmark name indicates the version of the Trojan and ranges from 0 to 9. This was added as a feature in case a new version of the same Trojan with the same placement would have been developed. The version number would differentiate the older version from the new one.

For example, MC8051-T1000 indicates that Trojan number 10 (T10) was inserted in the microcontroller 8051 (MC 8051) at the location number 0, and its version is 0. As another example, dma-T1020 means that Trojan number 10 (T10) was inserted in the DMA circuit at the location number 2 and its version is 0. As aforementioned, Trojan T10 in DMA is not necessarily the same as Trojan T10 in MC8051.

5.6.2 SAMPLE TRUST BENCHMARKS

Following are some benchmarks with a brief description of their enclosed Trojan:

- **Insertion Phase – Fabrication:** Trojans can also be realized by adding/removing gates or changing the circuit layout during GDSII development, and the mask during fabrication.
  Sample Benchmark: EthernetMAC10GE-T710 contains a Trojan triggered by a combinational comparator circuit, which seeks a specific 16-bit vector. The probability of Trojan activation in this case is 6.4271e-23. When the Trojan is triggered, its payload gains control over an internal signal in the circuit.
- **Abstraction Level – Layout:** Trojans can be realized by varying circuit mask, adding/removing gates, or changing gate and interconnect geometry to impact circuit reliability.
  Sample Benchmark: EthernetMAC10GE-T100 contains a Trojan on a critical path. A particular net is widened to increase coupling capacitance, thereby enabling crosstalk.
- **Activation Mechanism – Triggered Externally:** Trojans become activated under certain external conditions, such as by an externally enabled input.
  Sample Benchmark: RS232-T1700 contains a Trojan triggered by a combinational comparator. The trigger input probability is $1.59e^{-7}$ and it is externally controlled. Whenever the Trojan is triggered, its payload gains control over a particular output port.
- **Effect – Change Functionality:** After activation, a Trojan will change the functionality of a circuit.
  Sample Benchmark: RS232-T1200 contains a Trojan triggered by a sequential comparator with probability $8.47e^{-11}$. Whenever the Trojan is triggered, its payload gains control over a particular output port.
- **Location – Power Supply:** A Trojan can be placed in the chip power network.

Sample Benchmark: EthernetMAC10GE-T400 is modified with narrow power lines in one part of the circuit layout.

- **Physical Characteristic – Parametric:** A Trojan can be realized by changing circuit parameters, such as wire thickness.
  Sample Benchmark: EthernetMAC10GE-T100 contains a Trojan on the critical path. This Trojan widens a particular internal wire to cause timing violation.

Table 5.1 Trust benchmark characteristics

Category	Trojan Type	No.	Main Circuits
Insertion Phase	Specification	0	–
	Design	80	AES, BasicRSA, MC8051, PIC16F84, RS232, s15850, s35932, s38417, s38584
	Testing	0	–
	Assembly	0	–
Abstraction Level	RTL	51	AES-T100, b19, BasicRSA, MC8051, PIC16F84, RS232
	Gate	25	b19, EthernetMAC10GE, RS232, s15850, s35932, s38417, s38584, VGA LCD
	Layout	12	EthernetMAC10GE, MultPyramid, RS232
Activation Mechanism	Always On	11	AES-T100, MultPyramid, EthernetMAC10GE
	Triggered	79	AES, b19, BasicRSA, MultPyramid, PIC16F84l, RS232, s15850
Payload	Change Functionality	35	b19, Ethernet MAC10GE, MC8051, RS232, s15850, s35932, s38417, s38584
	Degrade Performance	3	EthernetMAC10GE, MultPyramid, s35932
	Leak Information	24	AES, BasicRSA, PIC16F84, s35932, s38584
	DoS	34	AES, BasicRSA, EthernetMAC10GE, MC8051, MultPyramid, PIC16F84, RS232
Location	Processor	26	b19, b19, BasicRSA, MC8051, MultPyramid, PIC16F84, s15850, s35932, s38417, s38584
	Crypto Modules	25	AES-T100 to T2100, BasicRSA-T100
	Memory	0	–
	I/O	4	MC8051, wb_conmax
	Power Supply	2	MC8051-T300, wb_conmax
	Clock Grid	2	EthernetMAC10GE

Table 5.1 presents a complete list of trust benchmarks that have been developed so far. They are categorized based on the Trojan taxonomy, including the number of trust benchmarks available for each type and the names of the main circuits/benchmarks into which the Trojans have been inserted. For instance, Table 5.1 shows that 25 Trojans are inserted at the gate-level, 51 at the RTL, and 12 at the

layout level, under the row “Abstraction Level”. As another example, the “Payload” row shows that 35 Trojans change circuit functionality, 3 degrade circuit performance, 24 leak information to the outside of a chip, and 34 perform a denial-of-service attack when activated. Note that some benchmarks fall under more than one category. Currently, there are a total of 91 trust benchmarks on the Trust-Hub website.

5.7 COUNTERMEASURES AGAINST HARDWARE TROJANS

Several Trojan detection approaches have been developed over the years. Without loss of generality, these approaches are classified into two broad categories, Trojan Detection and Trojan Prevention, each of which can further be classified into several subcategories, as shown in Fig. 5.6.

5.7.1 TROJAN DETECTION

Trojan detection is the most straightforward and commonly used approach to deal with hardware Trojans. It aims to verify the existing designs and fabricated SoCs without any supplementary circuitry. They are performed either at the design stage (that is, pre-silicon) to validate SoC designs or after the manufacturing stage (that is, post-silicon) to verify fabricated SoCs.

5.7.1.1 Post-silicon Trojan Detection

These techniques are employed after the chip is fabricated; they can be classified into destructive and nondestructive methods, as illustrated in Fig. 5.6.

Destructive methods: These techniques typically use destructive reverse-engineering to depackage an IC, and obtain images of each layer in order to reconstruct the design-for-trust validation of the end product. Destructive reverse engineering has the potential of giving very high assurance that any malicious modification in the IC can be detected, but it comes with high cost and could take several weeks and months for an IC of reasonable complexity. Additionally, at the end of this invasive process, the IC cannot be used, and one can only obtain the information for a single IC sample. Note that reverse engineering of modern complex SoCs is a tedious and error-prone process. Hence, in order to obtain the entire chip structure reverse engineered, one may use tens of ICs as deprocessing and depackaging could cause unintentional errors in the reverse engineering process. Therefore, in general, destructive approaches do not seem viable for Trojan detection. However, destructive reverse engineering on a limited number of samples can be attractive to obtain the characteristics of a golden batch of SoCs. Bao et al. [17] presented a machine learning method which adapts one-class support vector machine (SVM) to identify Trojan-free ICs for the golden model.

Functional tests: These techniques attempt to activate Trojans by applying test vectors and comparing the responses with the correct results. To be effective, such techniques require availability of a golden response. 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 in detecting hardware Trojans [12]. 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. Banga and Hsiao [18] and Chakraborty et al. [19] developed test pattern generation methods to trigger such rarely activated nets, and improved the possibility of observing

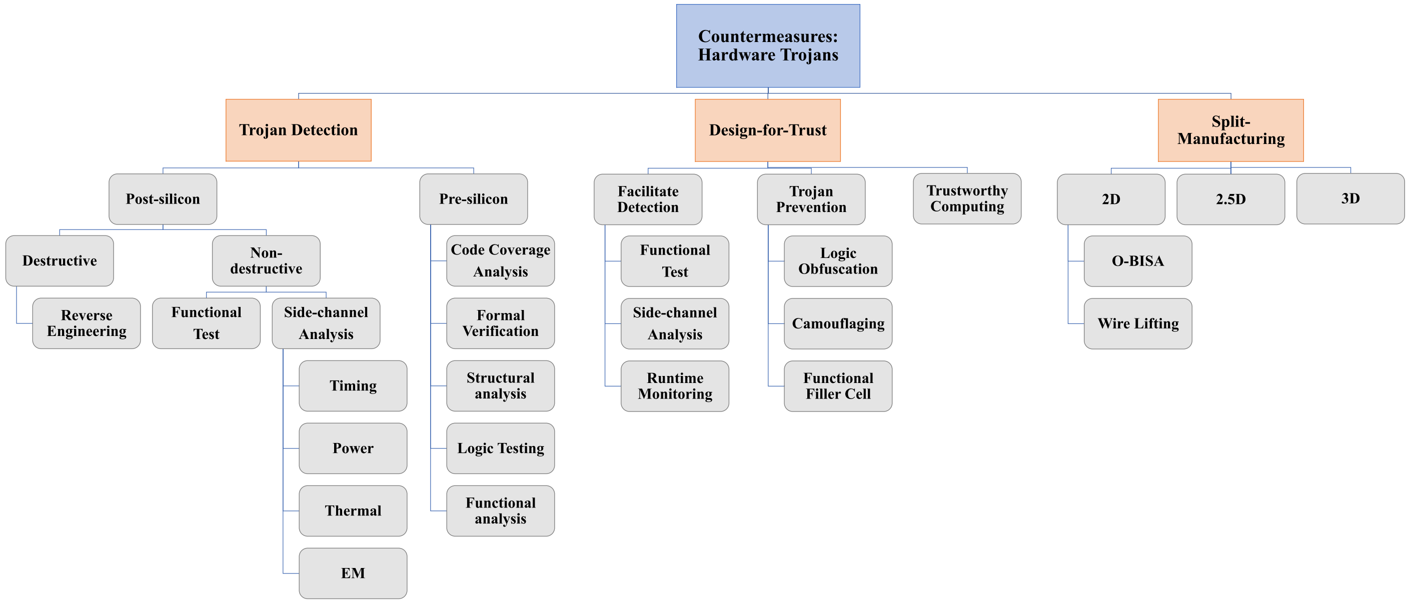

FIGURE 5.6

Taxonomy of hardware Trojan countermeasures.

Trojan's effects from primary outputs. However, due to the numerous logical states in a circuit, it is impractical to enumerate all states of a real design. Additionally, instead of changing the functionality of the original circuit [20], a Trojan that transmits information via nonfunctional means, for example, with an antenna or by modifying the specification, leads to functional tests failure to detect these kinds of Trojans.

Side-channel signal analysis: These approaches detect hardware Trojans by measuring circuit parameters, such as delay [21,22], power (transient [23] and leakage power [24]), temperature [25], and radiation [26,27]. They take advantage of side effects (that is, extra path delay, power, heat, or electromagnetic radiation) caused by additional circuits and/or activity from Trojan trigger/payload activation. However, the majority of the detection techniques assume that "golden ICs" (Trojan-free ICs) are available for comparison in order to identify Trojan-infected ICs. For example, Agrawal et al. [23] first demonstrated the use of side-channel profiles, such as power consumption and electromagnetic emanation for Trojan circuit detection. The process went as follows: The authors first generated the power signature profiles of a small number of ICs randomly selected from a batch of manufactured ICs. These ICs served as golden masters (Trojan-free ICs). Once profiled the golden masters underwent a rigorous destructive reverse-engineering phase, where they were compared piece by piece against the original design. If Trojan free, the ICs were then accepted as genuine ICs and their profiles served as power templates. The remaining ICs could then be tested efficiently and in a nondestructive manner by simply applying the same stimuli and building their power profiles. The profiles were compared using statistical techniques, such as principal component analysis against the templates obtained from the golden masters.

Side-channel analysis methods may succeed in detecting Trojans to some degree. However, their difficulty lies in achieving high coverage of every gate or net, and in extracting the tiny, abnormal side-channel signals of hardware Trojans in the presence of process and environmental variations. As the feature size of ICs shrinks and the number of transistors continue to increase, the increasing levels of process variations can easily mask the small side-channel signals induced by low-overhead and rarely triggered Trojans. Since they observed that filler cells are more reflective than other functional cells, recently, Zhou et al. [27] presented a backside imaging method to produce a pattern based on filler cells placed in the IC layout. Although this technique does not require golden chip, the comparison between the simulated image and measured optical image still suffers from the variations in the manufacturing process. Additional challenges include taking clear images at higher resolution that takes enormous amount of time.

5.7.1.2 Pre-silicon Trojan Detection

These techniques are used to help SoC developers and design engineers to validate third-party IP (3PIP) cores and their final designs. Existing pre-silicon detection techniques can be broadly classified into code coverage analysis, formal verification, structural analysis, logic testing, and functional analysis.

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 metric 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 an 3PIP. Hicks et al. [13] presented 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. Here, the authors proposed to remove these suspicious lines of RTL

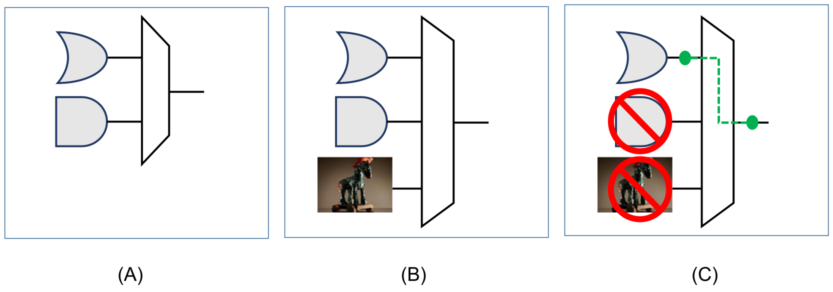

FIGURE 5.7

High-level overview of the unused circuit identification (UCI) technique. (A) designers develop hardware designs, (B) a rogue entity inserts hardware Trojan into the design, and (C) UCI technique identifies and removes suspicious circuits.

code from the hardware design and emulate it in the software level. Figure 5.7 illustrates the high-level flow of the UCI technique. This technique, however, does not guarantee the trustworthiness of a 3PIP. Authors in [28] have demonstrated that hardware Trojans can be designed to defeat UCI technique. This type of Trojans derive their triggering circuits from less likely events to evade detection from code coverage analysis.

Formal Verification: Formal methods, such as symbolic execution [29], model checking [30], and information flow [31] have been traditionally applied to software systems for finding security bugs and improving test coverage. Formal verification has also been shown to be effective in verifying the trustworthiness of 3PIP [32,33]. These approaches are based on the concept of proof-carrying code (PCC) to formally validate the security-related properties of an IP. In these 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 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, thereby indicating the presence of hardware Trojan. Figure 5.8 shows the overview of the PCC-based Trojan detection technique

Rajendran et al. [34] presented 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. It is possible to extract the triggering condition of the Trojan from the reported input patterns. Another similar approach has been shown in [35], which formally verifies unauthorized information leakage in 3PIPs. This technique checks for the property, “does the design leak any sensitive information?”. Due to the problem of space explosion, the limitation of these approaches is that the processing ability of the model checking is relatively limited. Although, these techniques present a promising approach for Trojan detection, each has certain challenges and limitations [36].

Structural Analysis: Structural analysis employs quantitative metrics to mark signals or gates with low activation probability as suspicious. Salmani et al. [37] presented a metric named ‘Statement Hardness’

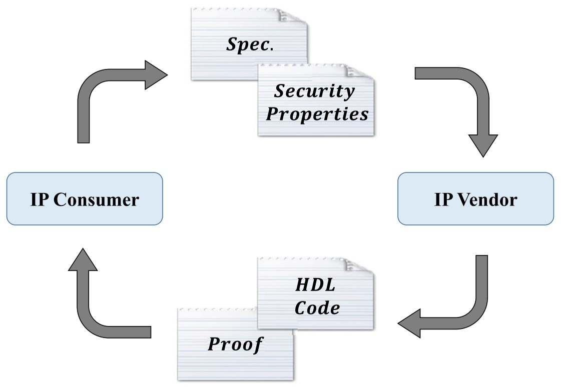

FIGURE 5.8

Overview of proof-carrying code (PCC) based Trojan detection technique.

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 that have low transition probability and are not testable through well-known fault testing techniques, such as stuck-at, transition delay, path delay, and bridging faults [38]. 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. Tehranipoor et al. [5] 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 postprocessing is required to analyze suspicious signals or gates, and determine if they are a part of a Trojan.

Logic Testing: The principal idea of logic testing is the same as the functional tests described earlier in the post-silicon Trojan detection section. The logic testing is conducted with simulation, whereas functional tests have to be performed on a tester for applying input patterns and collecting output responses. Therefore, existing techniques for functional tests are also applicable to logic testing. Of course, logic testing also inherits functional tests' pros and cons.

Functional Analysis: Functional analysis applies random input patterns and performs functional simulation of the IP to find suspicious regions of the IP that 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. Waksman et al. [39] presented 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

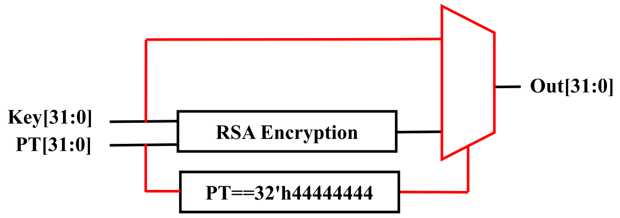

FIGURE 5.9

RSA-T100 Trojan. This Trojan leaks the private key when the plaintext (PT) data is *32'h44444444*.

net is lower than a predefined threshold, then the net is flagged as suspicious. For example, for the RSA-T100 [16] Trojan, the triggering condition is *32'h44444444* (shown in Fig. 5.9). 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 that this approach produces a large number of false-positive results and does not specify any method to verify if the suspicious signals are performing any malicious operation. Also, Zhang et al. [40] have shown how 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 4 cycles, making the 'Control value' of the triggering net 2^{-8}. Furthermore, FANCI cannot identify 'Always On' Trojans, which remain active during their lifetime and do not have any triggering circuitry.

Authors in [41] presented 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 keeps 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-product (SOP) and product-of-sum (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 [40] designed Trojans that can defeat VeriTrust by ensuring that the Trojan-triggering circuit is driven by a subset of functional inputs, in addition to triggering inputs. VeriTrust also shares the same limitation of FANCI, that is, not being able to identify "Always On" Trojans.

5.7.2 DESIGN-FOR-TRUST

As described in the previous section, detecting a quiet, low-overhead hardware Trojan is still very challenging with existing techniques. A potentially more effective way is to plan for the Trojan problem in the design phase through design-for-trust. These methodologies are classified into four classes according to their objectives.

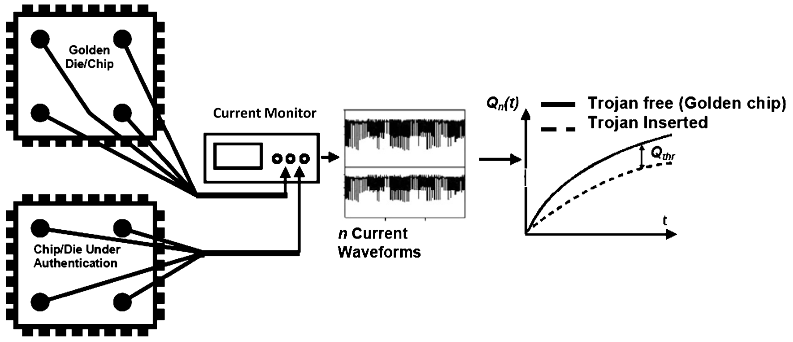

FIGURE 5.10

Current integration (charge) method.

5.7.2.1 Facilitate Detection

Functional Test: Triggering a Trojan from inputs and observing the Trojan effect from outputs are difficult due to the stealthy nature of Trojans. A large number of low-controllable and low-observable nets in a design significantly hinder the possibility of activating a Trojan. Salmani et al. [42] and Zhou et al. [43] attempted to increase controllability and observability of nodes by inserting test points into the circuit. Another approach proposed to multiplex two outputs of a DFF, Q and $\overline{Q}$, through a 2-to-1 multiplexer, and select either of them. This extends the state space of the design and increases the possibility of exciting/propagating the Trojan effects to circuit outputs, making them detectable [18]. These approaches are beneficial not only to functional-test-based detection techniques but also to side-channel-based methods that need partial activation of Trojan circuitry.

Side-channel Signal Analysis: A number of design methods have been developed to increase the sensitivity of side-channel-based detection approaches. The amount of current a Trojan can draw could be so small that it can be submerged into an envelope of noise and process variations effects; therefore, it may be undetectable by conventional measurement equipments. However, Trojan-detection capability can be greatly enhanced by measuring currents locally, and from multiple power ports/pads. Figure 5.10 shows the current (charge) integration methodology for detecting hardware Trojans presented in [44]. Salmani and Tehranipoor [45] proposed to minimize background side-channel signals by localizing switching activities within one region, while minimizing them in other regions through a scan-cell reordering technique. Additionally, some newly developed structures or sensors are implemented in the circuit to provide a higher detection sensitivity compared to conventional measurements. Ring oscillator (RO) structures [46], shadow registers [47], and delay elements [48] on a set of selected short paths are inserted for path delay measurements. RO sensors [49] and transient current sensors [50,51] are able to improve sensitivity to voltage and current fluctuations caused by Trojans, respectively. Besides, integration of process variation sensors [52,53] can calibrate the model or measurement, and minimize the noise induced by manufacturing variations.

Runtime Monitoring: As triggering all types and sizes of Trojans during pre-silicon and post-silicon tests is very difficult, runtime monitoring of critical computations can significantly increase the level of trust with respect to hardware Trojan attacks. These runtime monitoring approaches can utilize existing or supplemental on-chip structures to monitor chips behaviors [10,54] or operating conditions,

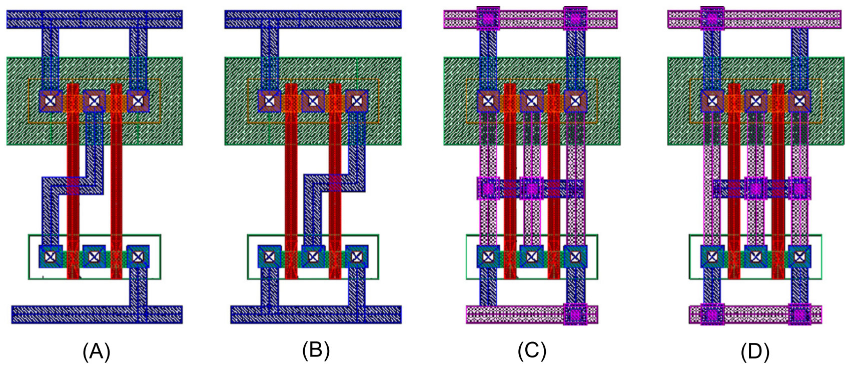

FIGURE 5.11

(A) and (B) show standard cell layout of traditional 2-input NAND and NOR gates, respectively. Here, the metal layers are different and, therefore, easy to differentiate. (C) and (D) show camouflaged standard cell layouts of 2-input NAND and NOR gates, respectively. Note that, the metal layers are identical and, therefore, difficult to distinguish.

such as transient power [50,55] and temperature [25]. They can disable the chip upon detection of any abnormalities or bypass it to allow reliable operation, albeit with some performance overhead. Jin et al. [56] present a design of an on-chip analog neural network that can be trained to distinguish trusted from untrusted circuit functionality based on measurements obtained via on-chip measurement acquisition sensors.

5.7.2.2 Prevent Trojan Insertion

These techniques consist of preventive mechanisms that attempt to thwart hardware Trojan insertion by attackers. To insert targeted Trojans, typically attackers need to understand the function of the design first. Attackers who are not in the design house usually identify circuit functionality by reverse engineering.

Logic Obfuscation: Logic obfuscation attempts to hide the genuine functionality and implementation of a design by inserting built-in locking mechanisms into the original design. The locking circuits become transparent, and the right function appears only when a right key is applied. The increased complexity of identifying the genuine functionality without knowing the right input vectors can lower the ability of inserting a targeted Trojan by attackers. For combinational logic obfuscation, XOR/XNOR gates could be introduced at certain locations in a design [57]. In sequential logic obfuscation, additional states are introduced in a finite state machine to conceal its functional states [19]. In addition, some techniques proposed to insert reconfigurable logics for logic obfuscation [58,59]. The design is functional when the reconfigurable circuits are correctly programmed by the design house or end-user.

Camouflaging: Camouflaging is a layout-level obfuscation technique to create indistinguishable layouts for different gates by adding dummy contacts, and faking connections between the layers within a camouflaged logic gate [60,61] (shown in Fig. 5.11). The camouflaging technique can hinder attackers from extracting a correct gate-level netlist of a circuit from the layout through imaging different layers; in that way, the original design is protected from insertion of targeted Trojans. Additionally, Bi et al.

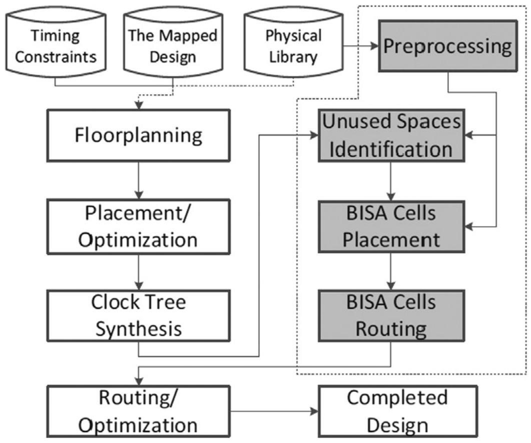

FIGURE 5.12

The BISA insertion flow.

[62] utilized a similar dummy contact approach and developed a set of camouflaging cells based on polarity-controllable SiNW FETs.

Functional Filler Cell: Since layout design tools are typically conservative in placement, they cannot fill 100% of the area with regular standard cells in a design. The unused spaces are usually filled with filler cells or decap cells that do not have any functionality. Filler cells are usually used during engineering change order (ECO) for improving debug and yield, whereas decaps are used to manage peak current in the chip, especially in areas where instantaneous power is quite significant. Thus, the most covert way for attackers to insert Trojans in a circuit layout is replacing filler cells, and to some degree decaps, because removing these nonfunctional cells has the smallest impact on electrical parameters. The built-in self-authentication (BISA) approach fills all white spaces with functional filler cells during layout design [63]. The inserted cells are then connected automatically to form a combinational circuitry that could be tested. A failure during later testing denotes that a functional filler has been replaced by a Trojan. Figure 5.12 shows the general BISA insertion flow. The white rectangles in Fig. 5.12 are conventional ASIC design flow, whereas the dark rectangles represent the additional steps required for BISA. These steps are as follows: (i) preprocessing (gather detailed information about the standard cell library), (ii) unused space identification, (iii) BISA cell placement, and (iv) BISA cell routing.

5.7.2.3 Trustworthy Computing

The third class of design for trust is trustworthy computing on untrusted components. The difference between runtime monitoring and trustworthy computing is that trustworthy computing is tolerant to Trojan attacks by design. Trojan detection and recovery at runtime—acting as the last line of defense—is necessary, especially for mission-critical applications. Some approaches employ a distributed software scheduling protocol to achieve a Trojan-activation-tolerant trustworthy computing system in a multicore processor [64,65]. Concurrent Error Detection (CED) techniques can be adapted to detect malicious outputs generated by Trojans [66,67]. In addition, Reece et al. [68] and Rajendran et al. [67] proposed to use a diverse set of 3PIP vendors to prevent Trojan's effects. The technique in [68] verifies the integrity of a design via comparison of multiple 3PIPs with another untrusted design performing a similar function. Rajendran et al. [67] utilize operation-to-3PIP-to-vendor allocation constraints to prevent collusions between 3PIPs from the same vendor.

For the design for trust (DFT) techniques that require circuitry added during the front-end design phase, the potential area and performance overheads are the chief concerns to designers. As the size of a circuit increases, the number of quiet (low controllability/observability) nets/gates will increase the complexity of processing and produce a large time/area overhead. Thus, the DFT techniques for facilitating detection are still difficult to apply to a large design that contains millions of gates. In addition, the preventive DFT techniques need to insert additional gates (logic obfuscation) or modify the original standard cells (camouflaging), which could degrade the chip performance significantly, and affect their acceptability in high-end circuits. The functional filler cells also increase power leakage.

5.7.2.4 Split-Manufacturing for Hardware Trust

Split-manufacturing has been proposed recently as an approach to enable use of state-of-the-art semiconductor foundries while minimizing the risks to an IC design [69]. Split manufacturing divides a design into Front End of Line (FEOL) and Back End of Line (BEOL) portions for fabrication by different foundries. An untrusted foundry performs FEOL manufacturing (higher-cost), then ships wafers to a trusted foundry for BEOL fabrication (lower-cost; shown in Fig. 5.13). The untrusted foundry does not have access to the layers in BEOL and, thus, cannot identify the "safe" places within a circuit to insert Trojans.

Existing split manufacturing processes rely on either 2D integration [70–72], 2.5D integration [73], or 3D integration [74]. The 2.5D integration first splits a design into two chips fabricated by the untrusted foundry and then inserts a silicon interposer containing interchip connections between the chip and package substrate [73]. Therefore, a portion of interconnections could be hidden in the interposer that is fabricated in the trusted foundry. In essence, it is a variant of 2D integration for split manufacturing. During the 3D integration, a design is split into two tiers fabricated by different foundries. One tier is stacked on top the other, and the upper tiers are connected with vertical interconnects called TSVs. Given the manufacturing barriers to 3D in industry, 2D- and 2.5D-based split manufacturing techniques are more realistic today. Vaidyanathan et al. [75] demonstrate the feasibility of split fabrication after metal 1 (M1) on test chips and evaluated the chip performance. Although the split after M1 attempts to hide all intercell interconnections and can obfuscate the design effectively, it leads to high manufacturing costs. Additionally, several design techniques have been proposed to enhance a design's security with split manufacturing. Imeson et al. [76] present a k-security metric to select necessary wires to be lifted to a trusted tier (BEOL) to ensure the security when split at a higher layer. However, lifting a large number of wires in the original design introduces large timing and power overhead and significantly impact chip performance. An obfuscated BISA (OBISA) technique can insert dummy circuits into the original design to further obfuscate the design with split manufacturing [77].

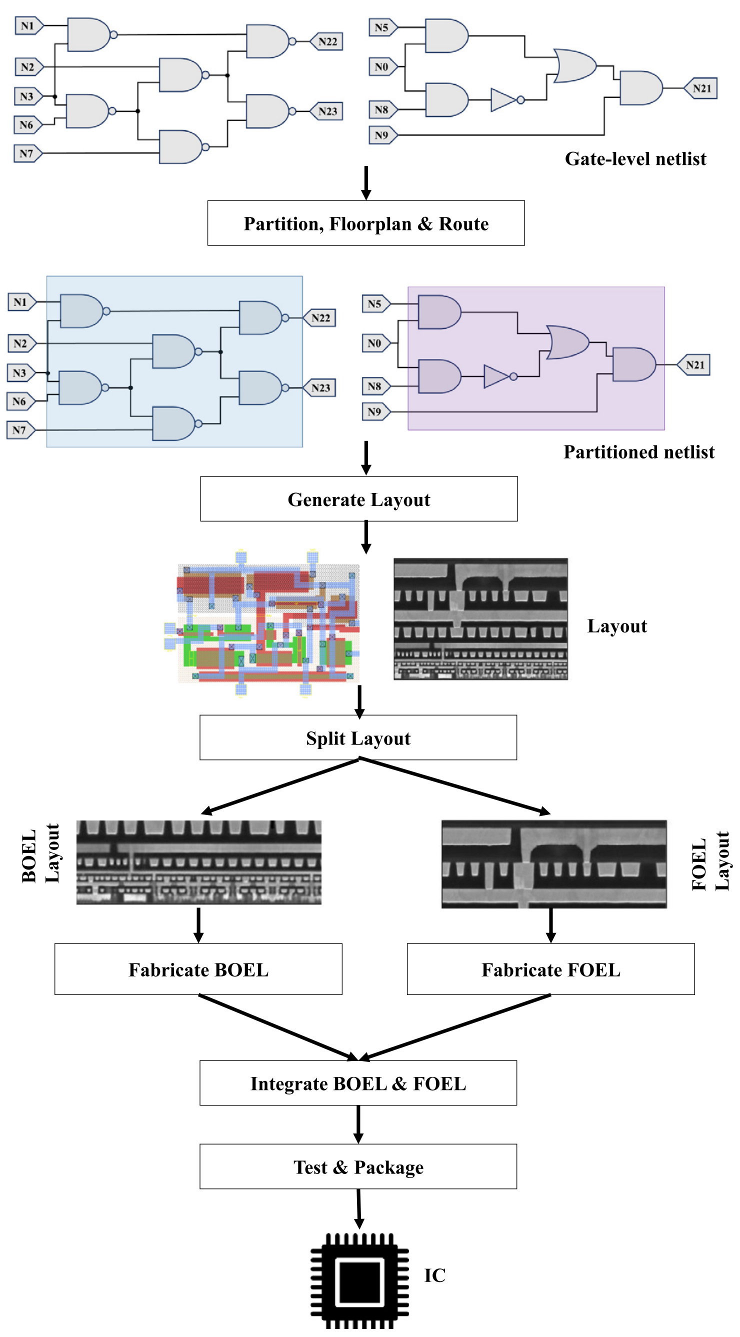

FIGURE 5.13

Split-manufacturing based IC design flow.

5.8 HANDS-ON EXPERIMENT: HARDWARE TROJAN ATTACKS

5.8.1 OBJECTIVE

This experiment is designed to give students an exposure to various forms of hardware Trojan attacks.

5.8.2 METHOD

The experiment is composed of several parts, all of which are designed on the HaHa platform. The first part of the experiment deals with mapping a design, namely a Data Encryption Standard (DES) module, i.e., a symmetric-key block cipher module, into the FPGA chip of the HaHa board. The second part illustrates the combinational Trojan design in the DES module. Students have to incorporate a malicious design modification to trigger malfunction based on a combination trigger condition. The second part of the experiment deals with designing a sequential Trojan instance in the same module.

5.8.3 LEARNING OUTCOME

By performing the specific steps of the experiments, students will learn different types of Trojan design, how they activate, and how they cause malicious impacts. They will also experience the challenges with respect to protecting a design against Trojan attacks.

5.8.4 ADVANCED OPTIONS

Additional exploration on this topic can be done through design of more complex Trojans, e.g., ones that can be triggered by temperature and ones that leak information (protection mechanisms were not included in the experiment).

More details about the experiment are available in the supplementary document. Please visit: http://hwsecuritybook.org.

5.9 EXERCISES

5.9.1 TRUE/FALSE QUESTIONS

1. 100% fault coverage ensures Trojan detection.
2. Trojan triggers are derived from low transition probability nets.
3. All hardware Trojans have a trigger.
4. Foundry is a trusted entity in the SoC design flow.
5. In general, sequential Trojans are more difficult to trigger compared to the combinational Trojans.
6. Trojans cannot be inserted after chip fabrication.

5.9.2 LONG-ANSWER TYPE QUESTIONS

1. Describe the motives of the semiconductor industry to shift to a horizontal business model. How does the horizontal business model introduce the risk of hardware Trojan insertion?

2. Who are the potential adversaries to implant a hardware Trojan? Provide a brief description on each of them. In your opinion, which one is the most difficult to defend?
3. Provide a brief description of a generic Trojan structure.
4. Describe the difference between a combinational and a sequential Trojan.
5. Provide an example of hardware Trojan in a cryptomodule.
6. Give a comparison of hardware Trojans in ASIC and FPGA designs.
7. Illustrate the Trojan taxonomy.
8. Describe the classification of Trojans based on the "Activation Mechanism".
9. Describe the classification of Trojans based on the "Payload".
10. Illustrate the taxonomy of Trojan countermeasures.
11. Briefly describe the built-in self-authentication (BISA) technique. Consider the following scenario: A rogue foundry wants to insert Trojan in BISA-protected design. The foundry wants to replace higher driving strength gates with the same functionality lower driving strength gates, for example, replace 8x buffer with 1x buffer, to make room for Trojan cell insertion. Give your opinion if such attacks can be initiated.

5.9.3 MATHEMATICAL PROBLEMS

1. The RSA-T100 Trojan is triggered when a 32-bit specific plaintext is applied. Calculate the probability of triggering this Trojan if one uses random patterns as plaintext.
2. The AES-T1000 Trojan is triggered when a 128-bit specific plaintext is applied. Calculate the probability of triggering this Trojan if one uses random patterns as plaintext.
3. The AES-T1100 Trojan is triggered when four specific 128-bit plaintext is applied in a specific order. Calculate the probability of triggering this Trojan if one uses random patterns as plaintext.

REFERENCES

[1] S. Adee, The hunt for the kill switch, IEEE Spectrum 45 (2008) 34–39.
[2] J. Markoff, Old trick threatens the newest weapons, The New York Times (2009), https://www.nytimes.com/2009/10/27/science/27trojan.html.
[3] A. Nahiyan, M. Tehranipoor, Code coverage analysis for IP trust verification, in: Hardware IP Security and Trust, Springer, 2017, pp. 53–72.
[4] M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection, IEEE Design & Test of Computers 27 (2010).
[5] M. Tehranipoor, H. Salmani, X. Zhang, Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection, Springer Science & Business Media, 2013.
[6] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research, ACM Transactions on Design Automation of Electronic Systems 22 (2016) 6.
[7] R.S. Chakraborty, F.G. Wolff, S. Paul, C.A. Papachristou, S. Bhunia, MERO: a statistical approach for hardware Trojan detection, in: CHES, vol. 5747, Springer, 2009, pp. 396–410.
[8] L. Lin, W. Burleson, C. Paar, MOLES: malicious off-chip leakage enabled by side-channels, in: Proceedings of the 2009 International Conference on Computer-Aided Design, ACM, pp. 117–122.
[9] Y. Liu, Y. Jin, Y. Makris, Hardware Trojans in wireless cryptographic ICs: silicon demonstration & detection method evaluation, in: Proceedings of the International Conference on Computer-Aided Design, IEEE Press, pp. 399–404.
[10] G. Bloom, B. Narahari, R. Simha, OS support for detecting Trojan circuit attacks, in: Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on, IEEE, pp. 100–103.

[11] S.T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, Y. Zhou, Designing and implementing malicious hardware, in: LEET'08, 2008, pp. 1–8.
[12] S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures, Proceedings of the IEEE 102 (2014) 1229–1247.
[13] M. Hicks, M. Finnicum, S.T. King, M.M. Martin, J.M. Smith, Overcoming an untrusted computing base: detecting and removing malicious hardware automatically, in: Security and Privacy (SP), 2010 IEEE Symposium on, IEEE, pp. 159–172.
[14] S. Mal-Sarkar, A. Krishna, A. Ghosh, S. Bhunia, Hardware Trojan attacks in FPGA devices: threat analysis and effective counter measures, in: Proceedings of the 24th edition of the Great Lakes Symposium on VLSI, ACM, pp. 287–292.
[15] R. Karri, J. Rajendran, K. Rosenfeld, M. Tehranipoor, Trustworthy hardware: identifying and classifying hardware trojans, Computer 43 (10) (2010) 39–46.
[16] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, M. Tehranipoor, Benchmarking of hardware Trojans and maliciously affected circuits, Journal of Hardware and Systems Security (2017) 1–18.
[17] C. Bao, D. Forte, A. Srivastava, On application of one-class SVM to reverse engineering-based hardware Trojan detection, in: Quality Electronic Design (ISQED), 2014 15th International Symposium on, IEEE, pp. 47–54.
[18] M. Banga, M.S. Hsiao, A novel sustained vector technique for the detection of hardware Trojans, in: VLSI Design, 2009 22nd International Conference on, IEEE, pp. 327–332.
[19] R.S. Chakraborty, S. Bhunia, Security against hardware Trojan through a novel application of design obfuscation, in: Proceedings of the 2009 International Conference on Computer-Aided Design, ACM, pp. 113–116.
[20] X. Wang, M. Tehranipoor, J. Plusquellic, Detecting malicious inclusions in secure hardware: challenges and solutions, in: Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, IEEE, pp. 15–19.
[21] Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in: Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, IEEE, pp. 51–57.
[22] K. Xiao, X. Zhang, M. Tehranipoor, A clock sweeping technique for detecting hardware Trojans impacting circuits delay, IEEE Design & Test 30 (2013) 26–34.
[23] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using IC fingerprinting, in: Security and Privacy, 2007. SP'07, IEEE Symposium on, IEEE, pp. 296–310.
[24] J. Aarestad, D. Acharyya, R. Rad, J. Plusquellic, Detecting Trojans through leakage current analysis using multiple supply pads, IEEE Transactions on Information Forensics and Security 5 (2010) 893–904.
[25] D. Forte, C. Bao, A. Srivastava, Temperature tracking: an innovative run-time approach for hardware Trojan detection, in: Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on, IEEE, pp. 532–539.
[26] F. Stellari, P. Song, A.J. Weger, J. Culp, A. Herbert, D. Pfeiffer, Verification of untrusted chips using trusted layout and emission measurements, in: Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on, IEEE, pp. 19–24.
[27] B. Zhou, R. Adato, M. Zangeneh, T. Yang, A. Uyar, B. Goldberg, S. Unlu, A. Joshi, Detecting hardware Trojans using backside optical imaging of embedded watermarks, in: Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, IEEE, pp. 1–6.
[28] C. Sturton, M. Hicks, D. Wagner, S.T. King, Defeating UCI: building stealthy and malicious hardware, in: Security and Privacy (SP), 2011 IEEE Symposium on, IEEE, pp. 64–77.
[29] C. Cadar, D. Dunbar, D.R. Engler, et al., KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs, in: OSDI, vol. 8, pp. 209–224.
[30] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu, Symbolic model checking using SAT procedures instead of BDDs, in: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, ACM, pp. 317–320.
[31] A.C. Myers, B. Liskov, A Decentralized Model for Information Flow Control, vol. 31, ACM, 1997.
[32] Y. Jin, B. Yang, Y. Makris, Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing, in: Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on, IEEE, pp. 99–106.
[33] X. Guo, 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, p. 145.
[34] J. Rajendran, V. Vedula, R. Karri, Detecting malicious modifications of data in third-party intellectual property cores, in: Proceedings of the 52nd Annual Design Automation Conference, ACM, p. 112.
[35] J. Rajendran, A.M. Dhandayuthapany, V. Vedula, R. Karri, Formal security verification of third party intellectual property cores for information leakage, in: VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on, IEEE, pp. 547–552.

[36] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, M. Tehranipoor, Hardware Trojan detection through information flow security verification, in: International Test Conference (DAC), 2017, IEEE, pp. 1–6.

[37] H. Salmani, M. Tehranipoor, Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level, in: Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on, IEEE, pp. 190–195.

[38] H. Salmani, M. Tehranipoor, R. Karri, On design vulnerability analysis and trust benchmarks development, in: Computer Design (ICCD), 2013 IEEE 31st International Conference on, IEEE, pp. 471–474.

[39] A. Waksman, M. Suozzo, S. Sethumadhavan, FANCI: identification of stealthy malicious logic using boolean functional analysis, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, pp. 697–708.

[40] J. Zhang, F. Yuan, Q. Xu, DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp. 153–166.

[41] J. Zhang, F. Yuan, L. Wei, Y. Liu, Q. Xu, VeriTrust: verification for hardware trust, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34 (2015) 1148–1161.

[42] H. Salmani, M. Tehranipoor, J. Plusquellic, A novel technique for improving hardware Trojan detection and reducing Trojan activation time, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 20 (2012) 112–125.

[43] B. Zhou, W. Zhang, S. Thambipillai, J. Teo, A low cost acceleration method for hardware Trojan detection based on fan-out cone analysis, in: Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference on, IEEE, pp. 1–10.

[44] X. Wang, H. Salmani, M. Tehranipoor, J. Plusquellic, Hardware Trojan detection and isolation using current integration and localized current analysis, in: Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08, IEEE International Symposium on, IEEE, pp. 87–95.

[45] H. Salmani, M. Tehranipoor, Layout-aware switching activity localization to enhance hardware Trojan detection, IEEE Transactions on Information Forensics and Security 7 (2012) 76–87.

[46] J. Rajendran, V. Jyothi, O. Sinanoglu, R. Karri, Design and analysis of ring oscillator based design-for-trust technique, in: VLSI Test Symposium (VTS), 2011 IEEE 29th, IEEE, pp. 105–110.

[47] J. Li, J. Lach, At-speed delay characterization for IC authentication and Trojan Horse detection, in: Hardware-Oriented Security and Trust, 2008. HOST 2008, IEEE International Workshop on, IEEE, pp. 8–14.

[48] A. Ramdas, S.M. Saeed, O. Sinanoglu, Slack removal for enhanced reliability and trust, in: Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2014 9th IEEE International Conference on, IEEE, pp. 1–4.

[49] X. Zhang, M. Tehranipoor, RON: an on-chip ring oscillator network for hardware Trojan detection, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, IEEE, pp. 1–6.

[50] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, S. Bhunia, Improving IC security against Trojan attacks through integration of security monitors, IEEE Design & Test of Computers 29 (2012) 37–46.

[51] Y. Cao, C.-H. Chang, S. Chen, Cluster-based distributed active current timer for hardware Trojan detection, in: Circuits and Systems (ISCAS), 2013 IEEE International Symposium on, IEEE, pp. 1010–1013.

[52] B. Cha, S.K. Gupta, Efficient Trojan detection via calibration of process variations, in: Test Symposium (ATS), 2012 IEEE 21st Asian, IEEE, pp. 355–361.

[53] Y. Liu, K. Huang, Y. Makris, Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting, in: Proceedings of the 51st Annual Design Automation Conference, ACM, pp. 1–6.

[54] J. Dubeuf, D. Hély, R. Karri, Run-time detection of hardware Trojans: the processor protection unit, in: Test Symposium (ETS), 2013 18th IEEE European, IEEE, pp. 1–6.

[55] Y. Jin, D. Sullivan, Real-time trust evaluation in integrated circuits, in: Proceedings of the Conference on Design, Automation & Test in Europe, European Design and Automation Association, p. 91.

[56] Y. Jin, D. Maliuk, Y. Makris, Post-deployment trust evaluation in wireless cryptographic ICs, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, IEEE, pp. 965–970.

[57] J.A. Roy, F. Koushanfar, I.L. Markov, Ending piracy of integrated circuits, Computer 43 (2010) 30–38.

[58] A. Baumgarten, A. Tyagi, J. Zambreno, Preventing IC piracy using reconfigurable logic barriers, IEEE Design & Test of Computers 27 (2010).

[59] J.B. Wendt, M. Potkonjak, Hardware obfuscation using PUF-based logic, in: Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design, IEEE Press, pp. 270–277.

[60] J. Rajendran, M. Sam, O. Sinanoglu, R. Karri, Security analysis of integrated circuit camouflaging, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, pp. 709–720.

[61] R.P. Cocchi, J.P. Baukus, L.W. Chow, B.J. Wang, Circuit camouflage integration for hardware IP protection, in: Proceedings of the 51st Annual Design Automation Conference, ACM, pp. 1–5.
[62] Y. Bi, P.-E. Gaillardon, X.S. Hu, M. Niemier, J.-S. Yuan, Y. Jin, Leveraging emerging technology for hardware security-case study on silicon nanowire FETs and graphene SymFETs, in: Test Symposium (ATS), 2014 IEEE 23rd Asian, IEEE, pp. 342–347.
[63] K. Xiao, M. Tehranipoor, BISA: Built-in self-authentication for preventing hardware Trojan insertion, in: Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on, IEEE, pp. 45–50.
[64] D. McIntyre, F. Wolff, C. Papachristou, S. Bhunia, Trustworthy computing in a multi-core system using distributed scheduling, in: On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International, IEEE, pp. 211–213.
[65] C. Liu, J. Rajendran, C. Yang, R. Karri, Shielding heterogeneous MPSoCs from untrustworthy 3PIPs through security-driven task scheduling, IEEE Transactions on Emerging Topics in Computing 2 (2014) 461–472.
[66] O. Keren, I. Levin, M. Karpovsky, Duplication based one-to-many coding for Trojan HW detection, in: Defect and Fault Tolerance in VLSI Systems (DFT), 2010 IEEE 25th International Symposium on, IEEE, pp. 160–166.
[67] J. Rajendran, H. Zhang, O. Sinanoglu, R. Karri, High-level synthesis for security and trust, in: On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International, IEEE, pp. 232–233.
[68] T. Reece, D.B. Limbrick, W.H. Robinson, Design comparison to identify malicious hardware in external intellectual property, in: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, IEEE, pp. 639–646.
[69] Trusted integrated circuits (TIC) program announcement, http://www.iarpa.gov/solicitations_tic.html, 2011, [Online].
[70] K. Vaidyanathan, B.P. Das, L. Pileggi, Detecting reliability attacks during split fabrication using test-only BEOL stack, in: Proceedings of the 51st Annual Design Automation Conference, ACM, pp. 1–6.
[71] M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, M. Fritze, Split-fabrication obfuscation: metrics and techniques, in: Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on, IEEE, pp. 7–12.
[72] B. Hill, R. Karmazin, C.T.O. Otero, J. Tse, R. Manohar, A split-foundry asynchronous FPGA, in: Custom Integrated Circuits Conference (CICC), 2013 IEEE, IEEE, pp. 1–4.
[73] Y. Xie, C. Bao, A. Srivastava, Security-aware design flow for 2.5D IC technology, in: Proceedings of the 5th International Workshop on Trustworthy Embedded Devices, ACM, pp. 31–38.
[74] J. Valamehr, T. Sherwood, R. Kastner, D. Marangoni-Simonsen, T. Huffmire, C. Irvine, T. Levin, A 3-D split manufacturing approach to trustworthy system development, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32 (2013) 611–615.
[75] K. Vaidyanathan, B.P. Das, E. Sumbul, R. Liu, L. Pileggi, Building trusted ICs using split fabrication, in: 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 1–6.
[76] F. Imeson, A. Emtenan, S. Garg, M.V. Tripunitara, Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation, in: USENIX Security Symposium, pp. 495–510.
[77] K. Xiao, D. Forte, M.M. Tehranipoor, Efficient and secure split manufacturing via obfuscated built-in self-authentication, in: Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on, IEEE, pp. 14–19.