CHAPTER 14

HARDWARE OBFUSCATION

CONTENTS

14.1 INTRODUCTION

Hardware obfuscation is a method of modifying a design, so that it becomes significantly difficult to reverse engineer, or copy. Figure 14.1 provides high-level view of the hardware obfuscation process, which transforms a design with respect to its functional behavior, and its structural representation. The

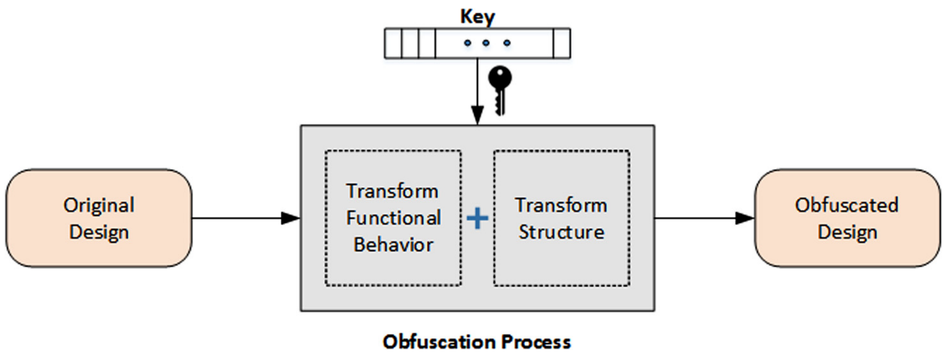


FIGURE 14.1

High level view of the hardware obfuscation process.

transformation process requires a "key", which is used to lock the design. An obfuscated design works in two modes. On application of the correct key, it gets "unlocked", and operates in normal mode, that is, it produces its normal functional behavior. With an incorrect key, it remains "locked"; it works in obfuscated mode, and produces wrong outputs.

Hardware obfuscation is an active field of research. It presents a fundamentally different approach to IP protection than existing ones. Existing approaches include passive solutions, such as patenting, copyrighting, and watermarking, which provide a mechanism to prove and claim ownership of an IP in the court of law, when an IP infringement is suspected. Existing security approaches fail to provide an active protection of an IP against piracy and reverse engineering in untrusted facilities. To address the limitations of existing approaches, over the past decade, hardware obfuscation has been studied as a promising defense mechanism that can ensure security of hardware IPs along the supply chain. In particular, its effectiveness has been studied on the basis of three major threats on hardware IP: (1) reverse engineering, (2) piracy, and (3) malicious modifications (i.e., hardware Trojan attacks, as described in Chapter 5). Although the majority of obfuscation research is aimed at addressing the first two threats, some methods have also been shown to provide protection against Trojan attacks in untrusted foundry. Obfuscation has been investigated for both ASIC and FPGA design flows, and for different levels of abstraction, for example, gate-level and register-transfer level IPs. Similar to encryption, the security model here relies on the secrecy of the key, not the algorithm used for obfuscation.

Hardware obfuscation requires application of a systematic set of transformations intended to provide provably robust protection. Figure 14.2 illustrates the major goals of the design transformation, and how they are achieved. One major goal is preventing black-box usage, which indicates that the transformed IP cannot be used as a "black-box" in a larger design, for instance, in an SoC to provide the desired functionality. This is achieved through the locking mechanism. The locking is accomplished by inserting specialized logic structures or gates controlled by the key that produce incorrect functionality for wrong keys, and vice versa. A strong lock, that is robust against functional or structural analysis attacks, is desired for a strong obfuscation. The second major goal is to conceal the design intent through judicious structural transformation. This is essential for two reasons: (1) it protects the lock itself, since a lock, which can be easily identified, can also be removed easily; and (2) it prevents leakage of design secrets, such as type of

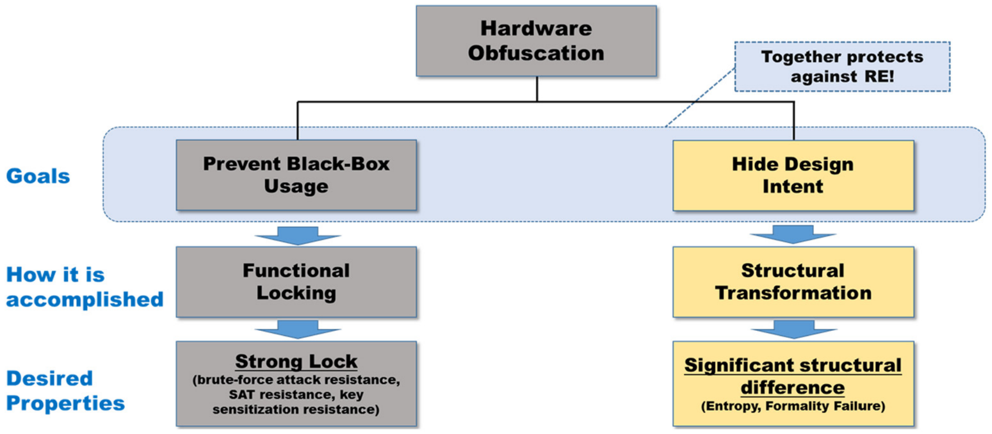

Hardware Obfuscation
Together protects against RE!
Goals
Prevent Black-Box Usage
Hide Design Intent
How it is accomplished
Functional Locking
Structural Transformation
Desired Properties
Strong Lock
(brute-force attack resistance, SAT resistance, key sensitization resistance)
Significant structural difference
(Entropy, Formality Failure)

FIGURE 14.2

Main goals of hardware obfuscation and properties to assess the obfuscation.

function realized and, logic style, which may provide important clues about a design to potential adversaries. Even partial extraction of design knowledge can be extremely useful for an adversary. For example, improper obfuscation of an arithmetic logic unit can reveal important features of the design, such as the overflow mechanism or pipeline structure. Hardware for many applications (for instance, digital signal processing, graphics, etc.) have very regular logic structure. Obfuscation for them presents even more difficult challenges with respect to hiding design intent.

14.1.1 PRELIMINARIES

In this section, we present definitions of obfuscation for both software and hardware, and point out the differences between them. We also talk about the advantages of hardware IP obfuscation over encryption.

14.1.1.1 Definition

The term "obfuscation" represents the method of obscuring or covering the actual substance of an information or the functional behavior of a product to protect the inherent intellectual property. In cryptology and software, an obfuscator Z is formally characterized as a "compiler" that reconstructs a program P to an obfuscated form $Z(P)$. $Z(P)$ must have the same functionality as P while being incomprehensible to an attacker aiming to construct P from $Z(P)$.

Obfuscation, in the context of a hardware design, i.e., "hardware obfuscation" is concerned with the protection of hardware IPs. These IPs are reusable block of logic, memory, or analog circuits, which are owned by their developers, and used by themselves, or other SoC design houses. Though the techniques for obfuscating hardware and software differ significantly, the primary goal of obfuscation remains unchanged: protection of IP from bad actors that are capable of piracy, reverse engineering, and malicious modification.

14.1.1.2 Software Versus Hardware Obfuscation

Software obfuscation primarily focuses on obscuring the implementation of the algorithm represented as code. Software obfuscation relies on various techniques, ranging from the addition of simple comments, changing symbol names, or the removal of white spaces to more sophisticated methods, such as modification of the program control-flow through loop unrolling [1].

Whereas a software can only be obtained either as code or a compiled binary, hardware can be obtained in many forms, including architecture-level description, RTL description, netlist, layout, fabricated chip, and FPGA configuration bitstream. As long as the hardware description can be found in textual representation (for example, RTL, gate-level netlist), many of the software obfuscation solutions can be applied to them. However, they may not provide adequate protection against aforementioned threat models. Once the hardware design is represented as an image (for instance, GDSII layout) or physical form (fabricated ICs), a completely different approach is required. For instance, the RTL code of a hardware IP could be obfuscated by leveraging some form of keyless software obfuscation method, such as redundant code injection. However, once the design is synthesized, the redundant portion that does not impact the actual functionality of the circuit could get eliminated, due to the inherent optimization goals of the synthesis tool. This would allow an attacker to observe the original functionality in synthesized representations of the hardware.

Electronic system design lifecycle is significantly different compared to its software counterpart. Unlike the software development process, hardware design, manufacturing, and testing process often require access to the hardware design, for example, the entire SoC. It gives opportunity to an adversary in an untrusted design/fabrication/test facility to have complete access to the design, and hence makes all IPs vulnerable to piracy and RE. Finally, for distributing software to evaluators and selling them to end-users, appropriate licensing mechanisms (for instance, node-locked licensing) can be used. However, such licensing approaches for hardware IPs to prevent piracy are difficult to accomplish. We discuss this topic in more detail in the next section.

14.1.2 WHY NOT ENCRYPT HARDWARE IPs?

Encryption provides a secure mechanism for IP delivery where robustness is mathematically provable. One might argue that good encryption algorithms (for example, Advanced Encryption Standard or AES) could be used to securely deliver and use hardware IPs in an untrusted supply chain. Encryption could be applied to protect hardware IPs in certain parts of the supply chain with commensurate support from CAD tools. In particular, it can work effectively to protect IPs in an FPGA design flow, where vendor-specific toolset manages the encryption/decryption process. However, in many stages of the supply chain, where an IP is utilized as a white-box, for example, synthesis and physical design, encryption does not apply. For instance, to be able to create the mask of a design, the actual GDSII file must be provided to the foundry. It could be encrypted during transportation to the untrusted foundry, but before fabrication, the IP must be decrypted, and the foundry would have access to the original (unencrypted) form. The same applies to DFT insertion facility that is responsible for inserting design for test resources (such as scan chain and test points) within a design.

During the functional simulation and FPGA prototyping of a soft IP, the design could come in encrypted form. However, the CAD tool, responsible for simulation and synthesis of the design, must obtain the encryption key. This is because only the decrypted version of the design maintains the functional and structural properties of a design that are required to simulate or synthesize it. Under

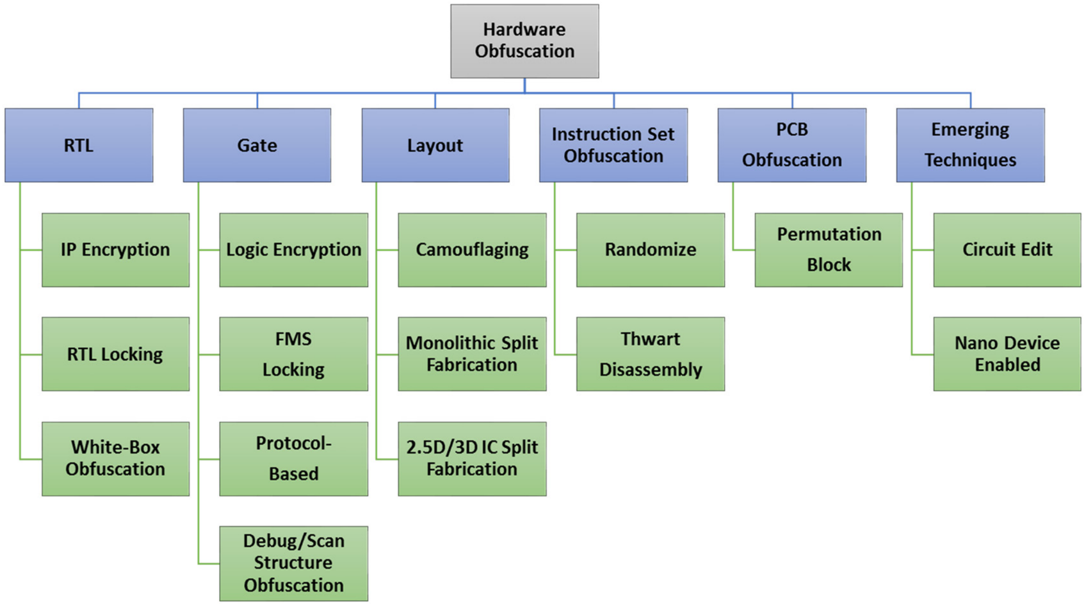


FIGURE 14.3

A taxonomy of hardware obfuscation techniques.

this framework, an IP is protected as long as the CAD tool responsible for key management is not compromised. Hence, due to the requirement of white-box usage of an IP at various untrusted stages in the supply chain, encryption is not considered as a viable solution for hardware IP protection.

14.2 OVERVIEW OF OBFUSCATION TECHNIQUES

Researchers have developed various hardware obfuscation techniques over the past decade. A taxonomy of these techniques is presented in Fig. 14.3, and briefly described below.

14.2.1 RTL OBFUSCATION

Register-transfer level IPs, that is, soft IPs, use high-level constructs to describe an IP using HDL, such as, Verilog or VHDL. Obfuscation of soft IPs typically represent more difficult challenges than their gate-level counterparts. This is because RTL constructs are easier to interpret in terms of the control and data flow, which makes hiding design intent more difficult. IP encryption has been generally used to protect soft IPs. In this method, the entire soft IP can be encrypted by common encryption techniques, such as AES or RSA. Key management is usually handled by the EDA tools (which are assumed to be trusted), and a design house that legally acquires an encrypted IP simply uses it in a design as a black-box.

Apart from encryption, various key-based obfuscation approaches have been studied for protection of soft IPs [2,3]. The RTL code of an IP could be first transformed into a control and data flow graph (CDFG) [2] or state transition graph [3]. The graph could then be modified by inserting "key states", that is, additional states that must be traversed on application of a specific input sequence or key to produce normal functional behavior. On application of a wrong key, it remains stuck in a nonfunctional, obfuscated mode. Soft IPs can also be obfuscated in terms of its intelligibility and readability, similar to traditional software obfuscation approaches. Techniques such as loop unrolling, change in net name, and reordering of statements could be applied to render an RTL code unintelligible, yet functionally identical to the original code [4]. It is called white-box obfuscation, since it does not incorporate functional or structural transformations.

14.2.2 GATE LEVEL OBFUSCATION

A gate-level IP, referred to as firm IP, comes in the form of a netlist. A netlist is a collection of standard logic cells and the nets, which represent connections between cells. A gate-level obfuscation technique requires insertion of extra gates, such as XOR, XNOR, and MUX into the netlist of a design [5,6]. These gates (and their logical output) are controlled by key bits, which serve as additional inputs to the netlist. These key bits can be stored in a tamper-resistant nonvolatile memory inside a chip or be derived from physical unclonable functions (described later in Chapter 12). Correct logic values ("1" or "0") at the key bits ensure intended functional behavior for the netlist. Without the correct values at the key bits, logic gates controlled by them generate wrong logic values at the internal circuit nodes, which lead to faulty outputs for the netlist. The process of controlling selected internal nodes of a circuit by external inputs is very similar to controllable test point insertion [7], which has been traditionally used to improve testability in a design. The internal nodes, which are connected to the key bits are strategically chosen to maximize output corruption (for wrong keys) and to increase structural changes across the netlist. A detailed study of such gate-level obfuscation techniques is presented in Section 14.3.

Another class of gate-level obfuscation techniques focuses on obfuscating the state space for IP protection. These techniques are applicable to gate-level sequential circuits. They transform the state transition function of the underlying finite state machine (FSM) [8]. The transformation serves the following two purposes: a) First, it locks the FSM, such that without application of a specific input sequence that serves as the key, the FSM remains locked, that is, does not produce correct functional behavior; and b) With application of correct key, the state machine is "unlocked", that is, it transitions to normal mode of operation. The locking of the FSM is accomplished by integrating an obfuscation FSM (OFSM) into the original FSM, which creates an FSM that operates into two modes: (1) obfuscation mode, where the state machine transitions through wrong states; and (2) normal mode, where it transitions through correct states. The approach uses the locked states of the FSM to transform the internal nodes of the combinational logic associated with the state machine. It is done in a very similar manner as the gate-level obfuscation techniques described above. Instead of taking the key values from additional key bits, it generates the enabling key values from the state elements. The key values drive "modification cells", as in the gate-level obfuscation, and create faulty internal values in the combinational logic for wrong key.

Sequential obfuscation has also been used to obfuscate the scan-chain of a design. As described earlier, scan-chains are DFT structures inserted into a design to improve its testability. It is important to enable security of scan-chain, such that they cannot be used by unauthorized user to leak secrets on chip, which are accessible through the scan chain [9]. Various obfuscation techniques have been developed to prevent such test infrastructure exploits. For example, researchers have proposed the use of test compression structures that compress the value of several scan flip-flops into a single output, thereby making the observation of individual FF values infeasible. Locking techniques have also been proposed that allow the designer to scramble the responses of the scan-chain (chain of scan FFs), unless a secret key is applied. More details on protection of scan structure is presented in Chapter 9.

14.2.3 LAYOUT LEVEL OBFUSCATION

Layout-level IPs, also known as hard IPs, come in the form of physical layout of a design. It consists of geometrical and spatial information related to a specific fabrication process. Such information is directly used in the fabrication of chips by a foundry. In order to protect the layout from piracy and possible malicious alterations (that is, Trojan implantation) by an untrusted foundry, several split manufacturing techniques have been proposed [10–12]. These individual approaches are applicable to either conventional or emerging (2.5D/3D) IC technologies. Split manufacturing techniques rely on fabricating part of a design—usually the expensive transistor/active layer, and few lower metal layers, called front-end-of-line (FEOL) layers—in an untrusted foundry using advanced processes, and then completing the remaining less expensive fabrications steps—usually, upper level metal layers, called back-end-of-line (BEOL) layers—in a trusted foundry using a less advanced process technology [13]. Since it hides the connectivity information from an untrusted foundry, the approach has inherent benefits in terms of IP protection. However, methods that place the routing resources (such as, interconnects and vias) judiciously, such that security-critical nets are routed in upper level metal layers in trusted foundry, can maximize the obfuscation.

Whereas the split manufacturing concept is attractive for IP protection, attacks have been mounted on published approaches. These attacks utilize proximity data to recover the missing connectivity information. It is based on the assumption that EDA tools use certain parameters, for example, gate distance to minimize wire length [14]. Moreover, the biggest hurdle to split manufacturing is that the design house is still required to maintain a foundry to complete the BEOL, which could be prohibitively expensive depending on the split layer. Further, foundry compatibility and wafer alignment with such monolithic split-fabrication techniques may also hurt IC yield.

Camouflaging: Another class of obfuscation uses configurable cells in strategic locations of a design to disable normal operation, and to hide design intent [15]. These configurable cells can be programmed after fabrication to implement different logic functions. Since an untrusted design/fabrication facility would not know their function, they act as “camouflaged” cells. These cells often have a layout that makes them appear similar to other standard cells (such as NAND, NOR, or XOR) in a library. To an attacker in the foundry, and someone who performs destructive reverse engineering of an IC to extract the design, “camouflaged” cells do not reveal the design from its layout. This technique has been discussed in detail in Section 14.3.2.

2.5D/3D IC obfuscation: Recent advances in 2.5D/3D integration technology can facilitate split manufacturing. One can perform wire-lifting on a layout, so that the lifted wires are fabricated as a separate layer in a trusted facility. Next, the complete IC can be assembled by using through-silicon-via (TSV) bond points in a normal 3D IC design flow [11]. Every gate in the design in the FEOL layers is structurally akin to at least k other gates in the same design (as the BEOL information of the upper tier is missing). This makes it difficult for an attacker (for instance, in an untrusted foundry) to identify the gates, and thus to extract the complete design, or insert malicious changes.

2.5D IC technology could be leveraged further to securely partition a gate-level design, so that two or more partitions of the design can be fabricated at an untrusted foundry, and the interposer layer connecting these partitions can be fabricated at a trusted facility [16]. Unfortunately, split manufacturing based on 2.5D/3D IC technology suffers from the same drawbacks of requiring a separate fabrication facility. Further, these techniques usually demand significant amounts of gate-swapping and wire-rerouting operations for obfuscation, leading to large area and delay overheads.

14.2.4 INSTRUCTION SET OBFUSCATION

Every processor has an underlying instruction set architecture (ISA), which represents the type of commands and data, address space, and operation codes (opcodes) supported by it. The ISA serves as an intermediary between the software and hardware of the computer, and is usually public knowledge. Unfortunately, this also means that the well-known ISA and the vulnerabilities (example, buffer overflow) of a processor used in the system make it vulnerable to remote or even invasive attacks (for instance, via compromise of the memory unit holding the instructions). Identical ISA across million of processors used in IoT, and other systems, helps an attacker in software infection, software IP piracy, and malware propagation (from one computer to another through a network).

To combat the predictability of the ISA, a software code can be obfuscated, such that it runs in only one computer and remains locked in another. One such example is as follows: Each byte of a code can be scrambled using pseudorandom numbers, and during execution, it can be unscrambled to produce the original code [17]. This means that any unauthorized program, which was never scrambled, will be unscrambled to random bits, thereby preventing any targeted malicious behavior. Alternatively, the instructions could be XORed with a secret key as they are transmitted between the processor, and the main memory [18]. Such obfuscation of code either requires support from OS, which securely deobfuscates it before execution, or judicious modification in the processor's hardware that deobfuscates the code at runtime to ensure correct operation.

Instruction set obfuscation can also disrupt the disassembly phase of reverse engineering a machine code, that is, converting the machine code to human-readable form [19]. This can be accomplished by carefully inserting "junk bytes" in the instruction stream of the code. These junk bytes cause an automatic disassembler to either misinterpret the instructions, or the control flow of the program, but do not affect the program's functionality (semantics), as they are unreachable instructions during runtime.

14.2.5 PCB OBFUSCATION

PCB designs are vulnerable to similar security issues as IPs used in SoC. Effective obfuscation of a PCB design can prevent piracy, reverse engineering, or tampering during manufacturing and deployment. A possible solution for PCB design obfuscation is to insert permutation blocks on the board [20]. The permutation block, implemented with a complex programmable logic device (CPLD), or an FPGA, would take a set of critical interconnects (for example, data bus of microcontrollers) and permute them based on a key before they reach their destination. The permutation could be resolved to the correct configuration only when the correct key is applied to the CPLD, or the FPGA that realizes the permutation block.

14.3 HARDWARE OBFUSCATION METHODS

In this section, we discuss some of the well-researched obfuscation methods applied to gate and layout-level hardware designs in detail.

14.3.1 LOGIC LOCKING

Logic locking methods hide the true functionality and the structure of a hardware IP through the insertion of new gates in the combinational logic [6,21,22]. We refer to these gates as "key gates". As discussed earlier, these new gates effectively "lock" the design. Once these gates are inserted, the design is resynthesized and technology-mapped (that is, mapped to the standard cell library for the target technology node). This step optimizes the design in the presence of key gates, and propagates the structural transformation into the design beyond the key gates. In order to make the obfuscated design functional, the exact key must be applied to the obfuscated IP. Wrong key input into those gates would produce wrong output. Hence, an authorized user of the IP must obtain the key for properly using the obfuscated design.

Choice of the locations where key gates are inserted depend on the obfuscation objectives. Many heuristics for key-gate insertion have been studied. In its simplest form, key gates are inserted in random locations into the design [6]. This possibility facilitates the advantage of randomly distributing the key gates and the resultant structural changes across the design. Other heuristics include cone-based key-gate insertion, which tries to account for fan-in and/or fan-out cone of an internal node. A node that has large number of inputs in its logic cone has the advantage of making brute-force, or fault-analysis-based attacks more difficult, as described later in this chapter. On the other hand, a node that has large output cone is likely to cause more output corruptions for a wrong key, than the ones with smaller output cone.

Let us illustrate the logic locking method using an example. Figure 14.4A shows an original design in form of a gate-level netlist, which is obfuscated using three key-gates in Fig. 14.4B. The functional inputs of the original design are *A, B, C,* and *D*. Input lines *K1*, *K2*, and *K3* are key-inputs, which are connected to key-gates (XOR and XNOR gates). Upon the application of the correct key value ($K1 = 0$, $K2 = 1$, $K3 = 0$), the design will produce correct outputs. If not, it will produce wrong outputs.

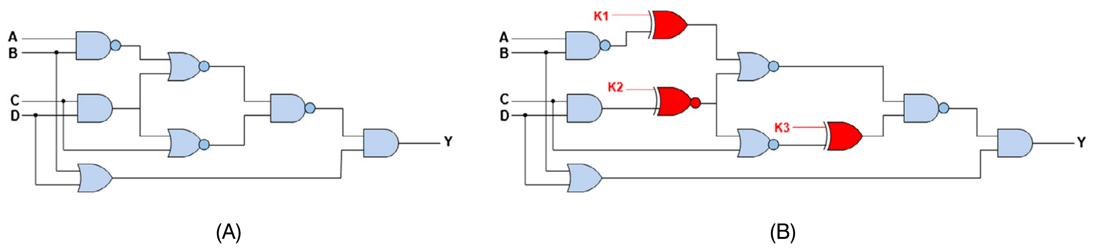


FIGURE 14.4

(A) Original circuit; (B) Obfuscated circuit with three key gates (red ones; dark gray in print version).

14.3.1.1 Security Properties and Metrics

A good obfuscation method needs to provide acceptable protection against IP piracy and reverse engineering. It, however, comes at additional design and test/verification cost. Most obfuscation methods incur considerable design overhead in terms of area, power or delay. It is important to enable a designer to perform quantitative analysis of an obfuscation method, so that security vs. design/test cost can be traded-off at the design stage. Next, we present relevant properties of obfuscation and the metrics that provide quantitative measures of security.

(i) **Correctness:** The logic locking techniques should maintain the original and correct functionality, that is, upon the application of the correct key, the design should produce the correct output(s). This property is inherited from conventional encryption techniques.

(ii) **Entropy and Hamming distance:** The technique should be resilient against attacks that guess the correct outputs by observing previous input-output combinations. To hinder these attacks, the entropy of the output response of a design, due to the application of wrong keys, should be increased. In other words, the Hamming distance between the output responses of the design under correct and wrong key should be ideally 50%. This Hamming distance represents the entropy of the output response under a faulty key input [21].

(iii) **Entanglement:** The key-gates of the obfuscated design should be irremovable. Since an attacker (for example, one in an untrusted foundry) is expected to have access to the internal nodes of an obfuscated design, he/she can simply remove the key-gates, and then retrieve the original design.

(iv) **Output corruptibility:** It is desirable to have higher output corruptibility to prevent both black-box usage, and to hide the design intent. First, it is important to ensure wrong outputs for large population of incorrect keys. Second, for each wrong key, it is important to increase the number of outputs that fail to produce correct function. Poor output corruptibility creates vulnerability against “bypass attacks”. For example, an obfuscated netlist, which creates wrong output for only one key pattern, can be easily fixed to produce original functionality by checking for the specific pattern, and forcing correct values (that is, “bypassing” the obfuscated logic) at the output for that pattern [23]. An obfuscated design with such “bypassing” can then be used as a black-box. Furthermore, a design with poor corruptibility can be more vulnerable to key sensitizing attack (KSA), or other attacks.

(v) **Resistance against key-guessing attacks:** An attacker could try to guess the correct key value from previously observed input-output pairs. To render the obfuscated design resistant to such key-guessing attacks, key-gates should be placed in such a way that the number of input-output pairs required to obtain the correct key value is increased exponentially with key size.

(vi) **Design overhead:** The logic locking techniques should aim to minimize the area, power, and delay overheads. The security properties listed above should be traded-off with the overheads to meet specific targets for an application.

14.3.1.2 Possible Attacks on Logic Locking

Multiple attacks have been reported against existing logic locking techniques. In these attacks, the main target of an attacker has been discovering the locking key. Note that a “brute-force attack” on obfuscation would be to try all possible key combinations, and observe the output values. The main aim of the attacker in this case becoming aware that the observed outputs match golden functional outputs is determined to be the correct key. In the worst case, an attacker needs to try 2^N possible key values for a key of size N bits. Similar to conventional encryption techniques, difficulty of discovering the key here is an exponential function of the key size. Hence, for a reasonably large key size, for example, 128-bit or larger, discovering the key using brute-force attack becomes computationally infeasible. Note that a wrong key can produce correct functional behavior for some patterns at the original inputs of a circuit. Hence, in order to guarantee that the correct key is found, it is important to verify correct functional behavior for all possible input combinations.

Key sensitizing attacks (KSA) [24]: The value of a key-bit can be obtained by using automatic test pattern generation (ATPG) tool, which comes from different CAD tool vendors (such as Synopsys, Cadence, Mentor), and are widely used by test engineers to generate efficient test patterns for a design. This attack assumes the availability of an unlocked chip or golden functional behavior of an IP. An attacker sensitizes a key-bit without masking or corruption by the other key-bits and/or outputs. By observing the output, it is possible to determine the value of the sensitized key-bits, provided that other key-bits have no interference in the sensitized path. Once an attacker determines an input pattern, which can sensitize a key-bit to an output without interference, he/she can then apply it to the functional IC. At this point, this pattern will be used to sensitize the correct value of the key-bit to an output. By observing the output, an attacker can resolve the key value.

Boolean satisfiability (SAT) attacks [24,25]: The SAT attack is a powerful attack for discovering the correct key from an obfuscated design by modeling it as a Boolean satisfiability problem, and using heuristic based algorithms to solve it. Similar to KSA, it also assumes the availability of the unlocked chip or golden functional outputs. SAT attack minimizes the key search space, and thus—compared to a brute-force attack on finding the key—it significantly reduces the effort required to find the correct key value. SAT attack rules out incorrect key values by using distinguishing input patterns (DIPs). An input value, for which at least two different key values produce different outputs, is called a DIP. In some cases, a single DIP can rule out more than one incorrect key value. The SAT attack randomly chooses the DIPs (or aligns with those dictated by the underlying SAT engine). With large number of incorrect keys being ruled out by a single DIP, very few patterns are needed to find out the correct key.

Let us consider the example circuit in Fig. 14.4B, which is an obfuscated version of the circuit in Fig. 14.4A. We will now discuss the SAT attack on this circuit. For this attack, an attacker needs the obfuscated netlist and a functional IC (or golden functional outputs). The obfuscated netlist can be obtained from different sources (for example, from an untrusted DFT-insertion facility or foundry), and a functional IC can be obtained from open market. During this attack, the first step is to find the input-output pairs from the functional IC. Then, a SAT solver tool randomly chooses the DIPs and attempt to rule out the wrong key values. Figure 14.5 shows the iterative process for ruling out invalid keys.

Figure 14.5 shows the functional outputs for different input patterns. It also shows the outputs for different combinations of input and key values for the circuit in Fig. 14.4. In Fig. 14.5, the eight possible key values for the three bit-keys are denoted as $key0, key1, ...key7$, where $key = \{K1,K2,K3\}$ and $key0 = \{0, 0, 0\}$, $key1 = \{0, 0, 1\}, ... key7 = \{1, 1, 1\}$, and so on. The outputs for different key values are marked in green (light gray in print version) and red (dark gray in print version) colors. Red and green denote wrong and correct outputs, respectively. In the first iteration, we assume that the DIP is chosen to be 1111. In this iteration, the SAT solver finds $key1$ as a wrong key combination, since the output for this key combination does not match with the functionally correct output. Hence, it removes $key1$ from the list of valid key values. In second iteration, 0011 is chosen as the DIP, and $key5$ is ruled out in the same way as in iteration 1. The SAT solver tool goes through multiple iterations to gradually rule out wrong key values. In this example, after 5 iterations, 5 key values ($key1$, $key3$, $key5$, $key6$, $key7$) are ruled out, and only 3 key combinations are left from which an attacker can easily find the

Inputs				Output	Output for Different Key Values								Ruled out key(s) in
A	**B**	**C**	**D**	**Y**	**key0(000)**	**key1(001)**	**key2(010)**	**key3(011)**	**key4(100)**	**key5(101)**	**key6(110)**	**key7(111)**	**each iteration (i)**
0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	1	1	1	1	1	1	1	1	1	1	
0	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	1	1	1	1	1	1	1	1	0	1	1	i=2 >> key5
0	1	0	0	1	1	1	1	1	1	1	0	1	
0	1	0	1	1	1	1	1	1	1	1	0	1	
0	1	1	0	1	1	1	1	1	1	1	1	0	i=3 >> key7
0	1	1	1	1	1	1	1	1	1	0	1	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	1	1	1	0	1	i=5 >> key6
1	0	1	0	0	0	0	0	0	0	0	0	0	
1	0	1	1	1	1	1	1	1	1	1	1	1	
1	1	0	0	0	1	1	0	1	1	1	1	1	
1	1	0	1	0	1	1	0	1	1	1	1	1	
1	1	1	0	1	1	1	1	0	1	1	1	1	i=4 >> key3
1	1	1	1	1	1	0	1	1	1	1	1	1	i=1 >> key1

FIGURE 14.5

Elimination of invalid key values under SAT attack to reduce the key search space.

correct key by trying them in the obfuscated netlist. However, if the SAT solver considers 1100 or 1101 as the DIP, all the incorrect key values will be ruled out in just one iteration.

In a practical scenario, SAT solver does not consider each of the key bits. It chooses a part of the key-bits; for instance, in the above example, 2 key-bits will be set to a known value, considering other ones as "don't care". At this point, if the SAT solver finds a DIP, it will just rule out those values of key-bits to be incorrect, and in this way, the correct key search space can be significantly minimized.

Countermeasures to attacks: In order to make the design resistant to key-guessing attacks, key-gates should be inserted in such a way that an attacker cannot propagate the output of a single key-gate [26], that is, the observed output should be a function of multiple key-gates. Key-gates can be inserted judiciously, so that they block each other's sensitization path. This method forms a "clique". With the increase in the size of the clique, the attacker's effort will increase as well.

To achieve resistance against SAT attacks, some cryptographic primitives or SAT-resistant logic circuits, called anti-SAT blocks, can be incorporated into the design. The idea is to prevent a SAT solver from efficiently minimizing the key space, that is, forcing it to go through exponentially large number of iterations. Anti-SAT block is a small circuit, which takes inputs from internal nodes of the original circuit along with some key inputs, and it exponentially increases the key search space. It corresponds to a high level of difficulty for a SAT solver to find the correct key values.

14.3.2 GATE CAMOUFLAGING-BASED OBFUSCATION

Gate camouflaging is an obfuscation technique, which is shown to provide effective countermeasure for RE attacks [15,27–29]. In this technique, a designer incorporates configurable camouflaged gates in selected locations of a design. This is similar in concept to logic locking, except configurable gates are inserted instead of key gates. The camouflaged CMOS logic cells can be configured to perform a variety of logic functions for the same layout. Figure 14.6A shows an example of a cell, where 19 contacts are used in the configurable CMOS cell [15]. If contacts 2, 4, 6, 8, 11, 12, 16, 17 are true, and the rest are dummy, the camouflaged cell operates as a NAND gate. Figure 14.6B shows an obfuscated circuit, where two logic gates have been replaced by configurable CMOS cells $C1$ and $C2$. Although an

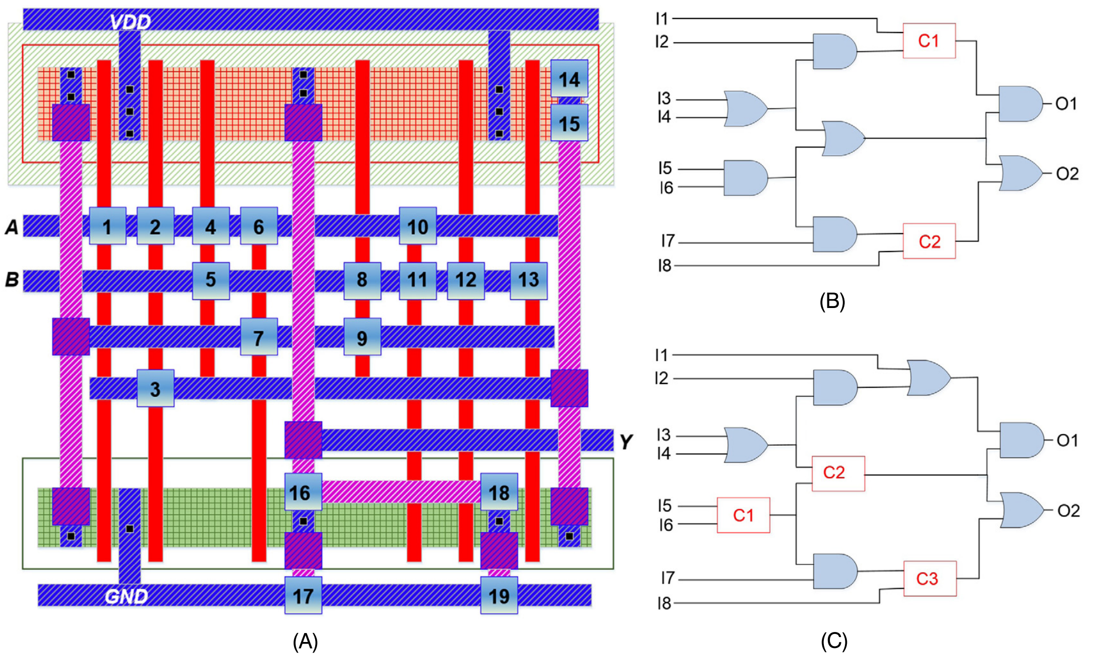


FIGURE 14.6

(A) A configurable CMOS cell with 19 contacts having the functionalities of a NAND, NOR, or XOR gate, where the configuration is made by programming the true and dummy contacts; (B) A circuit with two camouflaged gates $C1$ and $C2$; (C) Improved cell distribution to prevent a VLSI testing-based attack.

attacker may be able to successfully reverse engineer the layout of the circuit, it is difficult to determine the true functionality of the camouflaged logic gate. This, in turn, hides the functionality of the design, and makes the design resistant to both black-box usage and structural reverse engineering.

A possible approach to design configurable CMOS cells is to use true and dummy contacts [27, 28]. A true contact joins the dielectric between two nearby layers, which corresponds to an electrical connection. However, a dummy contact has a gap in between these connections (usually filled by an insulator such as SiO_2), and thus creates a fake connection. When an attacker tries to perform top-down image processing-based RE attack, he/she will not be able to detect whether a contact is true or dummy, since these are not identifiable through existing imaging technologies.

14.3.2.1 Attacks and Countermeasures

To know the exact functionality of the design shown in Fig. 14.6B, an attacker will have to evaluate $3^2 = 9$ possible combinations, since the cells $C1$ and $C2$ can be either XOR, NAND, or NOR gates. Furthermore, in the absence of scan-chain access, the attacker can only apply vectors to the circuit's primary inputs (PIs), and observe the corresponding output, to verify whether any guess is correct or wrong. This makes the attacker's job extremely challenging, as the logical effect of the camouflaged gate might not be directly observable at the primary output. In the example discussed above, we can observe the following:

(i) the output of a camouflaged gate, in response to the input “00”, can differentiate XOR from NAND and NOR. This is because input “00” for XOR produces an output of 0, whereas both NAND and NOR produce a 1 output;

(ii) the output in response to the input “01” or “10” can differentiate NAND and NOR, since NAND outputs 1, whereas NOR outputs 0.

Based on these observations, the attacker can apply a test vector “001XXXXX” (X represents don’t care values) at the PIs. This justifies the camouflaged gate $C1$’s inputs as “00”, and sensitizes $C1$’s output to primary output (PO) $O1$. When $O1$ is 1, $C1$ will either be NAND or NOR. This is known as a VLSI-testing-based attack. In such an attack, finding the entire truth table is not necessary, since a number of sensitizing input-output pairs is enough to reveal the functionality of a camouflaged cell.

To thwart VLSI testing-based attack, selective gate camouflaging can be applied. In this approach, logic gates that interfere with each other are camouflaged, so that they cannot be sensitized, and their output cannot be directly observed [15]. When one gate lies on the path between the other gate and an output, or the outputs of two gates merge into the same gate, interference will occur in between these two gates. Consider the example shown in Fig. 14.6C, where none of the camouflaged gates can be resolved. $C1$’s output cannot be observed from any of the POs without resolving $C2$ and $C3$. Then, $C3$’s inputs are not controllable unless $C1$ is resolved first. Furthermore, both the controllability of inputs and the observability of output of $C2$ depend on the functionality of $C1$. This will force the attacker to employ brute-force, that is, searching all possible functional combinations of these camouflaged gates [15]. For each possible combination, the attacker will have to simulate the design and compare the outputs with those of a correctly operational circuit.

Although enhanced IC camouflaging, based on gate interference, prevents VLSI-test based attacks, it does not ensure security against other possible attacks. For example, a SAT-based formulation may be used by an attacker to determine the true functionality of the camouflaged gates. In this attack, the SAT solver returns camouflaged gate assignments, which complies with the input-output patterns obtained from a functional IC.

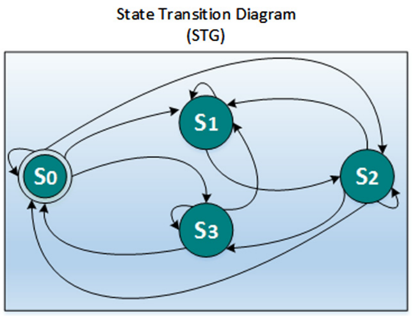
14.3.3 FINITE-STATE-MACHINE-BASED (FSM-BASED) HARDWARE OBFUSCATION

Logic locking approaches described earlier target obfuscation of combinational circuits. They do not directly modify the state machine of a sequential design. Whereas logic locking can indirectly impact the next state transition logic of a state machine, state space obfuscation techniques aim at directly modifying both the state machine and the associated combinational logic.

In digital sequential circuits, FSM is an abstract machine which is commonly used to implement the control logic. FSMs have finite number of states, and they transition from one state to another. The transitions between states are triggered by the input(s) to the FSM, and the current state. Often, the states of an FSM is used to drive output control signals. A simple example of an FSM is shown in Fig. 14.7, which shows its state transition diagram and the state table.

14.3.3.1 State Space Obfuscation

To enable state space obfuscation, a designer can modify the state-transition function of an FSM and internal circuit structure in such a way that the circuit operates in normal mode, only upon application of



State Table

Current State	Input	Next State
S0	00	S1
S0	01	S2
S0	10	S3
S0	11	S0
S1	00	S2
S1	01	S2
S1	10	S1
S1	11	S1
S2	00	S1
S2	01	S3
S2	10	S0
S2	11	S2
S3	00	S1
S3	01	S0
S3	10	S3
S3	11	S3

FIGURE 14.7

An example of FSM and its state-transition table showing all valid transitions.

a predefined enabling sequence of patterns at primary inputs, that is, the “key” [8]. The state-transition function of a sequential circuit could be modified by inserting an additional FSM called obfiscation FSM (OFSM). The inserted FSM could have all, or a subset of the primary inputs of the circuit as its inputs (including the clock and reset signals), and could have multiple outputs. At the start of operations, the OFSM would reset to its initial state, forcing the circuit to be in the obfuscated mode. Depending on the applied input sequence, the OFSM would go through a state-transition sequence, and only on receiving N specific input patterns in sequence would go to a state, which enables the circuit to operate in its normal mode. The initial state, and the states reached by the OFSM before a successful initialization, constitute the “pre-initialization state space” of the FSM. The states reached after the circuit enters its normal mode of operation constitute the “post-initialization state space”. Figure 14.8 shows the state diagram of such an FSM, with $P0 \Rightarrow P1 \Rightarrow P2$ being the correct initialization sequence. The input sequence $P0$ through $P2$ would be decided by the IP designer during obfuscation.

The OFSM could be used to control the mode of circuit operation. It could also modify the selected nodes in the design, using its outputs and a modification cell, for example, $M1$ through $M3$ in Fig. 14.8. The modified nodes are selected, such that they can greatly affect the behavior of the modified system if the correct key is not applied. Hence, if a user is unable to apply the correct input sequence (or key) to the FSM, the design will not reach the normal mode, and will traverse through invalid states in the obfuscated mode, producing incorrect functionality.

14.3.3.1.1 Efficiency Metric for State-Space Obfuscation

To quantify the quality of obfuscation, two different metrics can be used. First, the functional difference can be represented using the number of failing vectors for a large number of input patterns. The structural obfuscation efficiency can be represented by the structural difference between the original, and obfuscated design. One approach to measure structural difference is to derive % of failing verifi-

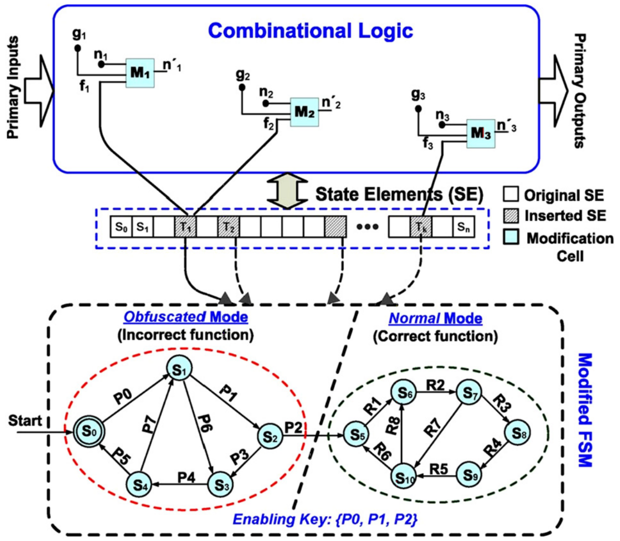


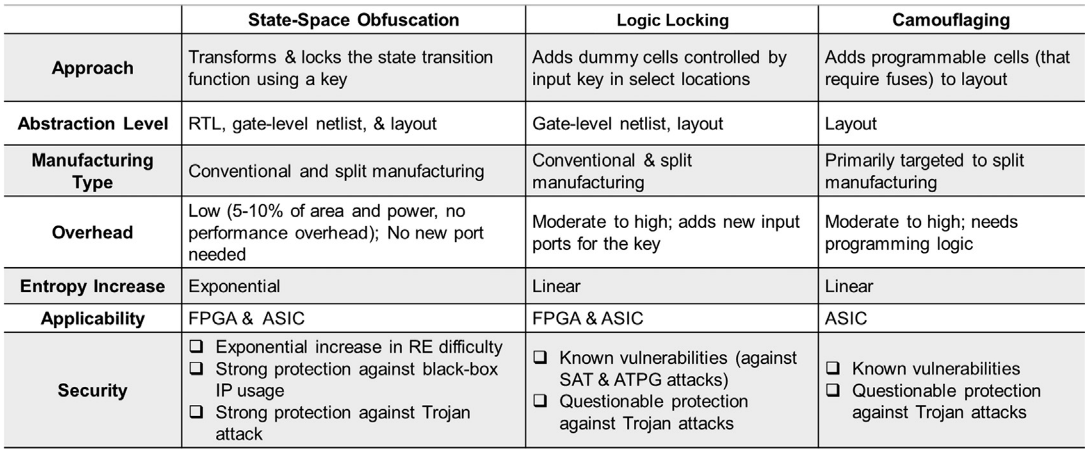
FIGURE 14.8

Functional and structural obfuscation of a state machine by modification of its state-transition function and internal node structures of the combinational part.

cation points when the original and obfuscated netlists are provided to a formal verification tool. The verification tool compares the two design based on a set of verification points, for example, flip-flop (FF) input or primary outputs. The obfuscation efficiency metric provides a way to assess the quality of state space obfuscation at design time, and allows the designer to maximize security under certain area, power, and delay constraints.

14.3.3.1.2 Comparison With Other Obfuscation Methods

While choosing the appropriate obfuscation method for a given IP, several factors need to be considered. The IP class (combinational/sequential), its representation within the design flow (RTL/gate-level), acceptable overhead, and required security benefits are some of the relevant parameters to consider. Figure 14.9 illustrates these features for three major classes of obfuscation. State-space obfuscation can be applied to almost all representations of digital IPs. Furthermore, they increase the reachable state space exponentially by adding few extra FFs, and hence provide an exponential increase in entropy. The area, power, and performance overhead of state-space obfuscation methods have also been shown to be very modest. Attacks on logic locking, for example, SAT attacks, which target key-recovery, are demonstrated on combinational circuits. Their effectiveness for a sequential design is not well-studied. To make SAT attack or KSA successful for sequential design, one needs to either: (1) unroll the sequential design into a combinational one, which involves exponential complexity in terms of number of possible states; or (2) obtain internal outputs of the combinational logic, which are input to the FFs. The later can be obtained by observing the values dumped to a scan-chain in response to an input pattern. However, that is often difficult to obtain due to the fact that many FFs may not



	State-Space Obfuscation	Logic Locking	Camouflaging
Approach	Transforms & locks the state transition function using a key	Adds dummy cells controlled by input key in select locations	Adds programmable cells (that require fuses) to layout
Abstraction Level	RTL, gate-level netlist, & layout	Gate-level netlist, layout	Layout
Manufacturing Type	Conventional and split manufacturing	Conventional & split manufacturing	Primarily targeted to split manufacturing
Overhead	Low (5-10% of area and power, no performance overhead); No new port needed	Moderate to high; adds new input ports for the key	Moderate to high; needs programming logic
Entropy Increase	Exponential	Linear	Linear
Applicability	FPGA & ASIC	FPGA & ASIC	ASIC
Security	❑ Exponential increase in RE difficulty ❑ Strong protection against black-box IP usage ❑ Strong protection against Trojan attack	❑ Known vulnerabilities (against SAT & ATPG attacks) ❑ Questionable protection against Trojan attacks	❑ Known vulnerabilities ❑ Questionable protection against Trojan attacks

FIGURE 14.9

Comparative analysis of sequential and combinational obfuscation.

be part of scan-chain, or the scan access can be prevented by a chip manufacturer after production testing.

On the other hand, logic locking has been shown to be vulnerable against several functional and structural analysis attacks. An effective obfuscation method is the one, which: (1) prevents deriving the key based on functional analysis (for instance, through SAT-based modeling), and (2) prevents structural signatures that make it vulnerable to remove the key gates or other structural changes incorporated to lock the design.

14.4 EMERGING OBFUSCATION APPROACHES

14.4.1 FPGA BITSTREAM OBFUSCATION

The FPGA configuration file, also called bitstream, is a valuable IP, which is susceptible to variety of attacks during system design and deployment. Attacks on bitstream include unauthorized reprogramming, reverse engineering, and piracy. Modern high-end FPGA devices support encrypted bitstreams, where vendor-specific FPGA synthesis tools produce an encrypted bitstream, which is decrypted inside an FPGA device before mapping the design. These FPGAs include on-chip decryption hardware. Encryption of bitstream provides some measure of security against major attacks, including piracy. However, it comes at the cost of on-chip decryption hardware, which requires additional hardware resources and adds to the configuration latency and energy. This is why low-end FPGA devices typically do not support encrypted bitstreams.

Security of the encrypted bitstream relies on the security of the encryption key. In the current business model, when FPGAs are used in a specific product (for example, a network router), all instances of the product uses the same encryption key. An original equipment manufacturer (OEM) often outsources

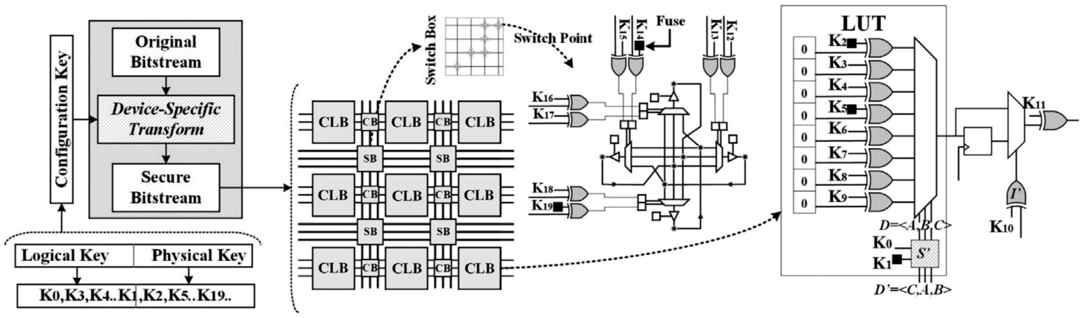


FIGURE 14.10

Overview of the bitstream obfuscation method using diversified FPGA architecture. From left to right: a two part (logical and physical) key is used to perform a device-specific obfuscation of the bitstream that is generated for a particular architecture. The obfuscated bitstream is mapped to appropriate FPGA device. Internal FPGA hardware resources are augmented with logic, which implements the inverse transform to make the bitstream functional. This ensures that bitstreams mapped to unauthorized devices will be nonfunctional. Because the logical key is time-varying, the architecture is changeable over time, and thus prevents known design attacks.

the FPGA programming step to third-party vendors, which need to have access to the encryption key. Besides, for remote upgrade, bitstreams are generally transmitted along with the decryption key. Both practices create significant vulnerability for key leakage, and hence compromises the security of an encrypted bitstream. Mathematically, encryption algorithms are known to be highly secure against brute-force attacks. However, in many cases, attackers can have physical access, and most encryption hardware are susceptible to side-channel attacks, for example, by key extraction through power profile signatures [30].

An adversary can convert an unencrypted bitstream to a netlist [31], thereby enabling IP piracy and malicious modifications, including Trojan insertion. The conversion step may not be necessary for Trojan insertion. Techniques such as unused resource utilization [32], which inserts Trojans in empty spaces in the configuration file, and mapping rule extraction [33], a type of known design attack, can be mounted on a bitstream for malicious modification. Moreover, if the hardware itself is cloned [34], a pirated bitstream could be used in a counterfeit hardware.

FPGA bitstream obfuscation provides a promising solution for bitstream protection against the aforementioned attacks. FPGA architecture could be modified using programmable elements, such that each device would become architecturally different from each other [35]. This changes the association between the FPGA device and bitstream, which is based on a configuration key. In this technique, each bitstream has its unique configuration key, resulting in unique bitstream for each device. A specific bitstream works for each physical FPGA device.

It provides protection against in-field bitstream reprogramming and IP piracy. This could be considered as a security-through-diversity approach to FPGA bitstream security. Furthermore, both physical (static) and logical (time–varying) configuration keys could be incorporated to ensure that attackers cannot use a priori knowledge about one device to mount an attack on another. Examples of architectural changes that allow the application of such configuration key is shown in Fig. 14.10, where the configuration storage (that is, SRAM cells) in the internal FPGA resources (look-up tables, switch boxes) are connected with XOR gates. Hence, even if the bitstream is stored onto these resources in obfuscated form, during operation when the correct configuration key is applied to the XOR gates, desired functionality is achieved.

The device-specific bitstream transformation can be done in the back-end of a vendor's tool flow, for example, after place & route, but before bitstream generation. Based on the configuration key of a particular device, the bitstream would be obfuscated, which would be node-locked onto a particular FPGA device. In addition to the node-locking of the configuration, bitstream obfuscation prevents an attacker from using the same bitstream in other devices, it also increases the difficulty of making an intelligent modification of the bitstream to compromise the system.

The bitstream obfuscation mechanism could also be incorporated in legacy FPGAs without any architectural modification [36]. FPGA dark silicon, that is, unused look-up-table resources (LUT resources), already available in these FPGAs could be used to enable obfuscation of the LUT contents. It helps to drastically reduce the overhead of the obfuscation method. The typical island-style FPGA architecture consists of an array of multi-input, single-output LUTs. Generally, LUTs of size n can be configured to implement any function of n variables, and require 2^n bits of storage for function responses. The nature of FPGA architecture requires that sufficient resources be available to accommodate for the worst-case mapping requirements. For example, some newer FPGAs may support 7 input functions, requiring 128 bits of storage for the LUT content. However, typical designs are more likely to use 5 or fewer inputs, while less frequently utilizing all 7. One can exploit the underutilized or unused LUT resources for the purpose of obfuscation. For example, an unused input of an LUT can be converted into a key input, where a specific value of the key input (0 or 1) would select the original function, whereas the other produces wrong output. For instance, consider a 3-input LUT, which contains 8 content bits, used to implement a 2-input function, $Z = f(X, Y)$. A third input K can be added, in such a way that the function becomes $Z' = f(K, Z)$, where $Z' = Z$, if the correct value of K is applied. Since designs with moderate complexity occupy thousands of LUTs, a large key would be used to obfuscate the whole IP. Thus, an attacker without the key would be unable to use the IP, or modify it intelligently. In this approach, the obfuscated design could be node-locked if a device-specific physical unclonable function generated key is used for obfuscation, making the bitstream functional only when mapped onto a specific FPGA device.

14.5 USE OF OBFUSCATION AGAINST TROJAN ATTACKS

Hardware obfuscation could be leveraged to protect against Trojan attacks. With respect to Trojan attacks, obfuscation can help in two ways: (1) it facilitates Trojan detection through functional, or side-channel, analysis by hiding the design intent, in particular, by hiding the rare events or attractive payloads from an intelligent adversary; and (2) by making the Trojan insertion difficult and possibly invalid in an obfuscated design.

Researchers have studied the role of state space obfuscation as a countermeasure against Trojan attacks [37]. The obfuscation scheme is based on modifying the state transition function of a given circuit by expanding its reachable state space, and enabling it to operate in two distinct modes: the normal mode and the obfuscated mode. Such a modification obfuscates the rareness of the internal circuit nodes, thus making it difficult for an adversary to insert hard-to-detect Trojans. It also decreases the

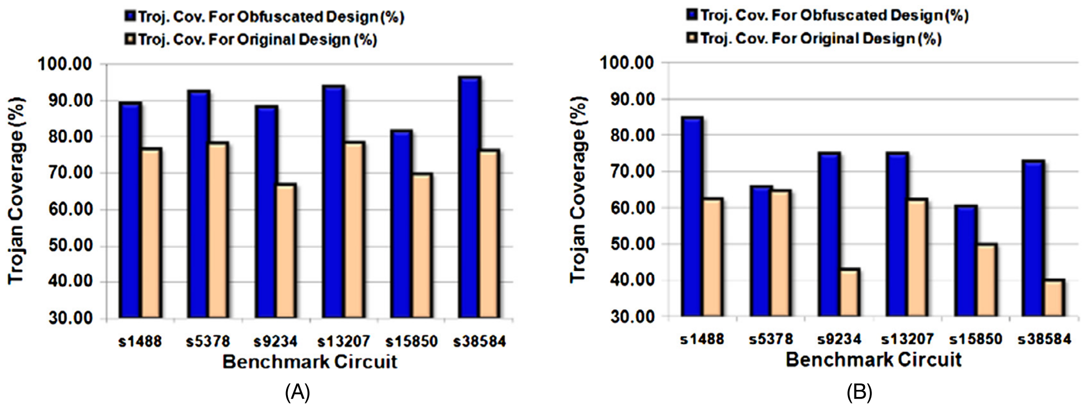


FIGURE 14.11

Improvement of Trojan coverage in several obfuscated ISCAS89 designs (using state space obfuscation) compared to the original design for (A) Trojans with two trigger nodes, and (B) Trojans with four trigger nodes.

potency of some inserted Trojans by making them activate only in the obfuscated mode. The combined effect leads to higher Trojan detectability and higher level of protection against such attack.

To find the rare trigger conditions, an adversary would require accurate estimations of internal node signal probability. One approach to accomplish that is through multiple random initialization of a given circuit to a reachable state, and then applying random input vectors. However, if the starting state in the adversary's simulations is in the obfuscated mode, because of the extreme rareness of the condition that allows the transition from obfuscated to normal mode, the simulations would most likely remain confined in the obfuscated mode. As a result, the signal probability for the circuit nodes calculated by the adversary would deviate significantly from those calculated if the simulations would have taken place with the states in the normal mode. Similar situation would prevent the adversary from finding the poorly observable nodes as potential payloads of a Trojan. Hence, if the adversary designs and inserts a Trojan based on wrong controllability/observability, there is a high probability that the Trojan would be triggered and detected with post-manufacturing logic testing. To increase this probability, the size of the obfuscation state space should be made as large as possible compared to the normal state space, by the addition of n extra state elements. State space obfuscation can provide a large improvement in Trojan coverage for random patterns, as shown in Fig. 14.11.

14.6 HANDS-ON EXPERIMENT: HARDWARE IP OBFUSCATION

14.6.1 OBJECTIVE

This experiment is designed to help the students explore the concepts of hardware obfuscation for IP protection. Using the HaHa platform, the students will learn how to apply various hardware obfuscation techniques to protect a design from unintended usage, for example, piracy and reverse engineering.

The experiment will also give the students the ability to perform attacks on an obfuscated design, with the goal of retrieving the functional behavior, or the structure of the original design.

14.6.2 METHOD

The first part of the experiment illustrates combinational obfuscation of a design, and the second part will focus on the sequential obfuscation. Students have to initially map an example design into the FPGA inside the HaHa platform. Next, they will apply a key-based logic locking mechanism, where the locked design will keep producing invalid outputs until the inserted key value is correct.

14.6.3 LEARNING OUTCOME

By performing the specific steps of the experiments, the students will learn how to apply a hardware obfuscation technique to any given design. They will also gain experience about the challenges with respect to balancing the security and design overheads (for example, area, power, or performance).

14.6.4 ADVANCED OPTIONS

Additional exploration on this topic can be done by applying more sophisticated attacks (such as SAT-based attack) to break the obfuscation and obtain the key, and improving the robustness of the obfuscation process.

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

14.7 EXERCISES

14.7.1 TRUE/FALSE QUESTIONS

1. Reverse engineering of hardware IP is considered illegal.
2. None of the software obfuscation approaches are applicable to hardware IPs.
3. The design house is untrusted in certain scenarios in an IC supply chain.
4. For maximum output entropy in logic locking, Hamming Distance should be 100%.
5. SAT attack directly finds the key used for obfuscation.
6. The features of design integrated with gate camouflaging can never be reverse engineered.
7. The entire truth table is not needed to attack a design obfuscated by camouflaging.
8. In state space obfuscation, modification cells are added to hide the newly inserted FSM.
9. An attacker cannot reverse engineer the FPGA netlist from the obfuscated bitstream.
10. FPGA-mapped designs are efficiently mapped, keeping almost no unused space in look-up-table memory cells.

14.7.2 SHORT-ANSWER TYPE QUESTIONS

1. In a semiconductor supply chain, what are the security and trust issues in an untrusted design house from the IP vendor's point of view?

2. List all possible security and trust issues in an IC supply chain when the design house sends the design for fabrication to an untrusted facility.
3. Why is encryption not a good enough solution for protecting hardware IP? What are the limitations of such an approach?
4. Briefly discuss SAT attack on logic locking. Also, describe the effect of a distinguishing input pattern (DIP) generated during the SAT attack.
5. What are the programmable resources within the FPGA architecture that could be modified to enable bitstream obfuscation?
6. During state space obfuscation, what would be the minimum clock cycle for applying a 256-bit key for an obfuscated IC with 18 primary inputs? Assume only 16 of the primary inputs can be used for applying the obfuscation key.
7. What is a possible way for the attacker to identify the inserted FSM that applies the obfuscation key in FSM-based obfuscation?
8. Discuss the differences between combinational and sequential obfuscation.

14.7.3 LONG-ANSWER TYPE QUESTIONS

1. Determine the correct key for the logic encrypted circuit below in Fig. 14.12 (in other words, what key makes x = x' and y = y'). Explain your answer.

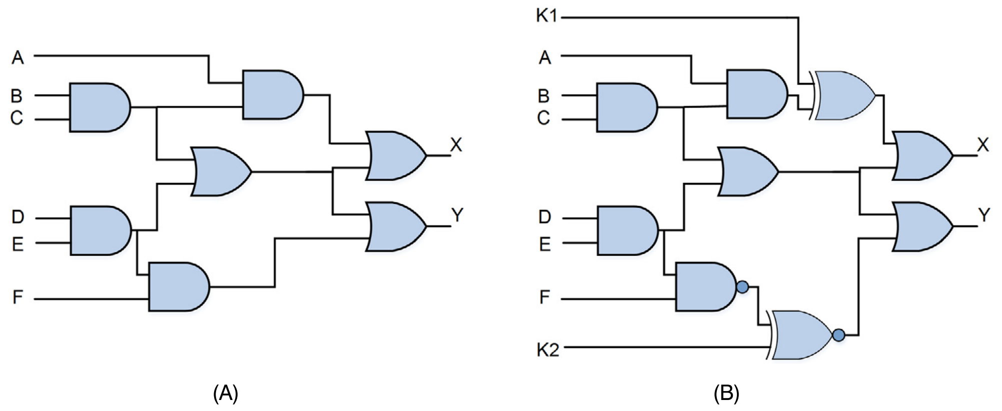


FIGURE 14.12

(A) Original design, (B) Obfuscated design.

2. Describe the vulnerabilities in each step of the supply chain of an IC.
3. Briefly describe the possible obfuscation-based solutions for countering hardware IP piracy at each step of the supply chain.
4. For state space obfuscation, describe possible attacks on an obfuscated design to reverse engineer it, considering both functional and structural attacks. Is it SAT attack resistant? Provide reasoning behind your answer.
5. Discuss how a certain hardware obfuscation method could make Trojan insertion difficult. Elaborate with respect to any of the well-known obfuscation methods.

6. Discuss different methods for RTL obfuscation.
7. Describe the two methods of node-locking FPGA bitstream, and compare their main differences.
8. Suppose a 2-input XOR function mapped to a 3-input LUT (Fig. 14.13A) is obfuscated with an additional key K (Fig. 14.13B). What could be the value of the bitstream that renders the XOR function for the correct key, K = 0? If a wrong key is applied, the function is inverted. Write the 8 bits in the following sequence: Bit_{111} - - - - - - - Bit_{000}
**Hint: Look at the obfuscation method described in [36].*

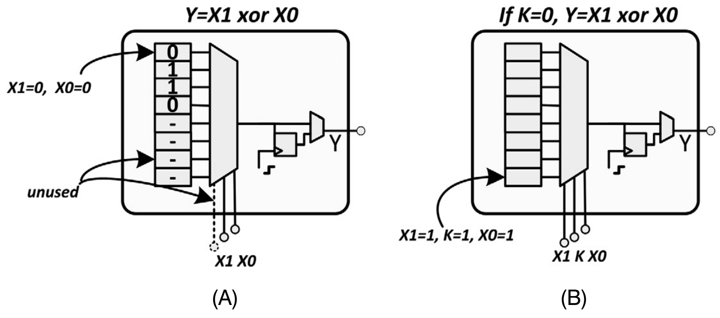


FIGURE 14.13

(A) Lookup table with 2-input function, (B) Obfuscated look-up table expanded to 3-input function.

REFERENCES

[1] C. Collberg, C. Thomborson, D. Low, A taxonomy of obfuscating transformations, Technical Report, Department of Computer Science, The University of Auckland, New Zealand, 1997.
[2] R.S. Chakraborty, S. Bhunia, RTL hardware IP protection using key-based control and data flow obfuscation, in: VLSI Design, 2010. VLSID'10. 23rd International Conference on, IEEE, pp. 405–410.
[3] A.R. Desai, M.S. Hsiao, C. Wang, L. Nazhandali, S. Hall, Interlocking obfuscation for anti-tamper hardware, in: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, ACM, p. 8.
[4] M. Brzozowski, V.N. Yarmolik, Obfuscation as intellectual rights protection in VHDL language, in: Computer Information Systems and Industrial Management Applications, 2007. CISIM'07. 6th International Conference on, IEEE, pp. 337–340.
[5] J.A. Roy, F. Koushanfar, I.L. Markov, Ending piracy of integrated circuits, Computer 43 (2010) 30–38.
[6] J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Logic encryption: a fault analysis perspective, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, pp. 953–958.
[7] N.A. Touba, E.J. McCluskey, Test point insertion based on path tracing, in: VLSI Test Symposium, 1996.
[8] R.S. Chakraborty, S. Bhunia, HARPOON: an obfuscation-based SoC design methodology for hardware protection, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28 (2009) 1493–1502.
[9] B. Yang, K. Wu, R. Karri, Scan based side channel attack on dedicated hardware implementations of data encryption standard, in: Test Conference, 2004. Proceedings. ITC 2004. International, IEEE, pp. 339–344.
[10] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, L. Pileggi, Efficient and secure intellectual property (IP) design with split fabrication, in: Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on, IEEE, pp. 13–18.
[11] 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.
[12] U. Rührmair, S. Devadas, F. Koushanfar, Security based on physical unclonability and disorder, in: Introduction to Hardware Security and Trust, Springer, 2012, pp. 65–102.

[13] K. Vaidyanathan, B.P. Das, E. Sumbul, R. Liu, L. Pileggi, Building trusted ICs using split fabrication, in: Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on, IEEE, pp. 1–6.
[14] J.J. Rajendran, O. Sinanoglu, R. Karri, Is split manufacturing secure?, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, pp. 1259–1264.
[15] 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.
[16] 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.
[17] E.G. Barrantes, D.H. Ackley, T.S. Palmer, D. Stefanovic, D.D. Zovi, Randomized instruction set emulation to disrupt binary code injection attacks, in: Proceedings of the 10th ACM Conference on Computer and Communications Security, ACM, pp. 281–289.
[18] G.S. Kc, A.D. Keromytis, V. Prevelakis, Countering code-injection attacks with instruction-set randomization, in: Proceedings of the 10th ACM Conference on Computer and Communications Security, ACM, pp. 272–280.
[19] C. Linn, S. Debray, Obfuscation of executable code to improve resistance to static disassembly, in: Proceedings of the 10th ACM Conference on Computer and Communications Security, ACM, pp. 290–299.
[20] Z. Guo, M. Tehranipoor, D. Forte, J. Di, Investigation of obfuscation-based anti-reverse engineering for printed circuit boards, in: Proceedings of the 52nd Annual Design Automation Conference, ACM, p. 114.
[21] J. Rajendran, H. Zhang, C. Zhang, G.S. Rose, Y. Pino, O. Sinanoglu, R. Karri, Fault analysis-based logic encryption, IEEE Transactions on Computers 64 (2015) 410–424.
[22] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, B. Rouzeyre, A novel hardware logic encryption technique for thwarting illegal overproduction and hardware Trojans, in: On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International, IEEE, pp. 49–54.
[23] Xiaolin Xu, Bicky Shakya, Mark M. Tehranipoor, Domenic Forte, Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks, in: International Conference on Cryptographic Hardware and Embedded Systems, Springer, 2017, pp. 189–210.
[24] P. Subramanyan, S. Ray, S. Malik, Evaluating the security of logic encryption algorithms, in: Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on, IEEE, pp. 137–143.
[25] M. El Massad, S. Garg, M.V. Tripunitara, Integrated circuit (IC) decamouflaging: reverse engineering camouflaged ICs within minutes, in: NDSS.
[26] J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Security analysis of logic obfuscation, in: Proceedings of the 49th Annual Design Automation Conference, ACM, pp. 83–89.
[27] L.-w. Chow, J.P. Baukus, C.M. William Jr., Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide, 2002, US Patent App. 09/768,904.
[28] L.W. Chow, J.P. Baukus, B.J. Wang, R.P. Cocchi, Camouflaging a standard cell based integrated circuit, 2012, US Patent 8,151,235.
[29] 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.
[30] A. Moradi, A. Barenghi, T. Kasper, C. Paar, On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs, in: Proceedings of the 18th ACM conference on Computer and Communications Security, ACM, pp. 111–124.
[31] J.-B. Note, É. Rannaud, From the bitstream to the netlist, in: FPGA, vol. 8, p. 264.
[32] R.S. Chakraborty, I. Saha, A. Palchaudhuri, G.K. Naik, Hardware Trojan insertion by direct modification of FPGA configuration bitstream, IEEE Design & Test 30 (2013) 45–54.
[33] P. Swierczynski, M. Fyrbiak, P. Koppe, C. Paar, FPGA Trojans through detecting and weakening of cryptographic primitives, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34 (2015) 1236–1249.
[34] K. Huang, J.M. Carulli, Y. Makris, Counterfeit electronics: a rising threat in the semiconductor manufacturing industry, in: Test Conference (ITC), 2013 IEEE International, IEEE, pp. 1–4.
[35] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, S. Bhunia, MUTARCH: architectural diversity for FPGA device and IP security, in: Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, IEEE, pp. 611–616.
[36] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, S. Bhunia, Robust bitstream protection in FPGA-based systems through low-overhead obfuscation, in: ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on, IEEE, pp. 1–8.
[37] R.S. Chakraborty, S. Bhunia, Security against hardware Trojan attacks using key-based design obfuscation, Journal of Electronic Testing 27 (2011) 767–785.