CHAPTER

HARDWARE IP PIRACY AND REVERSE ENGINEERING

7

CONTENTS

7.1 INTRODUCTION

Semiconductor industry is increasingly relying on a hardware IP based design flow, where reusable, pre-verified hardware modules are integrated to create a complex SoC design of intended functionality. Due to increasing design/verification cost and faster time-to-market demands, it has been difficult for a single manufacturing company to design, develop, and fabricate a complete SoC. Consequently, IP-based hardware design process has become a global trend, where IP vendors design, characterize, and verify hardware IP blocks of specific functionality. Most SoC design houses purchase these IP blocks from third-party IP vendors—often distributed across the globe—which are then integrated into an SoC. Such an approach can significantly reduce the design/verification cost (due to reuse of IPs), while drastically lowering the time to market (by alleviating the design/verification time of SoC building blocks).

Hardware IP blocks can be classified into three broad categories based on their use cases and types of signal they process: (1) Digital IPs, where an IP receives digital inputs and process them to produce digital outputs, for example, processor core, graphics processing units (GPUs), digital signal processing blocks, encryption/decryption block, and embedded memory; (2) Analog and mixed-signal IPs, where some or all of the inputs/outputs of an IP are analog signals, and information processing is done on analog or mixed (digital and analog) signals, such as analog-to-digital or digital-to-analog-converter, amplifier, and integrator; and (3) infrastructure IPs, which are nonfunctional IPs integrated into an SoC to facilitate various operations, such as test, debug, verification, and security. Due to evergrowing computing demands, modern SoCs tend to include many heterogeneous processing cores, for example, multiprocessor SoC (MPSoC), together with reconfigurable cores, in order to incorporate logic that is likely to change as standards and requirements evolve. These IP blocks are integrated with an interconnect fabric (for instance, a bus, or network-on-chip interconnect) to meet target functionality and performance of a system.

The pervasive practice of integrating hardware IPs into an SoC design, however, severely affects the security and trustworthiness of SoC computing platforms. Statistics show that the global market for third-party semiconductor IPs is growing at a steady rate over the years, and it is predicted that it will grow by about 10% between 2018–2022 [1]. Due to growing complexity of the IPs and the SoC integration process, SoC designers increasingly tend to treat these IPs as a black-box and rely on the IP vendors on the structural/functional integrity of these IPs. However, such design practices greatly increase the number of untrusted components in an SoC design and make the overall system security a pressing concern. Hardware IPs acquired from untrusted third-party vendors can have diverse security and integrity issues. An adversary inside an IP design house can deliberately insert a malicious implant or design modification to incorporate hidden/undesired functionality. Such additional functionalities can serve broadly two purposes for an adversary: (1) it can cause malfunction in the SoC that integrates the IP; and (2) it can facilitate information leakage through a hardware backdoor that enables unauthorized access or by directly leaking secret information (e.g. cryptographic key, or internal design details of an SoC) from inside a chip.

In addition to deliberate malicious changes in a design, IP vendors can also unintentionally incorporate design features, e.g., hidden test/debug interfaces that can create critical security loopholes. In 2012, a breakthrough study by a group of researchers in Cambridge revealed an undocumented hardware-level backdoor in a highly secure military-grade ProAsic3 FPGA device from MicroSemi (formerly Actel) [2]. Similarly, IPs can have uncharacterized parametric behavior (e.g. power/thermal), which can be exploited by an attacker to cause irrecoverable damage to an electronic system. In a recent report, researchers have demonstrated such an attack where a malicious update of a firmware destroys the processor it is controlling by affecting the power management system. It manifests a new attack mode for IPs, where firmware/software update can maliciously affect the power/performance/temperature profile of a chip to either damage a system or reveal secret information using an appropriate side-channel attack, e.g. a fault or timing attack [3].

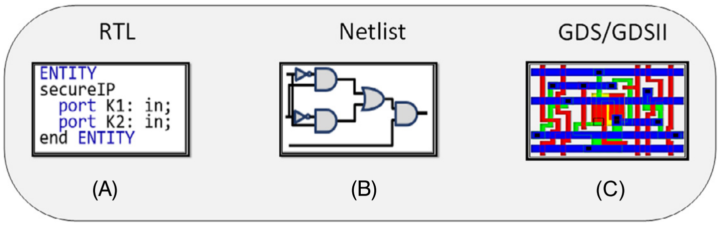
SoC designers require solutions to verify the integrity of an IP acquired from an untrusted vendor. On the other hand, the IPs themselves become vulnerable to piracy and reverse engineering attacks. Bad actors in a design house with access to an IP can steal and claim ownership of it. They can make pirated copies of the design, overproduce, and sell them illegally [4,5]. They can also reverse engineer an IP to understand the design intent and/or to make modifications to it for altering its functionality. The modified IPs can then be used in a SoC design or sold illegally without paying revenue to the original IP vendor. Hence, IP vendors need solutions to protect these IPs from piracy and RE attacks.

In this chapter, we describe possible attacks on hardware IPs in the life cycle of modern electronic products. We focus on two major classes of attacks: (1) tampering attacks resulting in IP trust issues, and (2) IP piracy and reverse engineering attacks. We consider both ASIC and FPGA design flows and describe the corresponding IP security issues.

7.2 HARDWARE INTELLECTUAL PROPERTY (IP)

An IP core is commonly defined as a reusable and modular unit of logic, cell, block, or IC layout designed and owned by an IP vendor. Whether targeted for ASIC or FPGA design flow, the IP blocks act as the basic building blocks of any hardware design. Due to the reusable and portable nature of IP cores, they play a major role in the current trend of the semiconductor industry towards globalization [6]. A single SoC typically contains IPs from multiple vendors. For example, the power management circuitry may come from an analog IP vendor in USA, whereas the cryptographic IP core might come from a separate vendor in Europe. These IPs can generally be classified as (i) soft IP, (ii) firm IP, and (iii) hard IP (illustrated in Fig. 7.1). A brief description of each class is provided below:

Soft IP: An IP developed in synthesizable register transfer level (RTL) format is known as a *soft IP*. RTL is basically a representation of the digital circuitry through flow of data between registers and logical operations on those signals carrying the data. Soft IPs are designed using hardware description languages, such as Verilog, SystemVerilog, or VHSIC hardware description language (VHDL), using control/data flow constructs supported by them. The application of HDL to create hardware IPs is analogous to the way software IPs are developed through computer programming languages, such



```
ENTITY
secureIP
   port K1: in;
   port K2: in;
end ENTITY
```

FIGURE 7.1

Different types of hardware IP cores. (A) Soft IPs, i.e. register transfer level (RTL) representation of hardware designs. (B) Firm IPs, i.e. gate-level netlists. (C) Hard IPs, layout of an IP usually represented as graphic database system (GDS/GDSII) files.

as C, C++, Java, and Python. Generally, chip designers have limited accessibility and capability of function-level modification if the soft IPs are obtained from third-party vendors.

Firm IP: An IP represented as a gate-level netlist is known as a *firm IP*. The netlist is basically a Boolean-algebra-based abstraction that shows how the logical functions of the IP are implemented through generic gates and standard cells. The firm IP cores also have high portability; they can be mapped to any process technology. Firm IPs are comparatively difficult to reverse engineer than soft IPs.

Hard IP: A hardware IP represented in a layout format, such as GDS (graphic database system), is known as a *hard IP* or a *hard macro*. These IPs are already mapped to a particular process technology. Hard IPs are the final form of IPs before the layout of the whole chip (e.g., an SoC) is created. Consequently, it is not possible to customize these IP cores for different process technologies by the manufacturers. However, due to the low-level representation, hard IPs are useful in the precise determination of area, timing, and performance profile of a chip. Analog and mixed-signal IPs often come in the form of hard IPs, as these are usually defined in the low-level physical description.

7.3 SECURITY ISSUES IN IP-BASED SoC DESIGN

An illustration of different attack types on Hardware IPs in terms of attackers and intent of the attacks is provided in Fig. 7.2 [7]. The following are the common security threats on hardware IPs.

Hardware Trojans: An adversary present in the design house or in the foundry can incorporate malicious circuitry in a design.

IP piracy and IC overbuilding: It is possible for an IP user or a rogue actor in untrusted foundry to pirate the IP and deliver it to unauthorized entities or market competitors. The foundry can produce pirated copies of the IC without the knowledge and permission of the parent company. The overproduced ICs can be sold in the black market at a cheaper price.

Reverse Engineering (RE): Reverse engineering refers to the process, where an adversary tries to reveal the functionality of the original design to reuse the IP illegally. The level of abstraction in RE may vary, based on the IP or IC, and the attacker's intent.

7.3.1 HARDWARE TROJAN ATTACKS

The functionalities of the Trojan include controlling, modifying, disabling, or snooping the contents of the design under attack [4,8,9]. Detecting a stealthy hardware Trojan can be extremely difficult in any hardware IP. The current practice of functional and formal testing methods fails to verify the circuits exhaustively due to scalability issues. Alternative solutions, such as reverse engineering and machine learning based methods, are either infeasible or ineffective in providing high confidence. A detailed discussion on hardware Trojan attacks is provided in Chapter 5 of the book.

7.3.1.1 Attack Model

Two different attack instances are considered in the hardware Trojan attack model [7]. Both attack scenarios are illustrated in Fig. 7.3. In the first scenario, an attacker in the foundry maliciously manipulates the lithographic masks of the IC to insert a Trojan. The insertion procedure might include

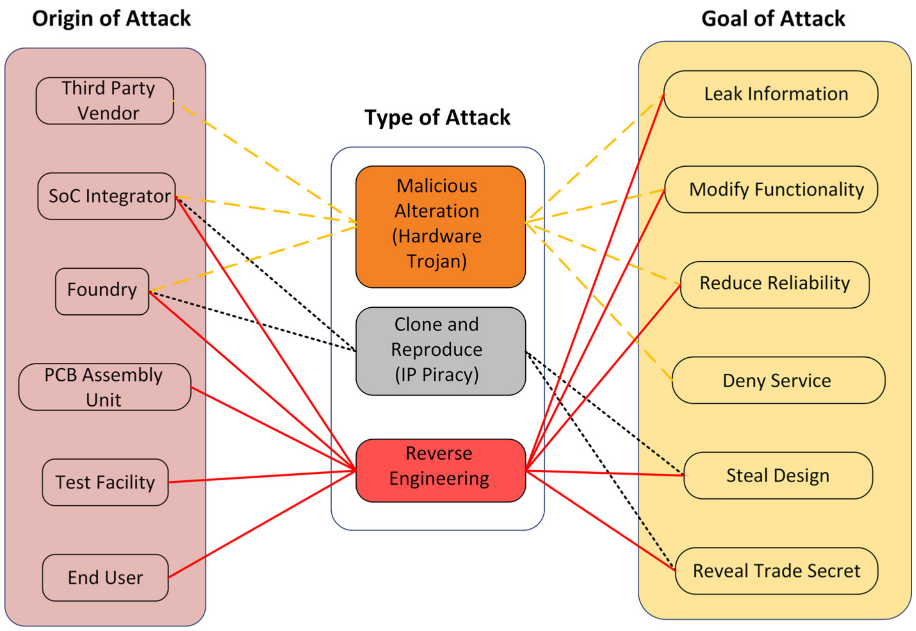


FIGURE 7.2

A classification of attacks on hardware IPs. It shows origin, type, and goal of each attack. The primary attack types are malicious alteration (i.e., hardware Trojan), cloning and reproduction, and reverse engineering.

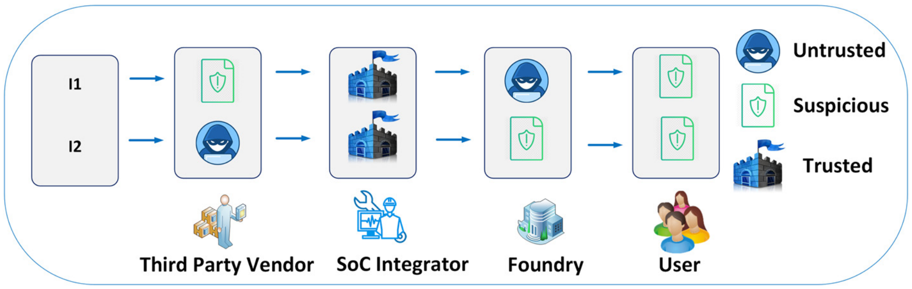


FIGURE 7.3

Two hardware Trojan attack scenarios: (I1) by an attacker in the foundry; (I2) by a rogue third-party vendor. Three types of entities are considered in every attack: an untrusted entity or the attacker, the trusted entity or the defender, and the suspicious entity, which can be an attacker or an accomplice to the attacker.

addition, deletion, or modification of functional gates from the original design [5,10]. The user and third-party vendor are considered suspicious and untrusted, respectively. In the second attack instance, the presence of a rogue entity is considered in a third-party design house, or an in-house chip design

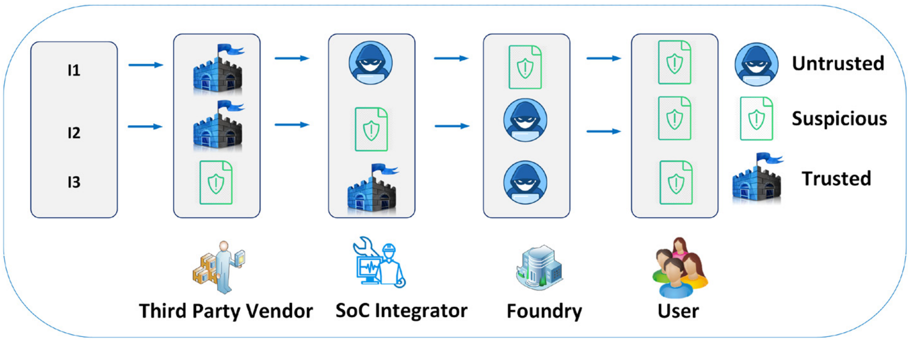


FIGURE 7.4

Three possible attack instances of IC/IP piracy and overproduction are considered here. IC/IP piracy concerns mostly arise from untrusted SoC design house and foundry, whereas the IC overproduction issues come solely from untrusted foundries.

team. It is highly unlikely for the verification team to detect such an inside attack, if they are not informed about the vulnerability beforehand [11,12]. All other entities are assumed untrusted in this attack instance.

7.3.2 IP PIRACY AND OVERPRODUCTION

An attacker with access to an IP (for example, a chip design house that buys an IP core from the IP vendor) can steal and claim ownership of the design. The attacker can make an illegal copy or "clone" of the IP. If the IC design house is the adversary, then it can sell it to another chip design house (after minor modification) claiming the IP to be its own [13]. Likewise, an untrusted fabrication house can make an illegal copy of the GDS-II database supplied by a chip design house and, then, illegally sell them as hard IP. An untrusted foundry can manufacture and sell counterfeit copies of the IC under a different brand-name [14].

7.3.2.1 Attack Model

Three different attack instances are illustrated in Fig. 7.4 [7]. Instance 1 shows that an attacker situated in IC integration house can pirate the 3PIP (third-party IP) and overproduce the IC illegally. The user and IC foundry is untrusted in this scenario, whereas 3PIP vendor is trusted. It is possible for the attacker to make pirated copies more than the licensed number allowed by the parent company. Instance 2 depicts how an attacker located in the foundry can extract the layout of a design and make pirated copies of the 3PIP. The trustworthiness of the SoC integrator and user is not guaranteed in this attack. The vendors are a trusted entity in this scenario. In the case of instance 3, the adversary is located in the foundry and is capable of pirating the IC design for illegal overproduction and subsequent selling to untrusted users. Third-party IP sellers are deemed suspicious in this instance. However, the SoC integrator is assumed to be trusted [15].

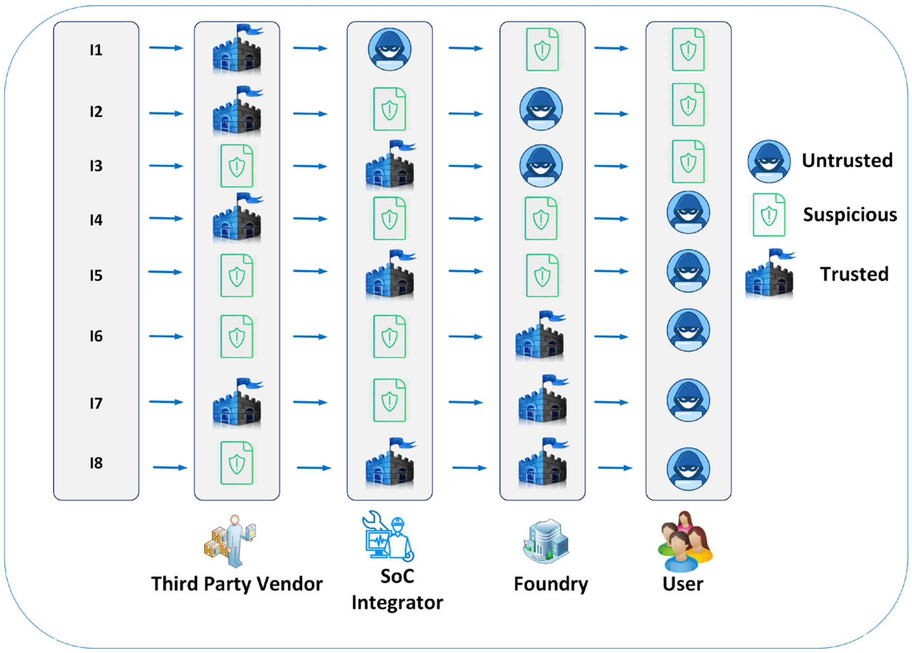


FIGURE 7.5

Reverse engineering attacks in IP. Eight (8) individual attack scenarios have been considered with varying degrees of threats originating from multiple sources. Third-party vendors, SoC integrators, chip foundry, and end-users are deemed to be the key entities of the attack instances.

7.3.3 REVERSE ENGINEERING

Reverse engineering is a complex process involving steps, such as attempts to infer the functionality of the design, extraction of the gate-level netlist, and identification of the device technology [16]. Several techniques and tools have been analyzed by researchers for reverse engineering IPs and ICs [17]. Reverse engineering knowledge can be illegally employed to steal or pirate a design, identify device technology, and illegally fabricate target IC. The primary objective of reverse engineering is to successfully procure an expected level of abstraction of the design. Once the abstraction level is reached, an adversary can exploit the primary input/output to figure out the functionality of the design. An adversary can also employ the knowledge obtained via reverse engineering to extract a gate level netlist of a competitor's IP. Thus, it is feasible for a malicious entity to misuse stolen IP as their own invention, sell it, or fabricate illegal ICs [16]. The target level of abstraction that any adversary wants to acquire depends on the purpose for the reverse engineering.

Figure 7.5 illustrates several attack instances of reverse engineering [7]. Instance 1 illustrates the possibility of an attacker reverse engineering the third-party IP in the SoC integration house. In this case, the foundry and users are assumed untrustworthy. The third-party vendor, however, is assumed to be trustworthy. A solution to this security issue is obfuscating the design (as described in Chapter 14) to prevent the access of the attacker to the original or golden design in the design house.

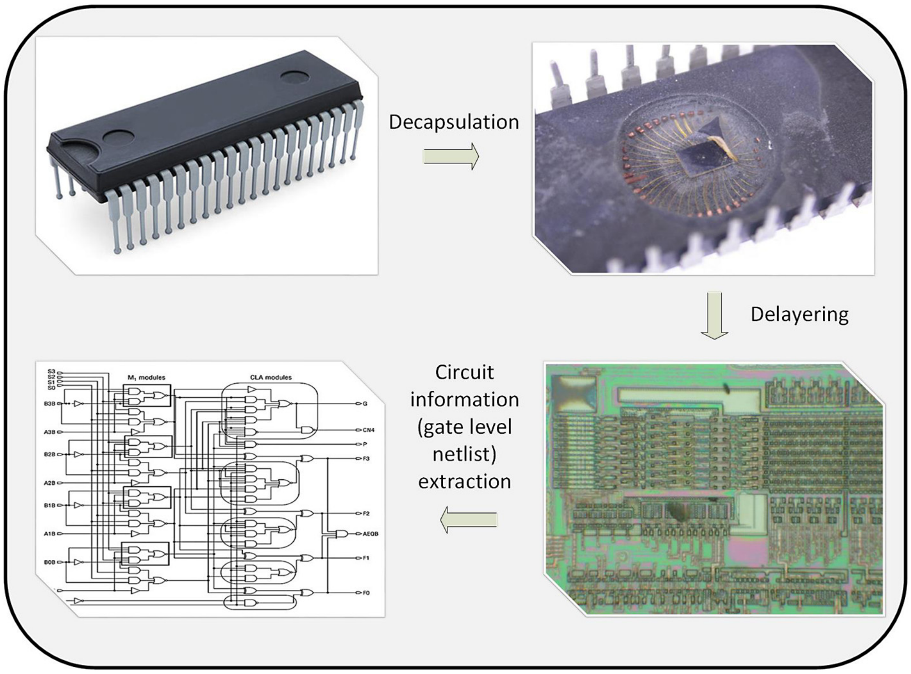


FIGURE 7.6

The basic steps of IC reverse engineering include removing the die from package, decapsulation of the IC, multiple stages of delayering, followed by scanning electron microscope (SEM) imaging of each layer, and finally, the gate-level netlist extraction to retrieve design functionality.

Instance 2 portrays an attack scenario, where the adversary can retrieve the third-party IP from the layout of an IC in the foundry. Similar to instance 1, the 3PIP vendor is assumed trusted, but the SoC integrator and user are assumed to be suspicious. Delivering an obfuscated design to the untrusted SoC integrator and the associated foundry would be an appropriate solution against such attack.

Instance 3 depicts an attack scenario, where an adversary launches a reverse engineering attack on the IC in the foundry. The attacker can retrieve the transistor-level layout from the reverse engineered design and, eventually, obtain the gate-level netlist. In this scenario, the SoC integrator is trusted, but the 3PIP provider and user are untrusted. Obfuscation of the target design is an effective solution in this attack scenario [17–19]. In instances 4 to 8, the attack scenarios depict user as someone, who does the reverse engineering. The reverse engineering process for an IC typically includes steps, such as, depackaging the IC, delayering it, obtaining layer images, and extracting the design netlist. Whereas design obfuscation by the third-party vendor is a solution applicable to scenario 4, scenario 5 requires an obfuscation of the layout by the SoC integrator. In case of attack instances 6 to 8, camouflaging the design by a trusted foundry would prevent the security breaches. Camouflaging provides an additional layer of security beyond design obfuscation.

Several approaches have been studied in the literature to explore the vulnerabilities of reverse engineering and develop countermeasures. Researchers have proposed algorithms for extracting gate-level netlists from the layout [19]. It has been shown that an exploitation of structural isomorphism can help to reveal the functionality of data path modules [18]. Attacks based on the behavioral matching of unknown units against known library components, such as adders, counters, and registers are also investigated [20]. In some cases, Boolean satisfiability theorem is applied to reveal the functionality of unknown modules through comparison with known library modules [21].

7.3.3.1 An Illustrative Example of IC Reverse Engineering

Figure 7.6 illustrates the key steps of reverse engineering an IC. The first step is removing the die from the package without causing any damage to its physical structure and functionality. Once the die is removed, it is cleaned and planarized for scanning electron microscope (SEM) imaging. The process of SEM imaging is performed in an iterative manner through decapsulation and multiple stages of delayering. SEM images of isolated regions are taken at first, and the group of images are stitched together later for extracting design information after each of the delayering steps. The final goal of the process is to extract the gate-level netlist (a list of connections among the components of the design) from the images and retrieve the circuit functionality from the netlist. Commercially available CAD tools can be used to obtain the circuit functionality from gate-level netlists.

7.4 SECURITY ISSUES IN FPGA

FPGAs are *reprogrammable* devices that have long been used as a prototyping platform for hardware designs. Over time, FPGAs have found applications in various domains, including automotive, networking, defense, and consumer appliances. Designs mapped onto the FPGA could potentially perform better in terms of power consumption and execution speed compared to software implementation in general-purpose processors. When a given task is implemented in a processor, it has to be executed within the architectural restrictions inherent to its general-purpose design. However, for FPGAs, the hardware design itself can be transformed appropriately and optimized for the target application (for example, data encryption or digital signal processing) through re-wiring configurable hardware resources. Hence, FPGAs typically provide higher performance and energy-efficiency in many applications than their processor counterparts. The combination of improved performance and reconfigurability opens the door for applications that require both, such as artificial intelligence and signal processing. FPGAs are being increasingly used in many security-critical systems. Therefore, designs mapped onto FPGAs have become an attractive target for adversaries who try to compromise a system. Furthermore, designs mapped to FPGAs are often considered as valuable IPs, which are vulnerable to theft and piracy. In this section, we describe these security issues in detail. The following subsections contain a brief description of the internals of FPGAs, design mapping process, and production lifecycle of FPGA-based systems. We also discuss the vulnerable points within this development cycle, and describe the attack models in detail.

7.4.1 FPGA PRELIMINARIES

To make a hardware IP or design functional, FPGAs use various reconfigurable resources. The design (that is, the RTL code or gate-level netlist) has to be transformed to a specific format that can use those resources inside the FPGA. If programmed correctly, the configured FPGA hardware provides the same functionality as the intended design. Most importantly, this design could be updated at any time by reprogramming the FPGA with a different configuration file. The process of generating the configuration file (also known as the bitstream) from the actual hardware design is shown in Fig. 7.7A.

FPGAs could be viewed as an array of programmable modules, where each module can efficiently serve a different purpose. The design and definition of internal resources vary across different FPGA vendors, such as Xilinx, Altera (currently Intel), and MicroSemi. However, in this section, we follow the naming conventions used by one of the major FPGA vendors, Xilinx. In Xilinx FPGAs, some of the common programmable modules are lookup tables (LUTs), Configurable Logic Blocks (CLBs), Connection Boxes (CBs), Switch Boxes (SBs), Block RAM (BRAM), Digital Signal Processing (DSP) blocks, and Input-Output Blocks (IOBs). Figure 7.7B shows a simplified architecture of the programmable fabric of an FPGA containing these diverse resources.

During the bitstream generation process, the design is first broken down to small segments of Boolean functions, each suitable for specific programmable hardware component inside the FPGA. This is referred to as *synthesis*, and the output is an FPGA-mapped netlist. Figure 7.8 shows a Full Adder design represented in gate-level format and its corresponding FPGA-synthesized version. Several gates are merged to one LUT that contains the Boolean function as the configuration bit (shown in hexadecimal). This netlist is then placed and routed, which defines the configuration of the connection boxes and switch boxes. Finally, the configuration bits are concatenated into a single file, called the *bitstream*, which is used to configure the FPGA. This process of bitstream generation and programming is done using a software tool provided by the FPGA vendor, such as the Vivado Design Suite for

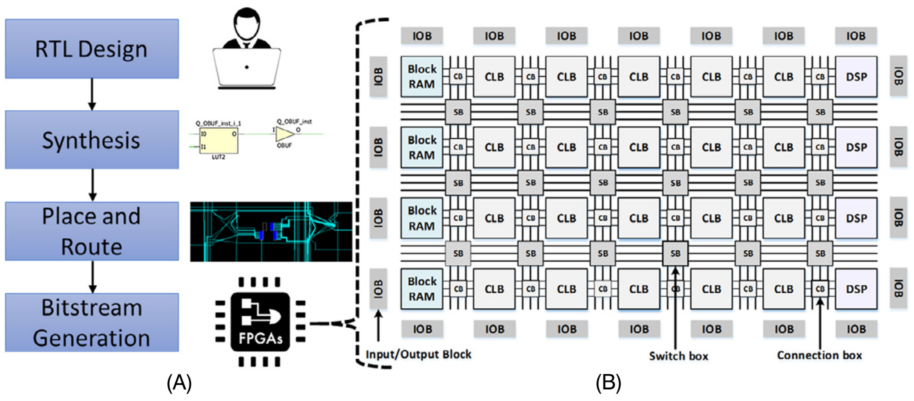


FIGURE 7.7

(A) FPGA bitstream generation flow, where an RTL design is converted to a stream of configuration bits that is used to program a FPGA. (B) Simplified architecture of an FPGA fabric containing CLBs, Block RAMs, DSP blocks, routing resources, and IO Blocks. Whereas the idea of having programmable resources is common across the FPGAs of all vendors, the available resources and their organization differ.

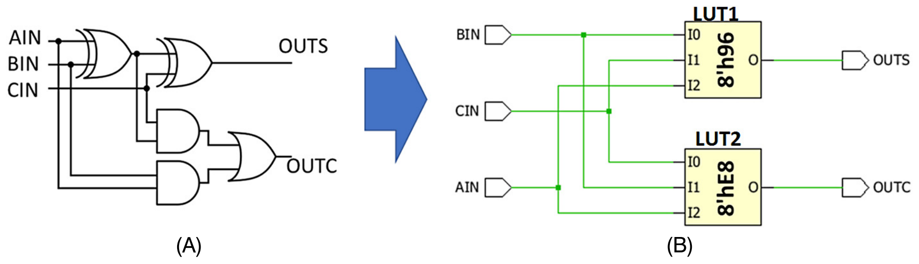


FIGURE 7.8

Example of FPGA synthesis: (A) A gate-level design of full-adder, (B) FPGA mapped netlist consisting of LUTs with 8-bit configuration contents $8'h86$ and $8'hE8$ after synthesizing the design using a vendor tool.

Xilinx FPGAs. More details regarding the fundamentals of FPGA architecture and its application are available in [22].

7.4.2 LIFECYCLE OF FPGA-BASED SYSTEM

An FPGA bitstream is vulnerable to different threats throughout the development and operational lifecycle of the system. Therefore, the vulnerabilities should be addressed at appropriate stages of its development. Depending on the application, the entities involved in each step and the corresponding lifecycle could differ. We discuss various possible entities, followed by an example lifecycle for FPGA that illustrates major vulnerabilities.

7.4.2.1 Entities

We consider the possible individuals, manufacturers, and hardware/software vendors that could directly or indirectly affect the security of the FPGA-mapped IP. These entities are briefly introduced below:

FPGA Vendor: Vendors offer the FPGA devices or FPGA-based solutions to the end users or developers, who integrate FPGAs into their products. Altera and Xilinx are the lead vendors in the programmable logic market. In 2014, Xilinx held almost 45%–50% of the market share, whereas Altera accounted for 40%–45% [23]. Just like the IC design houses, most of the FPGA vendors are fabless, and they depend on off-shore foundries and other third-party manufacturers.

Off-Shore Foundry: FPGAs consist of a base array integrated with different peripherals, and other components. The design and manufacturing process of the base array is similar to the ones for standard IC. As shown in Fig. 7.9, an FPGA vendor sends the layout of the base array in the form of GDSII (mask files) for fabrication.

Off-Shore Facility: The fabricated base array is forwarded to another facility for packaging, and assembly of the FPGA device. This facility could also be off-shore to reduce the manufacturing cost [24].

FPGA-based System Developer: FPGAs are widely integrated into systems, including automotive, defense, network processing, and consumer electronics. The companies developing such products buy FPGAs directly from a vendor, or through a third-party distributor as standalone FPGA ICs or ICs mounted on PCBs. They also buy or develop soft IPs, firmware, software, and various hardware com-

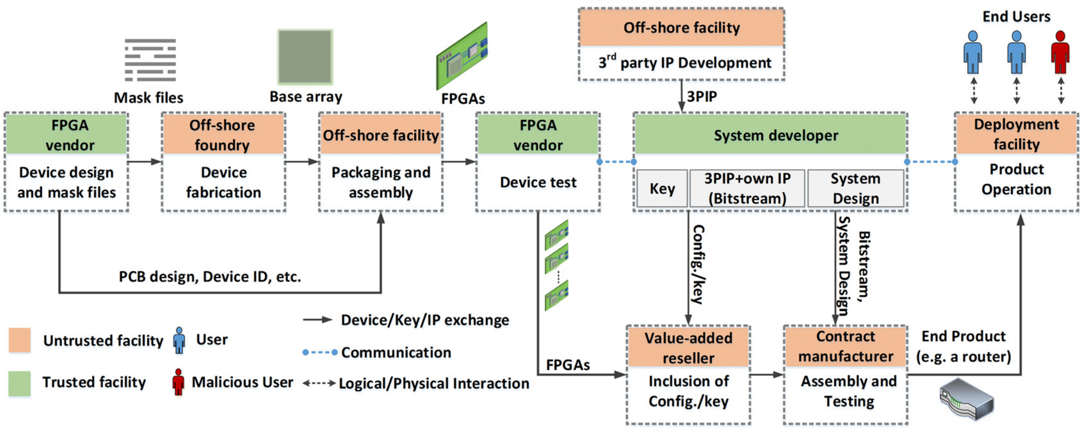


FIGURE 7.9

Figure illustrates the development lifecycle of an FPGA-based system. The base array of the FPGA is typically fabricated in an off-shore foundry. If the FPGA is sold as a board-level solution, the subsequent assembly process is performed at another third-party facility. These FPGAs (standalone ICs or boards) are bought through distributors assigned by the actual FPGA vendor. Based on the system developer's requirement, a contract manufacturer assembles a complete FPGA-based product.

ponents as needed. The integration of all the hardware and software components are often done through third parties. The developers often embed security features within the system to prevent cloning, reverse engineering, and tampering. When designing future devices, the FPGA vendors often discuss with their major consumers what security features they would like to see in future FPGA hardware.

Value-Added Reseller: A value-added reseller (VAR) usually programs features or configurations (modes of operation) to an existing product on behalf of the system developer. Though the presence of a VAR—in addition to the primary manufacturer—may appear redundant, it could become a necessity for the integration of confidential features that could not be shared with the primary manufacturer. For instance, storing cryptographic keys in an FPGA [25] may require a VAR, since providing both the decryption key and the encrypted bitstream to the same third-party could facilitate IP theft. Besides, FPGA vendors may only certify certain third-parties to sell or distribute devices on their behalf [26]. Therefore, a VAR could be present in many supply chains.

Contract Manufacturer: Development and deployment of the complete product may require assembling, repairing, and testing of large volumes of PCBs, and shipping them to the product buyer. Therefore, a system developer could hire one or more capable third-party manufacturers. The system design and components are sent to the contract manufacturer (CM) for production and testing. The CM may also buy the required components (hardware or software) on behalf of the system designer.

End User: Once purchased, the system is deployed by the owner to serve the customer. The owner could be a government entity, private corporation, or an individual buying the product. In addition to an authorized customer, the product could interact with someone having illegal access. Both the owner and the users are considered end-users having a certain level of knowledge of the system, privilege, and physical access. While most end-users would interact with the system only to receive the intended service, certain users may have malicious intent, such as compromising the system or stealing critical information.

7.4.2.2 Lifecycle

Figure 7.9 provides an overview of the development lifecycle of an FPGA-based product. The flow could be realized differently with another set of entities, based on the application, end-product, or the company developing the system. However, we attempted to construct a flow that covers all possible vulnerabilities relevant to the hardware design implemented on the FPGA in form of a bitstream.

Since various forms of bitstream vulnerabilities could stem from the underlying FPGA hardware itself, we begin with the FPGA device production phase initiated by the FPGA vendor. The vendor defines the architecture of the base array, consisting of various programmable units (that is, CLB, CB, BRAM) that performs as the reconfigurable platform. The vendor develops the base-array layout to generate the corresponding masks for fabrication. Since most of the design houses (and the FPGA vendors) are fabless, the masks are sent to a third-party fabrication facility. This leads to vulnerabilities, including malicious modification of the base-array layout, and various forms of IP theft issues. Furthermore, due to its known regular structure, insertion of malicious logic into the base-array design could be easier [24]. For instance, an attacker could insert a Trojan logic within the base array that observes the configuration bits of a switch box inside FPGA, and triggers only when a certain bit pattern is found. Once triggered, the malicious logic can modify the configuration of other resources, leading to a denial-of-service attack, or leakage of secret information. A list of possible triggers and payloads is presented in [24].

As mentioned earlier, an FPGA could either be purchased as standalone IC, or as a board-level solution. For developing board-level solutions, a board-level design containing the base array is specified by the vendor. Similar to the base-array fabrication, the board-level assembly process could also be exported to the same, or a different third-party, where the fabricated base array is placed on a PCB and, then, assembled with other peripheral components according to the design. It is possible that an adversary in the untrusted facility could make a malicious modification to the board-level design that either leaks the bitstream, causes logical (design functionality), or physical (device functionality) malfunction. Though the FPGAs are tested by the vendor, insertion of board and chip-level hardware Trojans, which circumvent the testing procedure is a viable threat since only a small segment of the test cases can be covered [24,27].

System developers who require FPGAs for building their products usually buy them through third-party distributors, who are certified by the FPGA vendor. If a contract manufacturer is involved in the development process, the devices are sent to the CM facility, where FPGAs are assembled with other hardware and software components. Various forms of attacks on bitstream such as cloning, reverse engineering, and tampering could take place at this stage. However, as mentioned earlier, the system developer would introduce security features to protect against such attacks.

7.4.2.3 Attacks on FPGA Bitstream

A wide range of attacks on FPGA bitstream have been reported over time. Major classes of attacks on bitstream are shown in Fig. 7.10. The proliferation of FPGA devices in mission-critical applications has involved resourceful entities, such as nation states and funded organizations in the list of possible adversaries interested to break opponent's system. Furthermore, the distributed lifecycle of the FPGA-based system is involving more and more untrusted entities that present new adversaries. Below, we

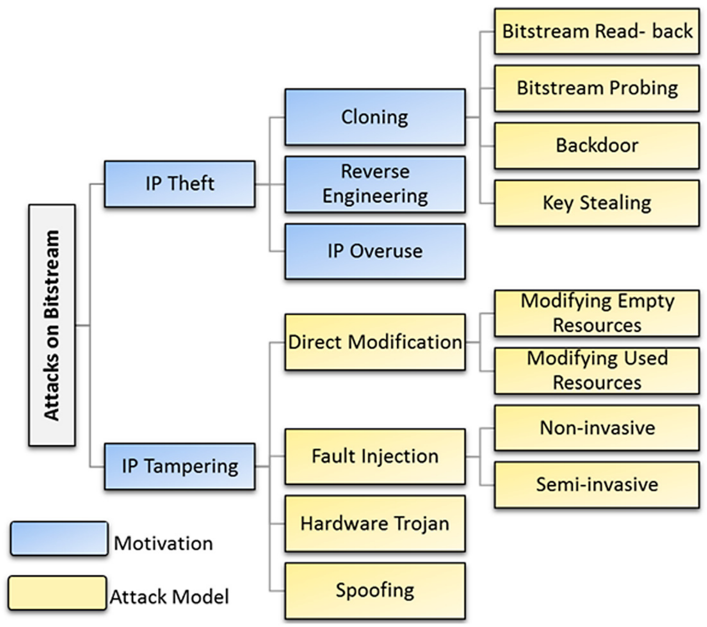


FIGURE 7.10

Various forms of attacks on FPGA bitstream with diverse motivation.

present various malicious motives to attack FPGA bitstream, along with corresponding attack models to accomplish them.

1. IP Theft: A design mapped onto the FPGA usually requires significant time and effort to develop, which makes the configuration bitstream of the design a valuable IP. Theft of an IP includes cloning, i.e., illegally using, or distributing the bitstream. The theft could also take place in the form of reverse engineering, where the design and functionality are extracted by analyzing the bitstream.

1(a) Cloning: The very nature of the FPGA device makes it vulnerable to cloning, since the same bitstream can be used in similar devices if found unencrypted, or even encrypted (if the encryption key is available). Throughout the development and deployment lifecycle of an FPGA-based system, the underlying bitstream could be prone to cloning attacks in a number of ways discussed below.

- *Bitstream Readback*
  Recall from Chapter 4, that JTAG is a common standard for in-circuit test. It is also used as a programming interface in majority of FPGAs. Programming and testing operations are initiated by sending different commands into the interface. Commands even exist for the retrieval of the configuration bits from the FPGA for bitstream integrity verification [28]. Therefore, unless deactivated, this could facilitate easy access to the unencrypted version of the bitstream.
- *Bitstream Probing*
  The volatile nature of SRAM FPGAs requires the bitstream to be loaded onto the reprogrammable fabric via the programming channel (for example, JTAG) from an external memory (i.e. flash) when

the system powers up. Therefore, intercepting this bitstream transfer using an electrical probe is one possible attack vector that facilitates cloning [29,30]. This attack is not applicable to nonvolatile (such as flash-based) FPGAs, since the configuration bits are always stored inside the reconfigurable fabric. Therefore, only invasive attacks on the non-volatile reconfigurable fabric are possible. Since physical access to a device with bitstream is required for mounting a probing attack, either an adversary at the contract manufacturer, or a malicious end user with physical access, could be the one attempting this.

- *Stealing of Decryption Key*
  Many modern FPGAs come with a built-in authentication block, e.g., based on keyed-hash message authentication codes (HMAC), which produces a fixed length message authentication code from an arbitrary length bitstream. Many FPGAs today also include a bitstream decryption (e.g., AES) block to support encrypted bitstream. An encrypted form of bitstream typically resides in the configuration flash. If authentication is used, the authentication key, and the hash digest, are encrypted along with the bitstream. As shown in Fig. 7.11, during power-up, the encrypted bitstream, authentication key, and hash are decrypted using the key stored inside the nonvolatile memory. Using the built-in authentication block inside the FPGA, a digest of the decrypted bitstream is generated, which is compared with the previously decrypted digest. If the bitstream has not been tampered prior configuration, the two digests must match. Hence, the symmetric encryption key is an essential secret in providing confidentiality and integrity to the bitstream. In the presence of a successful attack on the key, the bitstream not only becomes vulnerable to IP theft, but also could be tampered, because the authentication key is encrypted along with the bitstream. Side-channel attacks, such as differential power analysis (DPA), described in detail in Chapter 8, have been proven effective in retrieving the key [31–33]. Such attacks involve measuring and analyzing the power at power-up, when the key is used to decrypt the bitstream. The key could also be leaked through an adversary at VAR facility responsible for storing the key. If the key is stored inside the eFUSEs, the physical changes caused by the eFUSE programming could be seen through the metal layers in a decapped chip, using an SEM. Such an attack could only be mounted by a resourceful attacker, capable of performing destructive reverse engineering. Finally, during a remote upgrade, an attacker could try to obtain both the encrypted bitstream and the encryption key by intercepting the communication between the authorized person and the device.

1(b) Reverse Engineering: Bitstream reverse engineering (BRE) can allow an attacker to extract information on how a design was implemented. This facilitates intelligent modification of the IP, potentially for malicious reasons. An adversary may buy an existing FPGA-based product from the market; extract the IP through BRE; improve the functionality of the IP; and then use or resell it. Furthermore, bitstream tampering efforts to bypass certain restrictions may require the knowledge of the high-level design obtainable only through BRE. Successful plain-text BRE has been demonstrated for certain series of FPGA devices [34–36]. However, due to the absence of a standardized bitstream format, newer, potentially more sophisticated approaches, may be required for FPGAs of different series and vendors. The bitstream reversal process is further complicated by the presence of encryption. Unless an attacker has access to the key, the only way to understand the functionality of an encrypted bitstream (to a limited extent), is to treat the mapped design like a black-box, and observe the functional outputs for various inputs.

1(c) IP Overuse: IP overuse is one of the few instances in the lifecycle of an FPGA-based system, where the entity developing the system itself could be the adversary. At present, the system developer

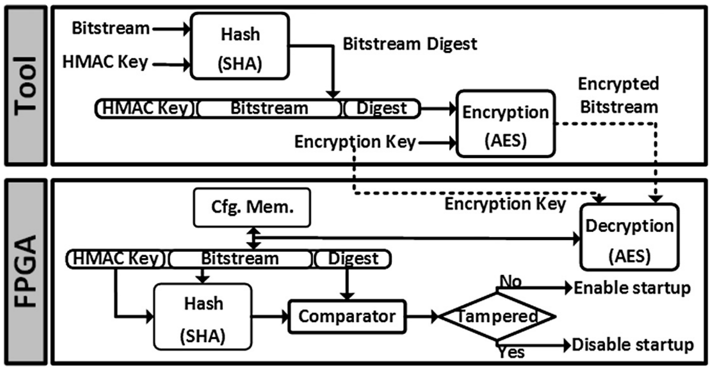


FIGURE 7.11

Typical flow for FPGA bitstream encryption and authentication.

buying third-party IPs in the form of RTL or bitstream could use them in any number of FPGAs. However, the IP developer may want her/his/its designs to be used in a fixed number of devices, or may want to charge per instance of use. To facilitate this, an active metering scheme that enables a pay-per-use licensing model was proposed in [37]. In current FPGA devices, if a unique unchangeable identifier is in place, the third-party IP provider could use that identifier to compile individual IPs for different devices to restrict the use of their IPs to a fixed number of devices. Node-locking of FPGA bitstream, similar to node-locked licensing approach in software, has been examined through low-overhead architectural modifications to the base array [38], and through a bitstream obfuscation technique [39].

2. Tampering: Malicious modification of bitstreams is a major concern for FPGA-based systems. An attacker could modify the bitstream to bypass certain restrictions, or to evade a security feature executed by the bitstream. Bitstream tampering could also be used to trigger a logical or physical malfunction at a specific time during the operation of the device. Several attack models exist for bitstream tampering, as discussed below.

- *Fault Injection*
  At run-time, individual bits of the mapped configuration could be altered by injecting faults in both a non-invasive and semi-invasive manner. While non-invasive attacks require no physical change to the target hardware, a semi-invasive attack requires a limited amount of hardware change to facilitate the attack. Non-invasive fault injection approaches include focused radiation and power adjustment [40]. A semi-invasive attack in the form of optical fault injection has been demonstrated with flashgun and laser pointer to change individual bits of SRAM in a microcontroller [41]. These equipments are readily available and are relatively inexpensive. Therefore, a similar attack model is a viable threat for SRAM FPGAs.
- *Direct Modification*
  Direct modification of unencrypted bitstream to implement hardware Trojans has been demonstrated in [42]. However, the attack focuses on modifying unused resources, which appears to be a string of zeros in the configuration bits. This facilitates easy modification without rendering the bitstream

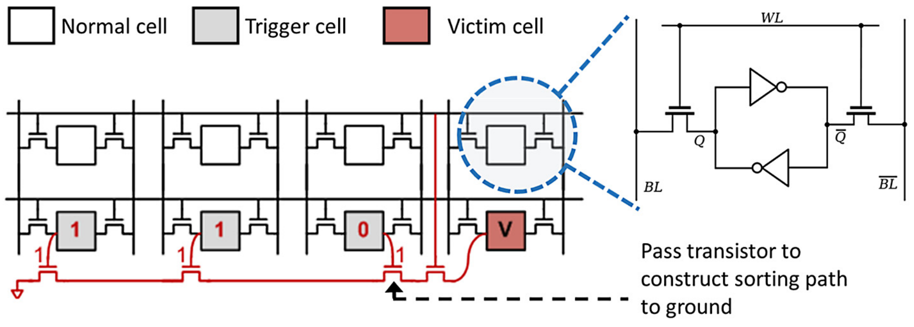
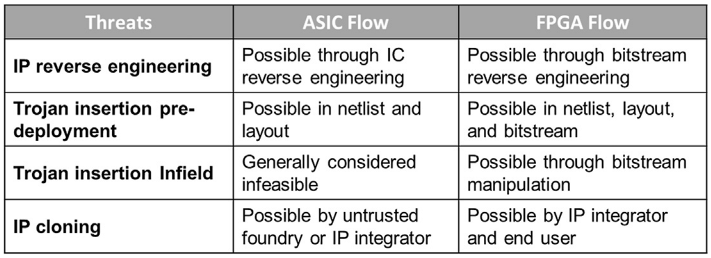


FIGURE 7.12

Hardware Trojan implemented in SRAM to compromise the value stored onto the victim cell (marked in red; dark gray in print version), when a specific pattern (1-1-0) is stored on the trigger cells (marked in gray).

nonfunctional, as may happen if used regions of the bitstream were modified instead. In [43], a cryptographic implementation of AES and 3DES on an FPGA was tampered by reverse engineering of the bitstream mapping format. This was done by iteratively mapping known functions, observing the changes in the bitstream, and repeating until the critical portion of the bitstream was identified. The ultimate goal of tampering was to leak a secret information that is processed within the design. In [44], a technique to extract the secret key of an FPGA implementation of an AES cryptomodule was demonstrated by performing a fixed set of bitstream manipulations. These rules are independent of the FPGA family and do not require an in-depth knowledge of the mapped design.

- *Hardware Trojan*
  During the manufacturing of FPGA devices in an untrusted foundry, hardware Trojans could be inserted into the base array layout, which, once triggered, can modify the configuration bits of a specific FPGA resource to induce a logical, or physical, malfunction. One motivation of the attacker at the foundry could be to create a bad reputation for the vendor, while providing a competitive edge to others [24]. The feasibility of implementing such Trojans in an SRAM array has been verified in [45]. As shown in Fig. 7.12, several trigger cells in the SRAM could be used to enable a path constructed using pass transistors that are maliciously inserted in the layout. If a particular pattern is stored onto those trigger cells, the path activates and shorts the victim cell to ground. The payload compromises the ability of the victim cell to store a specific value (0 or 1). Such Trojans could be used for forcing a desired value to specific configurable components (that is, LUTs) at a specific instance, as intended by the attacker.
- *Unauthorized Reprogramming*
  An FPGA could be reprogrammed with a completely different bitstream by an adversary. This could happen when the attacker has physical access to the FPGA, or can intercept the communication of bitstream during a remote upgrade. Such an attack could be mounted with the goal to infect other modules of the system using the compromised FPGA. Also, attackers may try to resell an FPGA-based product under a different vendor's name by replacing the original proprietary software with



Threats	ASIC Flow	FPGA Flow
IP reverse engineering	Possible through IC reverse engineering	Possible through bitstream reverse engineering
Trojan insertion pre-deployment	Possible in netlist and layout	Possible in netlist, layout, and bitstream
Trojan insertion Infield	Generally considered infeasible	Possible through bitstream manipulation
IP cloning	Possible by untrusted foundry or IP integrator	Possible by IP integrator and end user

FIGURE 7.13

Comparison of IP-oriented threats between ASIC and FPGA.

their own. If the original bitstream only allows proprietary software, such malicious reprogramming would be necessary.

The security issues pertaining to hardware IPs, including piracy and reverse engineering, have received significant attention in recent years. In this chapter, we focused on the major security issues for hardware IP (in both ASIC and FPGA design flow) during its life cycle. We analyzed that threat models for these flows, and pinpointed several open issues related to piracy, reverse engineering, and tampering. For a better understanding of these vulnerabilities, attacks are classified according to adversary's location, intention, and the surface of attack. Various forms of threats applicable to ASIC and FPGA are summarized in Fig. 7.13. Any modern electronic system relies on the trusted and secure operation of the underlying hardware IPs. Security issues in all hardware IPs used in a system need to be adequately addressed in order to build a secure and trusted system.

7.5 HANDS-ON EXPERIMENT: REVERSE ENGINEERING AND TAMPERING

7.5.1 OBJECTIVE

This experiment is designed to give students the opportunity to perform FPGA bitstream reverse engineering attacks. The experiment consists of several parts. These parts are designed on the HaHa platform, which utilizes an Altera MAX10 series FPGA chip. The experiment illustrates a method of reverse engineering an unencrypted bitstream with the goal of piracy or understanding the design intent, or its malicious modification.

7.5.2 METHOD

Students have to first map an example design into the FPGA module inside HaHa platform and generate the bitstream. Next, the students use a matching tool to compare the generated bitstream with an existing one; the goal of this tool is to identify known functions in the bitstream using a bitstream template. The students will also create various designs with minimal differences and compare the generated bitstreams.

7.5.3 LEARNING OUTCOME

By performing the specific steps of the experiments, the students will learn how bitstreams are generated and what format they use. They will use that knowledge to reverse engineer the bitstreams and retrieve the gate-level netlist of the design. They will also explore the challenges with respect to protecting an FPGA design against bitstream reverse engineering attacks.

7.5.4 ADVANCED OPTIONS

Additional exploration on this topic can be done through tampering the bitstream to obtain a hamming distance of 50% from the original output.

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

7.6 EXERCISES

7.6.1 TRUE/FALSE QUESTIONS

1. Modern-day ASIC design flow involves third-party vendors.
2. In-house designs are always trustworthy.
3. The primary goals of cloning an IP are piracy and overproduction.
4. Hardware Trojans could help in cloning ICs.
5. Foundries are considered trustworthy entities throughout the IC design flow.
6. Encrypted FPGA bitstreams cannot be tampered for bypassing a logic implemented within the design.
7. If encryption and authentication are used together, the threat of bitstream tampering can be completely mitigated.
8. The side-channel analysis is an invasive attack for stealing the decryption key.
9. Without knowing the bitstream to be mapped in the FPGA, a Trojan cannot be included in the FPGA hardware by the foundry.
10. FPGA-based implementations typically consume more power and area compared to their ASIC counterpart.

7.6.2 SHORT-ANSWER TYPE QUESTIONS

1. Which phases of the ASIC lifecycle are vulnerable to hardware attacks?
2. List the entities capable of launching attacks on hardware in different phases of the design flow.
3. What are the different methods of FPGA bitstream tampering?
4. Describe how built-in authentication and encryption features operate in an FPGA.
5. What is the motivation behind bitstream reverse engineering from an attacker's perspective?

7.6.3 LONG-ANSWER TYPE QUESTIONS

1. Describe the lifecycle of a modern electronic hardware / ASIC design.

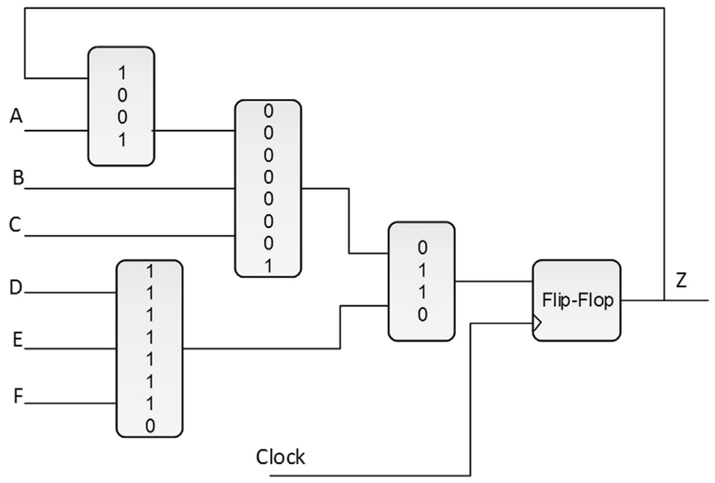


FIGURE 7.14

Figure for long-answer type question 6 and question 7.

2. Classify the attacks on hardware IPs in terms of the attacker's origin and intent. Briefly describe possible attack instances of hardware Trojan and IC overproduction.
3. How can an IP be exploited via reverse engineering? Briefly explain potential attack instances of reverse engineering.
4. Describe the typical development lifecycle of an FPGA-based product.
5. Describe the different attacks on an FPGA bitstream, which could occur during field operation.
6. Consider the FPGA synthesized netlist with four LUTs and one flip-flop (Fig. 7.14). The corresponding configuration bits are provided in binary (consider the top bit as MSB). Reverse the design to its gate-level version.
7. Consider the FPGA-synthesized netlist referred to in Question 6. With the minimum possible modification of LUT contents, mount a bitstream tampering attack that permanently inverts the output from the original one, and draw the tampered netlist.

REFERENCES

[1] Global Semiconductor IP Market Report 2018–2022.

[2] S. Skorobogatov, C. Woods, Breakthrough Silicon Scanning Discovers Backdoor in Military Chip, in: International Workshop on Cryptographic Hardware and Embedded Systems, Springer, pp. 23–40.

[3] E. Messmer, RSA Security Attack Demo Deep-Fries Apple Mac Components, 2014.

[4] R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: Threats and Emerging Solutions, in: High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International, IEEE, pp. 166–171.

[5] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M.S. Hsiao, J. Plusquellic, M. Tehranipoor, Protection against Hardware Trojan Attacks: Towards a Comprehensive Solution, IEEE Design & Test 30 (2013) 6–17.

[6] D. Forte, S. Bhunia, M.M. Tehranipoor, Hardware Protection Through Obfuscation, Springer, 2017.

[7] M. Rostami, F. Koushanfar, R. Karri, A Primer on Hardware Security: Models, Methods, and Metrics, Proceedings of the IEEE 102 (2014) 1283–1295.
[8] 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 (TODAES) 22 (2016) 6.
[9] S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan Attacks: Threat Analysis and Countermeasures, Proceedings of the IEEE 102 (2014) 1229–1247.
[10] F. Wolff, C. Papachristou, S. Bhunia, R.S. Chakraborty, Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme, in: Proceedings of the Conference on Design, Automation and Test in Europe, ACM, pp. 1362–1365.
[11] R.S. Chakraborty, S. Paul, S. Bhunia, On-demand Transparency for Improving Hardware Trojan Detectability, in: Hardware-Oriented Security and Trust, 2008. HOST 2008, IEEE International Workshop on, IEEE, pp. 48–50.
[12] S. Narasimhan, S. Bhunia, Hardware Trojan detection, in: Introduction to Hardware Security and Trust, Springer, 2012, pp. 339–364.
[13] E. Castillo, U. Meyer-Baese, A. García, L. Parrilla, A. Lloris, IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 15 (2007) 578–591.
[14] A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, G. Wolfe, Constraint-Based Watermarking Techniques for Design IP Protection, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20 (2001) 1236–1252.
[15] 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.
[16] S.E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, M. Tehranipoor, A survey on chip to system reverse engineering, ACM Journal on Emerging Technologies in Computing Systems (JETC) 13 (2016) 6.
[17] N. Asadizanjani, S. Shahbazmohamadi, M. Tehranipoor, D. Forte, Non-destructive PCB Reverse Engineering using X-ray Micro Computed Tomography, in: 41st International Symposium for Testing and Failure Analysis, ASM, pp. 1–5.
[18] M.C. Hansen, H. Yalcin, J.P. Hayes, Unveiling the ISCAS-85 Benchmarks: a Case Study in Reverse Engineering, IEEE Design & Test of Computers 16 (1999) 72–80.
[19] W.M. Van Fleet, M.R. Dransfield, Method of Recovering a Gate-Level Netlist from a Transistor-Level, 2001, US Patent 6,190,433.
[20] W. Li, Z. Wasson, S.A. Seshia, Reverse Engineering Circuits using Behavioral Pattern Mining, in: Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on, IEEE, pp. 83–88.
[21] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, S. Malik, Reverse Engineering Digital Circuits using Functional Analysis, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, pp. 1277–1280.
[22] I. Kuon, R. Tessier, J. Rose, FPGA Architecture: Survey and Challenges, Foundations and Trends in Electronic Design Automation 2 (2008) 135–253.
[23] K. Morris, Xilinx vs. Altera, calling the action in the greatest semiconductor rivalry, EE Journal (February 25, 2014).
[24] S. Mal-Sarkar, A. Krishna, A. Ghosh, S. Bhunia, Hardware Trojan Attacks in FPGA Devices: Threat Analysis and Effective Countermeasures, in: Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI, ACM, pp. 287–292.
[25] K. Wilkinson, Using Encryption to Secure a 7 Series FPGA Bitstream, Xilinx, 2015.
[26] Xilinx, Authorized Distributors, https://www.xilinx.com/about/contact/authorized-distributors.html, 2017. (Accessed 3 December 2017), [Online].
[27] S. Ghosh, A. Basak, S. Bhunia, How Secure are Printed Circuit boards Against Trojan Attacks? IEEE Design & Test 32 (2015) 7–16.
[28] Xilinx, Readback Options, 2009.
[29] R. Druyer, L. Torres, P. Benoit, P.-V. Bonzom, P. Le-Quere, A Survey on Security Features in Modern FPGAs, in: Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on, IEEE, pp. 1–8.
[30] S.M. Trimberger, J.J. Moore, FPGA Security: Motivations, Features, and Applications, Proceedings of the IEEE 102 (2014) 1248–1265.
[31] 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.
[32] A. Moradi, M. Kasper, C. Paar, Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures, in: Topics in Cryptology–CT-RSA 2012, 2012, pp. 1–18.

[33] A. Moradi, D. Oswald, C. Paar, P. Swierczynski, Side-Channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ACM, pp. 91–100.
[34] J.-B. Note, É. Rannaud, From the bitstream to the netlist, in: FPGA, vol. 8, p. 264.
[35] Z. Ding, Q. Wu, Y. Zhang, L. Zhu, Deriving an NCD file from an FPGA bitstream: Methodology, Architecture and Evaluation, Microprocessors and Microsystems 37 (2013) 299–312.
[36] F. Benz, A. Seffrin, S.A. Huss, Bil: a Tool-Chain for Bitstream Reverse-Engineering, in: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, IEEE, pp. 735–738.
[37] R. Maes, D. Schellekens, I. Verbauwhede, A Pay-Per-Use Licensing Scheme for Hardware IP Cores in Recent SRAM-based FPGAs, IEEE Transactions on Information Forensics and Security 7 (2012) 98–108.
[38] 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.
[39] 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.
[40] S. Trimberger, J. Moore, FPGA Security: from Features to Capabilities to Trusted Systems, in: Proceedings of the 51st Annual Design Automation Conference, ACM, pp. 1–4.
[41] S.P. Skorobogatov, R.J. Anderson, et al., Optical Fault Induction Attacks, in: CHES, vol. 2523, Springer, 2002, pp. 2–12.
[42] 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.
[43] 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.
[44] P. Swierczynski, G.T. Becker, A. Moradi, C. Paar, Bitstream Fault Injections (BiFl)–Automated Fault Attacks against SRAM-based FPGAs, IEEE Transactions on Computers (2017).
[45] T. Hoque, X. Wang, A. Basak, R. Karam, S. Bhunia, Hardware Trojan attack in Embedded Memory, in: IEEE VLSI Test Symposium (VTS), IEEE, 2018.