FPGA Bitstream Security: A Day in the Life

Adam Duncan*, Fahim Rahman†, Andrew Lukefahr*, Farimah Farahmandi†, Mark Tehranipoor†

*Intelligent Systems Engineering, Indiana University, Bloomington, Indiana 47401 USA
†Electrical and Computer Engineering, University of Florida, Gainesville, Florida 32611 USA
Email: adamdunc@indiana.edu

***Abstract*—Security concerns for field-programmable gate array (FPGA) applications and hardware are evolving as FPGA designs grow in complexity, involve sophisticated intellectual properties (IPs), and pass through more entities in the design and implementation flow. FPGAs are now routinely found integrated into system-on-chip (SoC) platforms, cloud-based shared computing resources, and in commercial and government systems. The IPs included in FPGAs are sourced from multiple origins and passed through numerous entities (such as design house, system integrator, and users) through the lifecycle. This paper thoroughly examines the interaction of these entities from the perspective of the bitstream file responsible for the actual hardware configuration of the FPGA. Five stages of the bitstream lifecycle are introduced to analyze this interaction: 1) bitstream-generation, 2) bitstream-at-rest, 3) bitstream-loading, 4) bitstream-running, and 5) bitstream-end-of-life. Potential threats and vulnerabilities are discussed at each stage, and both vendor-offered and academic countermeasures are highlighted for a robust and comprehensive security assurance.**

***Keywords*—FPGA Security, Encryption, Bitstream Protection**

I. Introduction

A field-programmable gate array (FPGA) is an integrated circuit with post-fabrication hardware programming capabilities used to implement custom functionality on a dedicated hardware platform [1]. Products ranging from low-cost consumer electronics to high-end commercial systems use FPGAs for reconfigurability, low development cost, and high-performance [2]. The specific hardware functionality programmed into an FPGA is defined by a binary file commonly known as a **bitstream** which is generated following a rigorous design, synthesis, and validation process. FPGAs are typically classified by the type of on-chip configuration memory used to store this bitstream file, with common examples being static random access memory (SRAM), Flash, and antifuse. Each configuration memory variant has associated performance, fabrication, and security tradeoffs as discussed in [2]. However, in each FPGA type, the primary FPGA-specific security concern eventually simplifies down to protecting the bitstream from either tampering or intellectual property (IP) piracy. Tampering an FPGA bitstream can compromise the root of trust, and thus the security, of an entire system. Just as consequential, IP piracy conducted at the bitstream level can have an enormous financial impact for the design house and system manufacturer.

A simplified design flow illustrating the loading of a bitstream into an FPGA is depicted in Figure 1(a). The FPGA manufacturer, such as Xilinx, Intel, or Microsemi, first produces the FPGA integrated circuit (IC), along with the proprietary bitstream development software. The user loads the design into the bitstream development software to generate the bitstream file. The bitstream is then loaded into the FPGA configuration memory when the device is powered on for functional operation.

Early FPGAs could only hold simple designs, e.g., 1000 ASIC equivalent gates for Xilinx XC2064 [3], making this design flow tractable. However, FPGA technology has matured, and the size and the complexity of the FPGA have grown over time. The Xilinx VU19P device released in 2019 contains over 9-million logic cells, or roughly 90-million ASIC gates [4]. A design utilizing a significant portion of these logic resources often requires a large team of designers, incorporating multiple third party IP (3PIP) blocks and legacy designs. The VU19P, like most recent FPGAs, also allows for partial reconfiguration, that is allowing a system programmer to reconfigure the FPGA while operating in the field with partial bitstream updates. Figure 1(b) shows the modern-day FPGA design flow with these additional entities interacting with each other and highlights their connection paths to the final bitstream responsible for the FPGA hardware configuration. As each entity shares a connection to the bitstream, they also pose a potential security threat to the authenticity, integrity, and confidentiality of the bitstream.

In this paper, we explore the journey an FPGA bitstream takes from conception to FPGA-based system obsolescence and present a comprehensive threat taxonomy to guide the reader. Industry and academic countermeasures are then presented to illustrate defenses against each threat. The reviewed protection mechanisms are composed of five stages: 1) bitstream-generation, 2) bitstream-at-rest, 3) bitstream-loading, 4) bitstream-running, and lastly, 5) bitstream-end-of-life (EOL). Our main contribution in this paper is to provide a comprehensive security assessment of the bitstream as it travels between these stages.

The rest of the paper is organized as follows: Related work and additional background information is provided in Section II. We introduce our bitstream lifecycle stages and present the threat taxonomy in Section III. Security threats and vulnerabilities, along with selected countermeasures, associated with the bitstream-generation stage are discussed in Section IV. Similar analysis is provided for the subsequent stages – bitstream-at-rest, bitstream-loading, bitstream-running, and bitstream-end-of-life – in Sections V, VI, VII, and VIII, respectively. Finally,

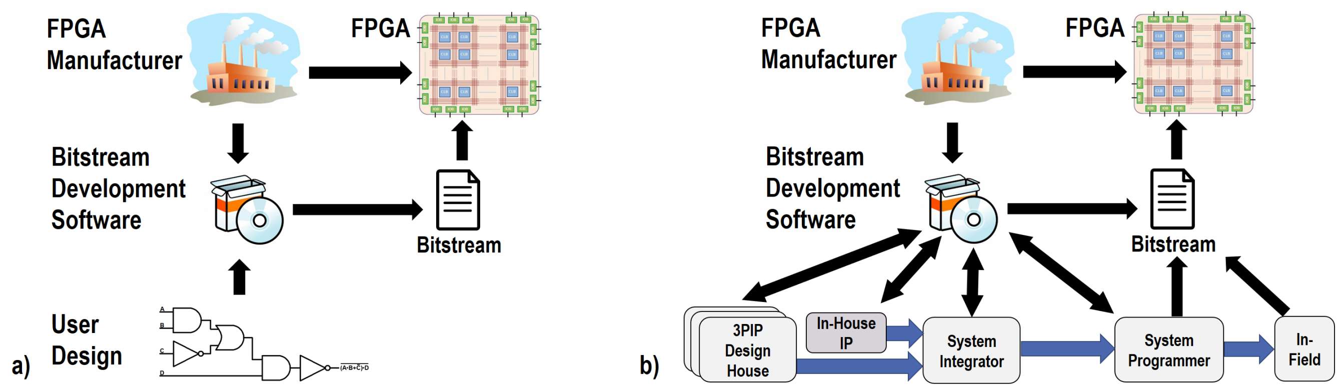

Fig. 1: a) Classical view of the FPGA design flow. b) Modern FPGA design flow involving multiple entities.

the paper is concluded in Section IX.

II. Background

Different entities involved in a modern and complex FPGA design flow are highlighted in Figure 1(b). The **3PIP design house** produces generic or client-specific IPs for the system integrator. The **system integrator** obtains and integrates the 3PIPs with in-house IPs to produce the actual bitstream for the FPGA. The **system programmer** represents the entity in charge of loading the bitstream into the FPGA. Lastly, **in-field** is used in reference of the FPGA operating in the field, such as inside a computer networking router, with its bitstream loaded into its physical configuration memory.

The physical configuration memory that stores the bitstream in an FPGA has a direct impact on the security and accessibility of a bitstream. SRAM-based FPGAs are the most common FPGA type, using volatile SRAM-based latches to store the bitstream. They are fabricated using standard state-of-the-art manufacturing processes allowing for high-performance and high-density [5]. However, they require off-chip bitstream storage, and must transmit the bitstream into the FPGA after it is powered on. Hence it is possible for an attacker to intercept the unprotected bitstream at the board level [5].

There also exist non-volatile FPGAs, such as Flash memory and antifuse-based, which store their bitstream inside the FPGA, eliminating the board-level bitstream interception problem. These FPGAs require additional manufacturing process steps and lack in the performance and density metrics of their SRAM-based counterparts [5]. Academic researchers have also proposed FPGA designs utilizing emerging non-volatile memories such as magneto-resistive RAM (MRAM) to produce higher performance non-volatile FPGAs [6].

Irrespective to complexity and memory architecture, FPGA security issues eventually simplify down to unauthorized access and tampering to the FPGA bitstream. For example, concerns may include attackers performing reverse engineering on proprietary IP or may involve the loading of an unauthorized design into an FPGA-based system to alter intended system behavior. Specific threats and countermeasures will be discussed throughout subsequent sections of this paper.

FPGA vendors have included bitstream protection features dating back to the earliest FPGAs. Xilinx published an application note in 1997 to program the FPGA at a secure facility and use a battery to maintain power throughout the lifetime of the system, preventing an attacker from intercepting the bitstream [7]. In 2001, Xilinx introduced bitstream encryption into their Virtex-II devices using the Data Encryption Standard (DES) [8]. Here, the bitstream is encrypted with an encryption key that is stored securely within the FPGA. Without knowledge of the encryption key, an adversary cannot reverse engineer or copy the bitstream. Other FPGA manufacturers have since included bitstream encryption in their devices, with encryption standards eventually migrating to include variants of the newer Advanced Encryption Standard (AES) [5].

In 2009, on-chip bitstream authentication was included by Xilinx in their Virtex-6 devices [5]. This authentication implements a keyed-Hash Message Authentication Code (HMAC) algorithm in hardware to compute the hash digest of a bitstream. The digest is compared to a pre-computed reference digest before bitstream loading, and the loading is aborted upon a mismatch. In 2015, Microsemi included physically unclonable function (PUF) protection to their bitstream encryption keys in their IGLOO2 and Smartfusion2 devices [9]. The PUF uses the inherent physical properties of the IC to generate a device-specific digital signature generated at runtime by the chip. This PUF value is then incorporated into the bitstream encryption scheme so that an attacker cannot thwart the encryption protection by obtaining the on-chip encryption key alone. Xilinx and Intel also offer similar solutions for their Ultrascale+ and Stratix-10 devices, respectively [10], [11].

III. Threat Model

The modern FPGA design flow experiences complex interactions among multiple involved entities as discussed in Section II. We present our threat taxonomy in Figure 2 to explore the threats and vulnerabilities facing the bitstream as it travels amongst these different FPGA entities. The top flow of Figure 2 illustrates the **Design Flow Entities** involved: 1) 3PIP Design House, 2) System Integrator, 3) System Programmer, 4) System in-Field, and lastly 5) Recycler.

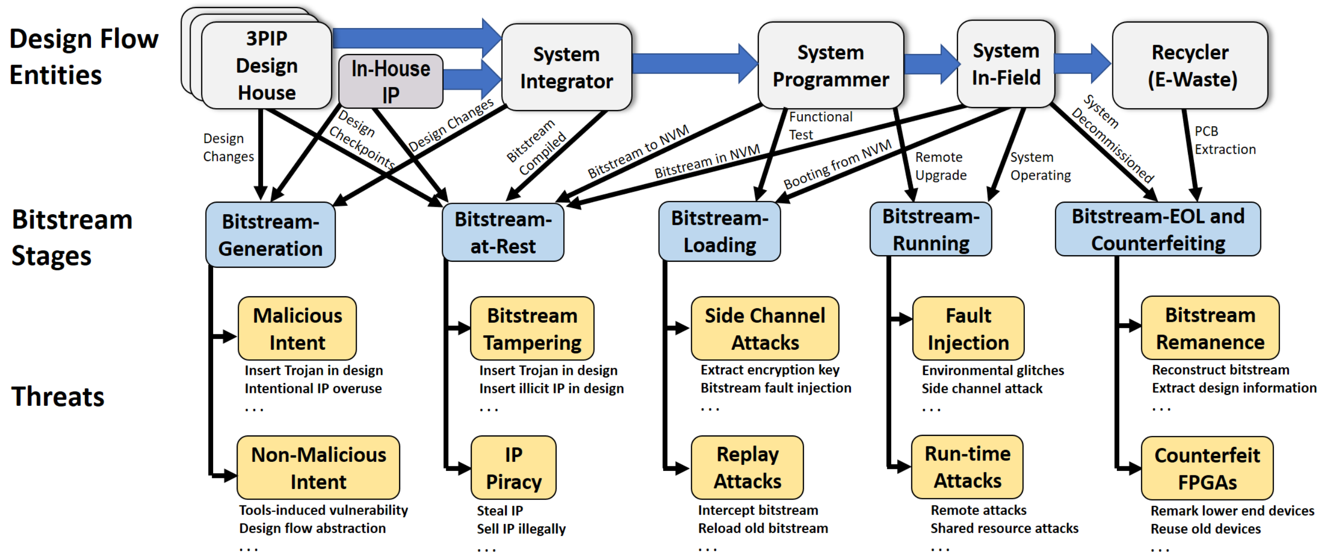

Fig. 2: A taxonomy of the different threats facing a bitstream as it traverses through a modern FPGA design flow composed of multiple entities.

Our five **Bitstream Stages** are located below these entities and describe the different points involved in the journey of a bitstream. **Bitstream-Generation** refers to the stage where the bitstream is physically being generated by either FPGA design tools or other means. **Bitstream-at-Rest** defines the stage where a bitstream has been generated and is stored either on a computer, in a cloud repository, or in a non-volatile memory that is not currently configuring the FPGA. **Bitstream-Loading** describes the physical act of loading the bitstream from its resting state into the FPGA configuration memory. **Bitstream-Running** is the state where a bitstream has been loaded into the configuration memory, and the FPGA is operating according to its programmed hardware configuration. Lastly, **Bitstream-EOL** is used to describe the decommissioning of the bitstream as well as physical FPGA-related threats tangentially related to the FPGA bitstream.

The interaction between the design flow entities and bitstream stages illustrates the complexity involved in modern FPGA security. The first observation is that each design flow entity has a connection to more than one bitstream stage. For example, the in-field system may contain a bitstream stored in a non-volatile memory on a PCB, categorized as bitstream-at-rest. After the system powers up, it enters the bitstream-loading stage, and transitions into the bitstream-running stage after the bitstream reaches the FPGA configuration memory.

Several threat categories are provided for each bitstream stage as shown in the bottom of Figure 2. The taxonomy also lists two examples for threat category. The subsequent sections of this paper will discuss these threats and associated countermeasures in detail with respect to each bitstream stage.

IV. Bitstream Generation

The life a bitstream begins with an intended hardware design specification that is targeted towards the FPGA. The design specification is then translated into a complete design IP that is either developed by the user, outsourced as 3PIP, includes other licensed IP, or is a combination of all. At this point, the FPGA design software takes the IP and synthesizes it into FPGA resources according to the targeted FPGA models and specifications. These resources are then placed within the FPGA fabric and routed together to create a final configuration. This final configuration is ultimately specified as the bitstream. This bitstream generation process can be seen in Figure 3. Two important things can be observed from this figure. First, the design flow is similar to that of an ASIC, and as such, the design specification and IP steps share the same threats and countermeasures found in the ASIC literature. Second, the synthesis, place and route, and bitstream generation steps have distinct differences compared to an ASIC, due to their reconfigurability and the fact that the physical FPGA fabric, including the model-specific hardware information, is often public and known to an attacker.

Our taxonomy in Figure 2 divides threats in the bitstream-generation phase into two categories: malicious intent and non-malicious intent. Malicious intent refers to an attacker deliberately performing an attack, such as Trojan insertion or IP overuse during the generation of a bitstream. Non-malicious intent is presented to cover the expanding threat space where vulnerabilities are unintentionally introduced by the complex FPGA design tools generating final bitstreams.

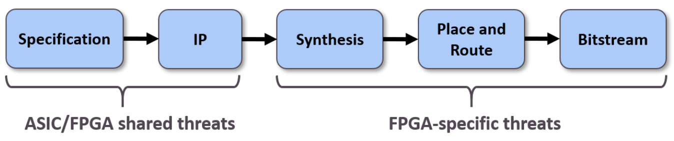

Fig. 3: A simplified view of the FPGA bitstream generation flow.

A. Malicious Design Flow Threats

Trojan Attacks: Attacks on the design IP within the bitstream generation flow are often very similar to attacks on design IPs in an ASIC design flow. For example, hardware Trojan insertion [12] shares the same basic attack principles for register transfer level (RTL) design IP, independent of whether the IP is targeting an FPGA or an ASIC. Once the design IP has been synthesized into the specific design elements inside the FPGA, the synthesized blocks become vulnerable to Trojan insertion attack. Mal-Sarkar et al. discussed FPGA-specific post-synthesis threat vectors [13] as illustrated in Figure 4. Here, logic blocks contain programmable lookup tables (LUTs), combinational logic primitives such as adders, and sequential elements in the form of flip-flops and latches. These elements are combined to implement combinational and sequential logic functions. There are also routing elements: local interconnects, connection boxes, and switch boxes which are used to route outputs between logic blocks.

An attacker, such as a rogue employee with access to the design in this post-synthesis state can change the properties of logic blocks to introduce a Trojan or modify the design functionality in some way. After synthesis, logic blocks go through a place and route step where they are placed within the FPGA fabric at specific locations. Similarly, routing elements are placed and configured to achieve the desired design functionality. An attacker here can potentially modify the placement of the blocks or add additional blocks to the design.

Lastly, the generated bitstream provides the correlation between the configuration memory inside the FPGA and the behavior of the logic blocks and routing elements. The bitstream can be attacked directly to modify the configuration memory which in turn modifies the functionality of FPGA, as will be discussed in Section V. During the synthesis and place and route steps, the designs are often checkpointed by the bitstream development software. These design checkpoints allow for the possibility of an insider threat to modify or insert elements in the design by the editing of the intermediate software file or by even creating a malicious modification to the FPGA design software [14].

Trojan Countermeasures: Borrowing from ASIC Trojan detection work, techniques presented by Salmani et al. [15] can be used to detect Trojans in FPGA designs at the IP and synthesis levels. These techniques operate on the principle that the triggering of a Trojan is likely a rare occurrence, and thus potentially identified by profiling and/or simulating a design to probe for rarely activated logic. At the place and route level, techniques have been presented for ASICs using the built-in self-authentication (BISA) [16] approach to add a test infrastructure inside a design to test for the placement of additional malicious logic. Khaleghi et al. extended this concept into the FPGA space to fill unused Logic Blocks and routing elements with a test-verifiable dummy design to prevent attackers from utilizing unused FPGA resources to insert Trojans [17].

IP Piracy Attacks: IP piracy-based threats in the bitstream-generation phase [18] involve common threats familiar to ASIC and software IP piracy. IP overuse refers to implementing more instances of an IP than specified by the IP licensing agreement, and it is becoming a larger threat as the market for FPGA IP grows. IPs without specific licensing protection are vulnerable to an attacker generating more bitstreams than allowed. IP theft, IP reuse, and IP reverse engineering are also a growing concern as techniques have been published discussing tool flows to convert between intermediate formats in the FPGA design flow cycle [19]. The specific issues regarding the direct manipulation of the bitstream at the end of the FPGA design flow are discussed in Section V.

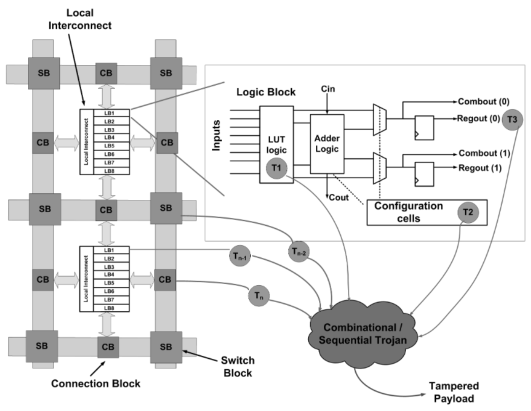

Fig. 4: The fundamental building blocks of the FPGA with highlighted Trojan insertion points [13].

IP Piracy Countermeasures: To detect the instances of IP piracy, watermarks [20] can be inserted by the user in the IP design stage and then evaluated at a later time to provide a proof of authorship. FPGA vendor software packages currently offer the distribution of third-party IP encrypted using IEEE standard p1735 [21] to protect against reverse engineering activities. To further protect against IP overuse, researchers have proposed methodologies incorporating a PUF response from the chip into the licensing to generate a device-specific key to enable design functionality within a given device [22]. The general concept is shown in Figure 5, where a locked bitstream component containing the IP is stored alongside a challenge in a non-volatile memory. At runtime, the PUF is evaluated, and its response is used to unlock the IP to enable its design functionality. Two-party variants of these licensing schemes have been proposed as well to improve efficiency [23], [24]. Logic obfuscation is another powerful technique used at the design level to defend against IP piracy [25]. Typical logic obfuscation schemes integrate logic locking gates into a design to disable normal functionality unless correct values are applied to the logic locking gate inputs.

B. Non-Malicious Threats

Attacks: Traditionally, the vendor bitstream generation tools do not inherently offer security checking while they implement the design flow. Consequently, there may be unintended security vulnerabilities introduced during the trans-

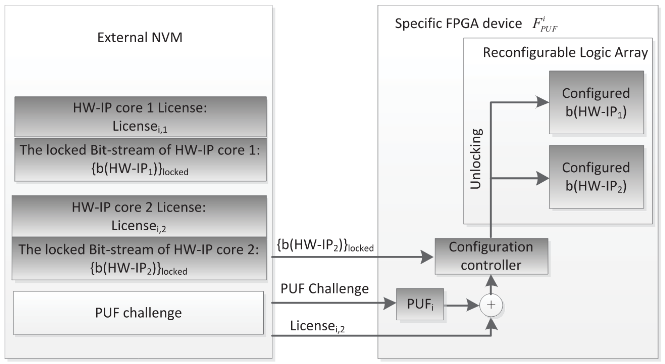
External NVM
HW-IP core 1 License: License$_{i,1}$
The locked Bit-stream of HW-IP core 1: $\{b(HW\text{-}IP_1)\}_{locked}$
HW-IP core 2 License: License$_{i,2}$
The locked Bit-stream of HW-IP core 2: $\{b(HW\text{-}IP_2)\}_{locked}$
PUF challenge
Specific FPGA device F^i_{PUF}
Reconfigurable Logic Array
Configured b(HW-IP$_1$)
Configured b(HW-IP$_2$)
Unlocking
$\{b(HW\text{-}IP_2)\}_{locked}$
Configuration controller
PUF Challenge
PUF$_i$
License$_{i,2}$

Fig. 5: A PUF-based licensing scheme binding IP to specific FPGA devices to prevent FPGA IP piracy and overuse [22].

lation of a design into the logic blocks inside the FPGA. High-level synthesis (HLS) specifically creates a higher level of design abstraction for logic block representation, which may increase the probability of unintended vulnerabilities. An example for this may be an AES encryption engine implementation, which is cryptographically secure at the C language abstraction but leaks information when it is synthesized to a hardware logic for an FPGA implementation [26]. Research findings compromising FPGA bitstream generation tools at various stages have also been published [14].

Countermeasures: The defense against tool-induced vulnerabilities first begins by adhering to proven best software security processes and verifying the tool authenticity through the use of a trusted vendor-provided hash during tool download and installation. At the design level, researchers have proposed a moving target defense to defend against attacks originating from malicious FPGA software tooling [27]. The movement of the target in this situation is the randomization of the synthesis and place and route operations by the vendor tools so that the attacker cannot predict the necessary information from a user design required to conduct a meaningful attack. Researchers have addressed high-level Trojan insertion by proposing Trojan-aware HLS [26]. Here, equivalence checking is performed between the original higher-level code and the lower-level code during the design space exploration (DSE) of the HLS operation. Moreover, a set of security properties can be developed to be used in formal verification tools to ensure the safe design translation. FPGA software vendors can also provide design checkpoint hashes which should be used along with proven best practices for software security during the design development.

V. Bitstream-at-Rest

After a bitstream has been generated, it needs to be stored someplace so that it can eventually be loaded onto the FPGA on its quest to perform its intended function. We use the term **bitstream-at-rest** to define this storage state. Bitstream storage locations for this stage can include multiple locations, including the hard disk of the computer used to run the FPGA development software during bitstream generation. Other storage locations can include a non-volatile memory used to configure the FPGA upon the application of power, or even a software repository of system-level firmware images containing an FPGA bitstream. Bitstreams in this state may be stored in their encrypted or plaintext versions, and are vulnerable to tampering or IP extraction.

In order to develop an attack against a bitstream or to extract its IP, a relationship between the bitstream and its hardware behavior must be established. The format of a bitstream for Xilinx, Intel, and Microsemi FPGAs is vendor proprietary and often serves as the first line of defense against such threats. However, multiple researchers have published techniques to successfully reverse engineer a vendor-proprietary plaintext bitstream into a netlist [29], [28]. The flow by Zhang et al. [28] depicted in Figure 6 is an example of one such technique where the bitstream is parsed into a functioning netlist that can be simulated and analyzed. Once a netlist has been obtained, it can be used to analyze a design for Trojans [28], or used for malicious activities such as IP piracy or tampering.

Our threat taxonomy in Figure 2 divides threats in the bitstream-at-rest stage into bitstream tampering and IP piracy categories. Example attacks and countermeasures are presented below.

A. *Bitstream Tampering*

Attacks: Chakraborty et al. first introduced the concept of Trojan insertion using plaintext bitstream manipulations in 2013 [30]. Since then, researchers have increased the sophistication of bitstream manipulation attacks to create automated blind attacks on soft IP blocks within an FPGA, such as soft-encryption cores [31]. Bitstream reverse engineering can further refine the scope of bitstream-based attacks to target specific soft IP blocks inside the FPGA fabric [32].

Countermeasures: Techniques proposed by Kamali et al. [33] and Karam et al. [34] help defend against tampering attacks by applying logic locking at the bitstream level to help obfuscate the netlist functionality from the attacker. To accomplish the logic locking, the authors construct keys at runtime using PUFs implemented within the FPGA fabric. The PUF responses for each device are then connected to lookup tables (LUTs) within the user design to act as a key so that

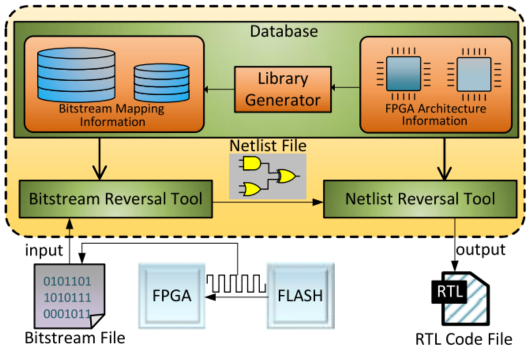

Fig. 6: A reverse engineering workflow translating a decrypted bitstream into a netlist [28].

correct design functionality will only occur if the correct key value is applied. Hence, if the attacker does not know the correct key values, the attacker will not be able to extract the correct functional netlist and as a consequence, will not be able to find the desired node to tamper.

B. *IP Piracy*

Attacks: Another attack goal may be to extract proprietary IP from a bitstream. Motivations for this may include not paying for IP and then subsequently using the IP illegally in a design, or reverse engineering an IP to extract proprietary information that may be used commercially.

Countermeasures: Inserting watermarks at the bitstream level has been proposed by Schmid et al. [35]. In contrast, RTL watermarking, this technique directly embeds a watermark into the LUT contents of a design. Watermark extraction and comparison is then performed at the bitstream level to determine authorship.

C. *Single Key Encryption*

Countermeasures: Modern Xilinx, Intel, and Microsemi FPGAs offer encrypted versions of the generated bitstreams to increase resistance to bitstream tampering, piracy, and reverse engineering activities. In fact, modern Microsemi FPGAs such as the Polarfire, Igloo2, and SmartFusion2 series, *only* store encrypted versions of their bitstream [36]. While researchers have demonstrated bitstream tampering attacks on encrypted bitstreams [31] that have an observable behavior, encryption makes targeted tampering attacks infeasible without knowledge of the encryption key.

D. *Red/Black Encryption*

Countermeasures: The concept of a red/black encryption scheme has been adopted by Intel, Xilinx, and Microsemi in their respective Stratix 10, Ultrascale+, and Polarfire product lines. The basic concept is outlined in Figure 7 for the Xilinx Ultrascale+ Zynq [37]. Here, the *red* key used to decrypt the bitstream is not stored directly inside the FPGA. Instead, a device-specific PUF is used to generate a *black* key that is stored inside the FPGA. Upon power-up, the PUF is exercised, and its response (black key) is used to generate the red key for decrypting the bitstream used to populate the FPGA fabric. As the black key is not directly used to encrypt the bitstream, it cannot be used by itself to decrypt the bitstream by an attacker.

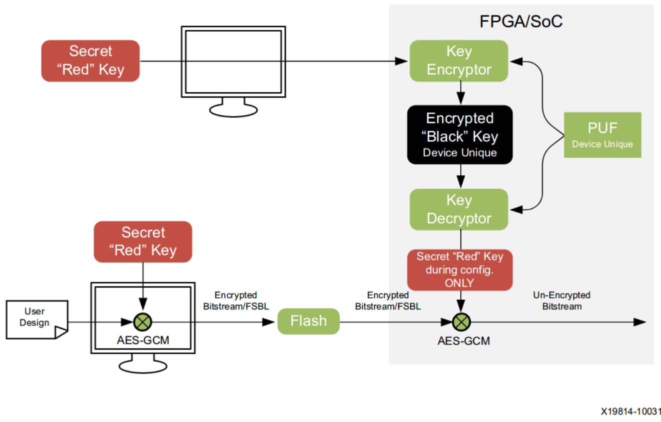

Fig. 7: A red/black encryption key flow where the key decrypting the bitstream at runtime is obfuscated from the key stored inside the FPGA [37].

VI. Bitstream-Loading

The **Bitstream-Loading** stage loads the bitstream into the configuration memory of the FPGA. The specifics of this stage vary depending upon the configuration memory variant from different vendors and FPGA device models. Non-volatile memory-based FPGAs, such as Flash-based or antifuse-based, only experience this stage when loading a new bitstream. SRAM-based FPGAs require the bitstream to be loaded every time power is applied to the FPGA to turn it on for functional application. A set of on-chip authentication and decryption circuitry is often employed by the FPGA during this stage to both authenticate and decrypt the bitstream before loading it into the configuration memory. FPGA-based system-on-chip devices, such as the Xilinx Zynq family, add additional features to the bitstream loading process in terms of bootloaders as well as physical processor cores implemented on the same silicon. Attacks considered within this stage originate from unintended side channels as well as loading outdated, and potentially vulnerable, bitstream versions.

Our threat taxonomy in Figure 2 divides threats in the bitstream-loading stage into two categories. First, we discuss side channel threats which involve the extraction of sensitive on-chip information during the loading of the bitstream. Second, we discuss replay attacks, where older, or potentially unauthorized versions of a bitstream are loaded into the FPGA.

A. *Side Channel Threats*

Attacks: Side channel attacks (SCA) have been applied to earlier generations of FPGAs to extract encryption keys by collecting information through unintended side channels. Encryption keys have been extracted in early generations of FPGAs by researchers analyzing the power consumption during the decryption process [38]. Similarly, information from a user design running in the fabric has been shown to leak in the electromagnetic spectrum [39]. Recently, laser-based approaches have shown the ability to read out on-chip information [40].

Countermeasures: FPGA vendors have addressed these attacks by implementing side-channel defenses [36], [10], [11] in their latest products to eliminate the leakage of key material. Key rolling limits the amount of time an attacker has to extract a given key, defeating attacks that rely upon multiple samples such as differential power analysis (DPA). The black/red scheme discussed in section V-D reduces the impact of a key being extracted as well since the key stored in the non-volatile memory of the FPGA is not the final key used to encrypt or decrypt the bitstream. In addition, the academic research

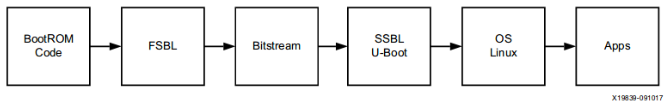

Fig. 8: Xilinx secure boot process flow where user code first stage bootloader (FSBL) loads the bitstream into the programmable logic of an FPGA-based SoC [37].

community has proposed physical techniques such as nanopyramids [41] to defend against these attacks. Nanopyramids are intended to be inserted in the device manufacturing flow to introduce random changes in the optical reflectance properties of silicon when conducting optical probing attacks, preventing an attacker using reflectance information to reveal information about the corresponding circuit storing key material.

B. Bitstream Replay Threats

Attacks: Bitstream versioning refers to the concept of having multiple versions of a bitstream for a given FPGA-based system. Analogous to the software world, a vulnerability can be discovered within an FPGA bitstream, requiring an updated bitstream to be loaded into the device. However, if the FPGA has already been deployed in the field, an adversary can potentially *downgrade* it to use the original bitstream containing the vulnerability [36]. Classical encryption and authentication techniques do not protect against this concern as the original vulnerable bitstream was encrypted and authenticated with the same encryption key as the updated bitstream. The act of securely transmitting an updated bitstream to the FPGA is another security concern.

Countermeasures: Microsemi addresses replay attacks by implementing a versioning control in their bitstream, combined with setting different non-volatile version control bits within their FPGA [36]. Xilinx offers QuickBoot [42] as a solution to load different bitstream versions in different non-volatile memory locations depending upon bits set in the bitstream. Researchers have addressed possible security concerns with the concept of transmitting an updated bitstream to a device by implementing an authenticated station-to-station protocol [43] or implementing custom protocols within user logic [44].

C. FPGA-based SoCs

Attacks: The introduction of the SoC-based FPGAs such as the Xilinx Zynq [37] and Microsemi SmartFusion [9] adds additional steps to the loading of the bitstream. SoC-based FPGAs introduce the concept of a first stage bootloader (FSBL), which is a user code designed to facilitate the loading of the bitstream as well as to configure the non-FPGA aspects of the SoC, such as the processor and other hard IP blocks. Figure 8 shows the Xilinx “secure boot” implementation where immutable BootROM code is used to boot the SoC and run the user code within the FSBL, which eventually loads the bitstream [37]. Attacks in this boot process can thus result from running a malicious FSBL code in the SoC processor.

Countermeasures: To protect the privacy and integrity of the FSBL, SoCs typically implement a FSBL authentication

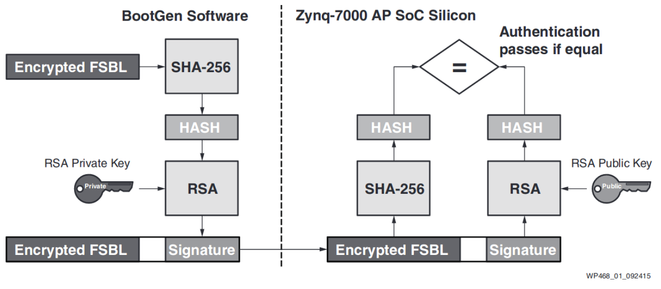

Fig. 9: Xilinx Zynq first stage bootloader (FSBL) authentication process [45].

scheme, such as the RSA-based authentication applied to the Xilinx Zynq series shown in Figure 9. Here, a public/private key pair is used to compare a hash signature on FSBL code with a hash signature stored in the FPGA’s non-volatile memory to only run authenticated FSBL code [45].

VII. Bitstream-Running

The **Bitstream-Running** stage defines the stage when the bitstream has been loaded into the configuration memory, and the FPGA is operating according to its hardware configuration. As shown in our threat taxonomy in Figure 2, this stage is vulnerable to fault injection and run-time threats originating within the fabric. Faults injected into the configuration memory, or directly into logic blocks and routing resources, can modify the functionality of the FPGA and are a primary concern in this stage. To correct faults in the configuration memory, and to provide the FPGA designer with more flexibility, modern FPGAs also include a partial reconfiguration framework within their architectures to allow for the bitstream to change dynamically at run-time. Partial reconfiguration allows the FPGA design itself to update portions of the design at run-time while keeping the remainder of the design intact.

We divide threats in the bitstream-running stage into two categories according to our threat model in Figure 2. First, we discuss fault injection on a running FPGA design. Next, we discuss the emerging topic of run-time attacks.

A. Fault Injection

Threats: Faults may be injected into an FPGA running a design through a variety of means, such as clock glitches, power glitches, electromagnetic pulses, laser exposure, or ionizing radiation [46]. The physical mechanisms behind fault injections have root in the physical transistors themselves. Therefore, threats to integrated circuits can be considered applicable to FPGAs. For example, random bit flips in the configuration memory caused by atmospheric single event upsets (SEUs) [47] are more of a concern as technology feature sizes shrink and thus more of a concern for newer FPGAs fabricated in state-of-the-art manufacturing processes. Targeted laser-based fault injection [48] has also been discussed by researchers, primarily as a means to replicate SEUs for hardening designs to space radiation effects.

Countermeasures: Partial reconfiguration cores, such as the internal configuration access port (ICAP) for Xilinx FPGAs, are included in modern FPGAs. Providing a user design with access to the partial reconfiguration core in an FPGA has been largely regarded as a security vulnerability [5] as the user design can then have the capability to read and write any area of the configuration memory. However, partial reconfiguration is also used as a mechanism to detect and correct for inadvertent bit flips in the configuration memory, such as those caused by radiation-induced single event upsets [49]. Several academic papers have proposed the use of the Xilinx ICAP to read the configuration memory of an FPGA at run-time and generate a hash for comparison against an expected hash in order to detect run-time tampering [50], [51]. In any usage of a partial reconfiguration core, proper safeguards must be put in place.

B. Run-time Attacks

The massively parallel nature of FPGAs has lent themselves to inclusion in data centers where users can purchase computing time. Amazon offers fee-based access to its FPGAs in the cloud through their Amazon Web Service (AWS) program [52]. Here, a shell architecture is described to abstract away communication links and create a separate application area in the FPGA. First, a design is created containing an 'AWS partial reconfigurable (PR) shell' to facilitate the loading of a user design. Next, the user design is loaded by the AWS PR shell to fit into the designated user 'custom PR logic' section of the floorplan. The shell protects a Peripheral Component Interconnect (PCI) Express connection in the 'static' region, manages clocking for the user region, and monitors activity elsewhere in the FPGA [53].

Threats: The shared FPGA computing resources described above have enabled a new class of remote side-channel attack, where one bitstream can leak or corrupt information in another bitstream, from a remote location. The example attack typically runs a user design, such as an oscillator-based array, on a shared FPGA fabric in order to affect another user's design [54], [55]. In Figure 10(a), Schellenberg et al. showed that a malicious bitstream can extract secrets from a victim bitstream sharing the same FPGA fabric [54] by corrupting the power in the shared power distribution network (PDN). It is also shown that malicious code running in the FPGA SoC can corrupt the PDN in a similar way to extract secrets from a victim FPGA design (see Figure 10(b)). Similarly, research has been performed to illustrate the possibility of a shared *system-wide* resource like the printed circuit board (PCB) PDN being used to affect an FPGA design. Here, the PCB PDN is corrupted by another PCB component to induce faults or leak information from the FPGA [55].

Countermeasures: Vendor-provided defenses to this type of attack include a bitstream-level screening of tenant bitstreams to check for suspicious functionality, such as multiple parallel ring-oscillator arrays that could potentially create power glitching. Modern FPGAs also incorporate on-chip voltage and temperature sensors to allow for the detection of

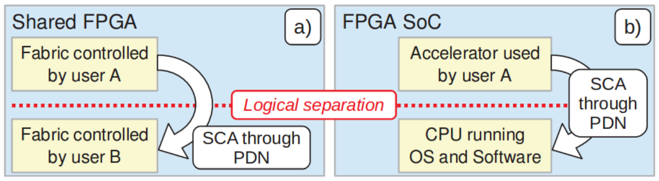
Shared FPGA
a)
Fabric controlled by user A
Fabric controlled by user B
SCA through PDN
Logical separation
FPGA SoC
b)
Accelerator used by user A
CPU running OS and Software
SCA through PDN

Fig. 10: Remote side-channel attack where user bitstream information can be extracted by a) malicious bitstreams on the same FPGA, or b) malicious CPU code on the same FPGA SoC [54].

anomalies in a shared FPGA resource, like the PDN. Vendor-provided soft-core defenses, such as the Xilinx Security Monitor (SecMon) core, implement the reading of on-chip voltage and temperature sensors, as well as the configuration memory health, to detect anomalous behavior and implement tampering penalties such the zeroization of the configuration memory, AES keys, or asserting the global reset of the FPGA [56].

VIII. Bitstream-End-of-Life

The last stage in our bitstream's lifecycle is defined as **bitstream-end-of-life**. We use this term to represent both the end-of-life (EOL) of a bitstream as well as a stage to capture threats to the physical FPGA device that are tangentially related to the bitstream. EOL in this context refers to when the FPGA running a bitstream has been decommissioned. This could refer to a formal decommission and destruction of a high-value proprietary system or the casual disposal of an FPGA-based networking router to a public trashcan. For other threats related to the bitstream, we briefly address FPGA device counterfeiting and reverse engineering.

We again refer to our threat taxonomy in Figure 2 and focus our discussion on two categories: data remanence and FPGA device counterfeiting.

A. Bitstream Remanence

Threats: As FPGA-based systems reach EOL, their bitstreams are also retired from use. A bitstream residing in an on-board non-volatile memory chip initially designed to program an SRAM-based FPGA may remain on that board indefinitely, remaining vulnerable to potential bitstream reverse engineering activities. Similarly, a bitstream stored in a Flash or antifuse-based FPGA may remain on the FPGA after the system has been disposed, creating a potential opportunity for bitstream extraction.

Countermeasures: FPGA vendors have incorporated zeroization mechanisms into their on-chip security features that allow for certain information stored within an FPGA to be deleted by a user or as a tamper penalty. This information space may include the original bitstream, or any other volatile and non-volatile information inside the FPGA. The Microsemi PolarFire FPGA family offers three levels of zeroization: like-new, recoverable, and unrecoverable [36]. The like-new option deletes user data and keys and returns the device to its factory state. Recoverable is more comprehensive and places the device in a state that is only recoverable by a Microsemi

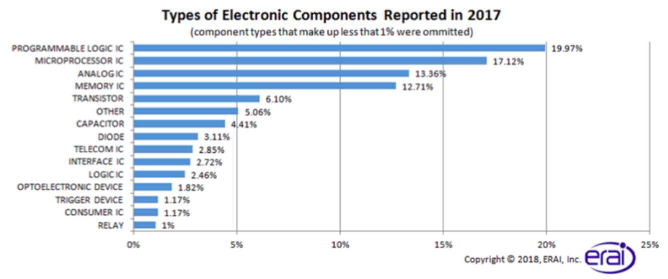

Fig. 11: programmable ICs (FPGAs) were the most reported counterfeit device type reported by ERAI in 2017 [22].

factory programming file. The unrecoverable option is the most thorough, incorporating the destruction of all on-chip data. Serialization certificates are provided by the device in all three cases via a JTAG/SPI instruction to prove that the operation was successful. Similar procedures are provided by other manufacturers as well [56].

B. FPGA Device Counterfeiting

Threats: FPGAs are frequently among the most popular counterfeit IC device types. As shown in Figure 11, ERAI listed FPGAs as occupying approximately 20 percent of their reported counterfeit part instances [57]. An example FPGA counterfeiting technique involves the selling of used devices as new devices. Remarking devices to represent more expensive devices is also a concern, as legacy and industrial/military grade FPGAs sell for a significant premium compared to their standard counterparts. These counterfeits FPGAs pose a threat to systems as their electrical and mechanical specifications as well as reliability are subject to compromise.

Countermeasures: FPGA vendors have addressed counterfeiting by both offering newer FPGA product lines designed to serve as drop-in replacements for legacy FPGAs and making their devices more difficult to counterfeit. Xilinx offers pin-compatible devices in their newer Ultrascale+ product lines that can replace older Ultrascale devices [58]. The major FPGA vendors also offer device-specific markings, such as unique packaging lid shapes, that defend against simple package remarking attacks [59]. Academic researchers have proposed the electrical characterization of oscillator structures programmed into the FPGAs to tease out reliability physics mechanism responses such as negative bias temperature instability (NBTI) and hot carrier injection (HCI) that can indicate whether an FPGA has had previous usage [60].

FPGA vendors have begun incorporating fabric-accessible mask-level device serial numbers and lot numbers, such as the DeviceDNA information found in Xilinx FPGAs, to allow users to determine the authenticity of a given FPGAs. DeviceID information indicating an FPGA product family line is often accessible through the IEEE joint test action group (JTAG) interface as well. Other counterfeit detection techniques designed for ASICs are also applicable to FPGAs and considered out of the scope for this paper [61].

IX. Conclusion

Our journey through the life of a bitstream has now come to an end. We have ventured through five different stages within the bitstream lifecycle: 1) bitstream-generation, 2) bitstream-at-rest, 3) bitstream-loading, 4) bitstream-running, and 5) bitstream-end-of-life. Each stage offered a connection to different entities in the FPGA design flow and contained unique threats along with countermeasures available from both FPGA vendors and academia. A threat taxonomy was introduced to capture the complex interactions between the bitstream stages and the design flow entities and highlight stage-specific threats.

Our threat taxonomy divided threats into two broad categories according to each distinct bitstream stage. More specific threats and countermeasures were discussed in each threat category to help inform the reader of the current state of the art. As with any security-based research, a holistic approach towards security is recommended for each design flow entity to identify the pertinent threats and implement appropriate countermeasures.

References

[1] Microsemi, "Field-programmable gate array technology.." Norwell, MA, USA:Kluwer, 1994.

[2] S. Trimberger, "Three ages of fpgas: A retrospective on the first thirty years of fpga technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015.

[3] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable gate array," in *Proceedings Custom Integrated Circuits Conference*, pp. 233–235, IEEE, 1986.

[4] Xilinx, "Ultrascale fpga product tables and product selection guide.." Xilinx, 2016.

[5] S. M. Trimberger and J. J. Moore, "Fpga security: Motivations, features, and applications," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1248–1265, 2014.

[6] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, "Spin transfer torque (stt)-mram–based runtime reconfiguration fpga circuit," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 2, p. 14, 2009.

[7] Xilinx, "Configuration issues: Power-up, volatility, security, battery back-up." Xilinx, Appl. Note XAPP092, 1997.

[8] Xilinx, "Method and apparatus for protecting proprietary configuration data for programmable logic devices." U.S. Patent 6 654 889, 2003.

[9] Microsemi, "Ug0443 user guide smartfusion2 and igloo2 fpga security and best practices." 2015.

[10] E. Peterson, "Xapp1098 (v1.3): Developing tamper-resistant designs with ultrascale and ultrascale+ fpgas," 2018.

[11] Intel, "Ug-s10security:intel stratix 10 device security user guide," 2019.

[12] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, pp. 10–25, 2010.

[13] S. Mal-Sarkar, A. Krishna, A. Ghosh, and S. Bhunia, "Hardware trojan attacks in fpga devices: threat analysis and effective counter measures," in *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, pp. 287–292, ACM, 2014.

[14] C. Krieg, C. Wolf, and A. Jantsch, "Malicious lut: a stealthy fpga trojan injected and triggered by the design flow," in *Proceedings of the 35th International Conference on Computer-Aided Design*, p. 43, ACM, 2016.

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

[16] K. Xiao and M. Tehranipoor, "Bisa: Built-in self-authentication for preventing hardware trojan insertion," in *2013 IEEE international symposium on hardware-oriented security and trust (HOST)*, pp. 45–50, IEEE, 2013.

[17] B. Khaleghi, A. Ahari, H. Asadi, and S. Bayat-Sarmadi, "Fpga-based protection scheme against hardware trojan horse insertion using dummy logic," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 46–50, 2015.

[18] A. Lesea, "Ip security in fpgas," *Xilinx http://direct. xilinx. com/bvdocs/whitepapers/wp261. pdf*, 2007.

[19] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: towards an open-source tool flow," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 41–44, ACM, 2011.

[20] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu, "Zero overhead watermarking technique for fpga designs," in *Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pp. 147–152, ACM, 2003.

[21] IEEE, "Ieee recommended practice for encryption and management of electronic design intellectual property (ip).ieee sa-1735-2014." 2014.

[22] J. Zhang, Y. Lin, Y. Lyu, and G. Qu, "A puf-fsm binding scheme for fpga ip protection and pay-per-device licensing," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 6, pp. 1137–1150, 2015.

[23] M. T. Rahman, D. Forte, Q. Shi, G. K. Contreras, and M. Tehranipoor, "Csst: an efficient secure split-test for preventing ic piracy," in *2014 IEEE 23rd North Atlantic Test Workshop*, pp. 43–47, IEEE, 2014.

[24] D. B. Roy, S. Bhasin, I. Nikolić, and D. Mukhopadhyay, "Combining puf with rluts: A two-party pay-per-device ip licensing scheme on fpgas," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, p. 12, 2019.

[25] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 83–89, ACM, 2012.

[26] A. Sengupta, S. Bhadauria, and S. P. Mohanty, "Tl-hls: methodology for low cost hardware trojan security aware scheduling with optimal loop unrolling factor during high level synthesis," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 36, no. 4, pp. 655–668, 2016.

[27] Z. Zhang, Q. Yu, L. Njilla, and C. Kamhoua, "Fpga-oriented moving target defense against security threats from malicious fpga tools," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 163–166, IEEE, 2018.

[28] T. Zhang, J. Wang, S. Guo, and Z. Chen, "A comprehensive fpga reverse engineering tool-chain: From bitstream to rtl code," *IEEE Access*, vol. 7, pp. 38379–38389, 2019.

[29] F. Benz, A. Seffrin, and S. A. Huss, "Bil: A tool-chain for bitstream reverse-engineering," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 735–738, IEEE, 2012.

[30] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware trojan insertion by direct modification of fpga configuration bitstream," *IEEE Design & Test*, vol. 30, no. 2, pp. 45–54, 2013.

[31] P. Swierczynski, G. T. Becker, A. Moradi, and C. Paar, "Bitstream fault injections (bifi)–automated fault attacks against sram-based fpgas," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 348–360, 2017.

[32] M. Ender, P. Swierczynski, S. Wallat, M. Wilhelm, P. M. Knopp, and C. Paar, "Insights into the mind of a trojan designer: the challenge to integrate a trojan into the bitstream," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 112–119, ACM, 2019.

[33] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *Proceedings VLSI (ISVLSI) 2018 IEEE Computer Society Annual Symposium on. EH-2001*, pp. 405–410, IEEE, 2018.

[34] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, and S. Bhunia, "Robust bitstream protection in fpga-based systems through low-overhead obfuscation," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–8, IEEE, 2016.

[35] M. Schmid, D. Ziener, and J. Teich, "Netlist-level ip protection by watermarking for lut-based fpgas," in *2008 International Conference on Field-Programmable Technology*, pp. 209–216, IEEE, 2008.

[36] Microsemi, "User guide polarfire fpga security." Microsemi, User Guide UG07532, 2018.

[37] E. Peterson, "Xapp1323 (v1.1): Developing tamper-resistant designs with zynq ultrascale+ devices," 2018.

[38] A. Moradi, A. Barenghi, T. Kasper, and 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*, pp. 111–124, ACM, 2011.

[39] E. De Mulder, P. Buysschaert, S. Ors, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede, "Electromagnetic analysis attack on an fpga implementation of an elliptic curve cryptosystem," in *EUROCON 2005-The International Conference on" Computer as a Tool"*, vol. 2, pp. 1879–1882, IEEE, 2005.

[40] S. Tajik, H. Lohrke, J.-P. Seifert, and C. Boit, "On the power of optical contactless probing: Attacking bitstream encryption of fpgas," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1661–1674, ACM, 2017.

[41] H. Shen, N. Asadizanjani, M. Tehranipoor, and D. Forte, "Nanopyramid: An optical scrambler against backside probing attacks," in *ISTFA 2018: Proceedings from the 44th International Symposium for Testing and Failure Analysis*, p. 280, ASM International, 2018.

[42] Xilinx, "Quickboot method for fpga design remote update." Xilinx, Appl. Note XAPP1081, 2014.

[43] J. Vliegen, N. Mentens, and I. Verbauwhede, "Secure, remote, dynamic reconfiguration of fpgas," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 4, p. 35, 2015.

[44] S. Drimer and M. G. Kuhn, "A protocol for secure remote updates of fpga configurations," in *International Workshop on Applied Reconfigurable Computing*, pp. 50–61, Springer, 2009.

[45] E. Peterson, "Wp468 (v1.0): Leveraging asymmetric authentication to enhance security-critical applications using zynq-7000 all programmable socs," *Retrieved October*, 2015.

[46] H. Li, G. Du, C. Shao, L. Dai, G. Xu, and J. Guo, "Heavy-ion microbeam fault injection into sram-based fpga implementations of cryptographic circuits," *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, pp. 1341–1348, 2015.

[47] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, and P. Alfke, "The rosetta experiment: atmospheric soft error rate testing in differing technology fpgas," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 317–328, 2005.

[48] V. Pouget, A. Douin, G. Foucard, P. Peronnard, D. Lewis, P. Fouillat, and R. Velazco, "Dynamic testing of an sram-based fpga by time-resolved laser fault injection," in *2008 14th IEEE International On-Line Testing Symposium*, pp. 295–301, IEEE, 2008.

[49] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "Fpga partial reconfiguration via configuration scrubbing," in *2009 International Conference on Field Programmable Logic and Applications*, pp. 99–104, IEEE, 2009.

[50] T. Güneysu, I. Markov, and A. Weimerskirch, "Securely sealing multi-fpga systems," in *International Symposium on Applied Reconfigurable Computing*, pp. 276–289, Springer, 2012.

[51] D. Owen Jr, D. Heeger, C. Chan, W. Che, F. Saqib, M. Areno, and J. Plusquellic, "An autonomous, self-authenticating, and self-contained secure boot process for field-programmable gate arrays," *Cryptography*, vol. 2, no. 3, p. 15, 2018.

[52] D. Pellerin, "Announcing amazon ec2 fi instances with custom fpgas." "https://www.slideshare.netlAmazonWebServices/announcing-amazon-ec2-fl-instances-with-custom-fpgas, retrieved,April13,2017".

[53] S. Trimberger and S. McNeil, "Security of fpgas in data centers," in *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, pp. 117–122, IEEE, 2017.

[54] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on fpgas," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1111–1116, IEEE, 2018.

[55] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 229–244, IEEE, 2018.

[56] Xilinx, "Security monitor ip core product brief." Xilinx, Product Brief, 2015.

[57] D. Akhoundov, "2017 erai reported parts analysis." "http://www.erai.com/ERAI_Blog/3139/Damir_Akhoundov_2017_ERAI_Reported_Parts_Analysis".

[58] Xilinx, "Ultrascale architecture and product data sheet: Overview." Xilinx, Datasheet DS890 (v3.10), 2019.

[59] Xilinx, "Xq ultrascale architecture data sheet: Overview." Xilinx, Datasheet DS895 (v2.0), 2018.

[60] M. M. Alam, M. Tehranipoor, and D. Forte, "Recycled fpga detection using exhaustive lut path delay characterization," in *2016 IEEE International test conference (ITC)*, pp. 1–10, IEEE, 2016.

[61] M. M. Tehranipoor, U. Guin, and D. Forte, "Counterfeit integrated circuits," in *Counterfeit Integrated Circuits*, pp. 15–36, Springer, 2015.