On the Malicious Potential of Xilinx' Internal Configuration Access Port (ICAP)

NILS ALBARTUS*, MAIK ENDER*, JAN-NIKLAS MÖLLER*, MARC FYRBIAK*, and CHRISTOF PAAR*, Max Planck Institute for Security and Privacy, Germany
RUSSELL TESSIER†, University of Massachusetts Amherst, USA

FPGAs have become increasingly popular in computing platforms. With recent advances in bitstream format reverse engineering, the scientific community has widely explored static FPGA security threats. For example, it is now possible to convert a bitstream to a netlist, revealing design information, and apply modifications to the static bitstream based on this knowledge. However, a systematic study of the influence of the bitstream format understanding in regards to the security aspects of the dynamic configuration process, particularly for Xilinx's Internal Configuration Access Port (ICAP), is lacking. This paper fills this gap by comprehensively analyzing the security implications of ICAP interfaces, which primarily support dynamic partial reconfiguration. We delve into the Xilinx bitstream file format, identify misconceptions in official documentation, and propose novel configuration (attack) primitives based on dynamic reconfiguration, i.e., create/read/update/delete circuits in the FPGA, without requiring pre-definition during the design phase. Our primitives are consolidated in a novel *Stealthy Reconfigurable Adaptive Trojan* (STRAT) framework to conceal Trojans and evade state-of-the-art netlist reverse engineering methods. As FPGAs become integral to modern cloud computing, this research presents crucial insights on potential security risks, including the possibility of a malicious tenant or provider altering or spying on another tenant's configuration undetected.

Additional Key Words and Phrases: FPGA Security, Hardware Security, Hardware Trojans, Bitstream Reverse Engineering, ICAP

ACM Reference Format:
Nils Albartus, Maik Ender, Jan-Niklas Möller, Marc Fyrbiak, Christof Paar, and Russell Tessier. 2023. On the Malicious Potential of Xilinx' Internal Configuration Access Port (ICAP). *ACM Trans. Reconfig. Technol. Syst.* 1, 1, Article 1 (January 2023), 29 pages. https://doi.org/10.1145/3633204

1 INTRODUCTION

Field Programmable Gate Arrays (FPGAs) serve as key implementation media for digital circuits [11] and are employed in a wide range of applications from consumer electronics (e.g., the iPhone [73] and virtual reality headsets such as the HTC Vive [19]) to high-performance cloud computing systems (e.g., Amazon EC2 F1 instances [5]). Although FPGAs are less power-efficient and have slower performance than Application Specific Integrated Circuits (ASICs), their field-programmable nature offers flexibility and adaptability, even during runtime.

From a security perspective, such dynamic reconfiguration capabilities, also known as runtime update support, present a compelling attack vector. This issue has become particularly acute recently

Authors' addresses: Nils Albartus, nils.albartus@mpi-sp.org; Maik Ender, maik.ender@mpi-sp.org; Jan-Niklas Möller, jan-niklas.mÃűller@mpi-sp.org; Marc Fyrbiak, marc.fyrbiak@mpi-sp.org; Christof Paar, christof.paar@mpi-sp.org, Max Planck Institute for Security and Privacy, Universitätsstr. 140, Bochum, NRW, Germany, 44801; Russell Tessier, tessier@umass.edu, University of Massachusetts Amherst, 130 Natural Resources Road, Amherst, MA, USA, 01003.

1936-7406/2023/1-ART1
https://doi.org/10.1145/3633204

since FPGA applications now often contain circuitry from multiple sources [63, 45], such as third-party intellectual property (IP) cores. Additionally, the globalized hardware design process often includes numerous untrusted stakeholders, which can challenge hardware design trustworthiness. Consequently, research has focused on the detection of malicious circuits, *a.k.a.* hardware Trojans, in FPGA bitstreams [42].

With advances in bitstream reverse engineering, the threat potential to FPGAs security has increased, in particular, to convert bitstreams to netlists, which allows for a deep understanding of designs. There currently exists several tools that allow for the conversion for device bitstreams out-of-the-box, particularly for contemporary Xilinx FPGAs. Popular projects, such as Project X-Ray and U-Ray [2, 3], have documented significant parts of Xilinx FPGA family bitstreams. In addition to the threat of reverse engineering, FPGAs are vulnerable to manipulations following design retrieval, cf. [69, 23, 36].

While the static attack vector of design modification has been explored, reverse engineering of the bitstream format also has consequences for dynamic (partial) reconfiguration. To the best of our knowledge, no systematic security analysis of the dynamic configuration process (with a focus on the capabilities of Xilinx Internal Configuration Access Port (ICAP) has been performed to date, despite the introduction of ICAP in the Virtex II architecture [UG002] in January 2001. We partially attribute this research gap to the low-level and complex nature of the reconfiguration process. In particular, such a security analysis requires in-depth comprehension of the FPGA bitstream file format, which has only matured in the past few years.

As partial reconfiguration is a critical factor for cloud FPGAs, the threat potential remains unclear. Modern cloud computing demonstrates FPGA multi-tenancy in a restricted form, i.e., numerous cloud vendors support FPGA computation and couple user logic with a predefined *shell* that is responsible for bus interfacing and dynamic reconfiguration activities [5]. Given that the shell functionality is not fully transparent to users, it poses a security risk in the context of potentially malicious cloud providers. Lastly, the concurrent use of circuits from multiple independent paying customers within an FPGA has been proposed [37] and is anticipated to become available to cloud users in the near future. As demonstrated in this work, the exposure of internal configuration interfaces can have severe security implications.

Goals and Contributions. In this paper, we focus on the security of Xilinx dynamic partial reconfiguration ICAP interfaces. The capabilities and security implications of attacker-controlled access to these interfaces are comprehensively and systematically assessed. Using knowledge of the Xilinx bitstream file format and exploiting misconceptions and inconsistencies in official documentation, we investigate novel and generic (attack) primitives that use dynamic reconfiguration to update FPGA hardware design circuitry at runtime. The attack primitives are included in a framework called STRAT (Stealthy Reconfigurable Adaptive Trojan Framework) that automatically generates self-modifying hardware Trojans using dynamic partial reconfiguration. Finally, we demonstrate the security consequences of attacker-controlled dynamic partial reconfiguration, including the readback of a plaintext bitstream even if bitstream encryption is deployed and the snooping of cryptographic key material in multi-tenant FPGA applications.

In summary, our contributions are:

- **In-Depth Security Analysis of Xilinx ICAP Interface.** We contribute to the understanding of FPGA security by performing a thorough analysis of dynamic partial reconfiguration using ICAP. Novel (attack) primitives based on dynamic reconfiguration that create, read, update, and delete arbitrary portions of an FPGA hardware design at runtime are implemented using these insights.

- **STRAT: Stealthy Reconfigurable Adaptive Trojan Framework.** As malicious designers face the threats of detection by state-of-the-art reverse engineering methods, we present design and implementation of STRAT, a framework to automatically deploy and remove hardware Trojans using our novel dynamic reconfiguration primitives, to conceal FPGA Trojans. By leveraging information about the bitstream file format, our generated hardware Trojans are precise and minimally-invasive, i.e., enabling targeted modifications down to single Look-Up Tables (LUTs), Flip-Flops (FFs), or wires. To further increase the stealthiness of the dynamic approach, STRAT incorporates a novel adaptive readout detection and response mechanism to automatically identify and remove Trojans before external bitstream readback is completed.
- **Security Implications for FPGAs in the Cloud.** The security implications of our novel primitives and framework are examined within the context of real-world FPGA applications. We explore the ramifications for FPGAs in cloud environments, in which a malicious tenant (or malicious cloud provider) can spy on and/or modify a victim tenant's configuration without detection.

2 TECHNICAL BACKGROUND

This background section includes information about bitstreams and netlist reverse engineering techniques.

2.1 Xilinx FPGAs

FPGAs in general are reconfigurable Integrated Circuits (ICs) that can be customized to implement a broad range of logic designs. FPGA logic includes Look-Up Tables (LUTs) which implement Boolean functions, Flip-Flops (FFs), wires, and specialized elements, such as multipliers and Block-RAMs (BRAMs). Throughout the paper, we use the term *fabric* to refer to the part of the FPGA that contains the user's design, while the *configuration engine* is the FPGA component that configures the fabric via internal or external configuration access ports. The following background focuses primarily on Xilinx 7-Series & UltraScale(+) FPGAs.

2.1.1 FPGA Configuration. FPGA configuration is performed by applying a bitstream – containing the FPGA's design – to one of the configuration ports. Deployed in-field the bitstream is usually stored on an external non-volatile memory. The bitstream itself is a specific command sequence of 32-bit words, which are read and write commands to registers in the FPGA's configuration engine. The bitstream can be used via both internal and external ports, i.e., JTAG, SPI, ICAP. The main part of the bitstream configuration process is writing the fabric configuration data to the Frame Data Register (FDRI), configuring the basic elements of the FPGA, i.e., LUTs, FFs, and routing. The fabric is programmed in frames, which consist of 93 (UltraScale+), 101 (7-Series), or 123 (UltraScale) words. Each frame within the fabric can be addressed individually by a frame address. Upon configuration, this address must be provided in the Frame Address Register (FAR). Figure 1 shows a general system overview of the FPGA configuration engine [UG570, UG470].

2.1.2 Bitstream Format. The Xilinx bitstream format follows a predefined structure, which is documented in user guides [UG470, UG570]. Before the configuration engine retrieves commands, the sync word (0xAA995566) must be sent, signaling the beginning of the bitstream. The main part of the bitstream consists of type 1 packets, indicating a read/write to a certain register, including the number of target words, and type 2 packets that optionally define larger word counts. This information is followed by the data associated with the register. As described in the previous section, fabric configuration occurs via writes to the FDRI register. It is important to note that a bitstream does not necessarily only write information that alters fabric configuration. A bitstream

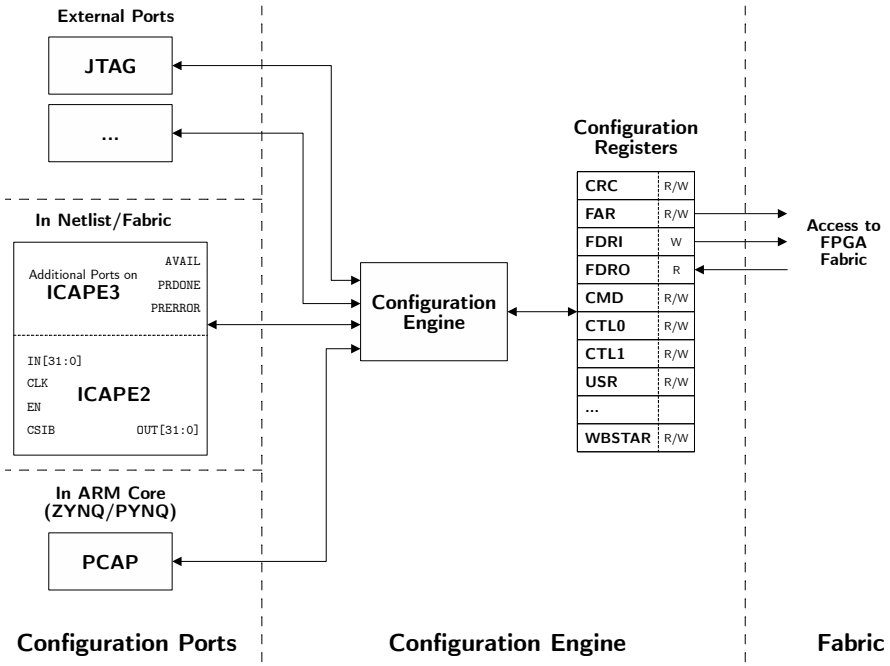


Fig. 1. Simplified FPGA Configuration Overview.

typically contains a sequence of commands that communicates with multiple configuration registers. Effectively, a bitstream defines a communication protocol, e.g., a bitstream can consist of only commands to read back a configuration status register rather than only including logic-altering information. While the general structure of the bitstream is vendor documented, the content of the configuration data used to configure the fabric is proprietary. This information for the 7-Series [3] and partially for UltraScale [2] FPGAs have been reverse engineered.

2.1.3 Bitstream Protection Schemes. Since hardware is often used as the root of trust within an embedded system, safeguarding it is naturally essential. In the case of FPGAs, the executed design is usually stored in external non-volatile memory. Therefore, securing the bitstream takes a central role in the security design of FPGAs. FPGA manufacturers have implemented bitstream protection systems to ensure (1) the integrity and authenticity as well as (2) the confidentiality of the bitstream. Several attacks against bitstream protection schemes have been proposed based on implementation/protocol flaws and hardware limitations (e.g., side-channel attacks). Most recently, Ender et al. broke Xilinx 7-Series FPGA's bitstream protection [22] and demonstrated attacks on the UltraScale(+) bitstream protection[21], which, however, can still be considered secure if correctly configured. Bitstream encryption side-channel and probing attacks have been successfully carried out on (older) generation FPGAs [52, 53, 54, 55, 70, 71].

2.1.4 Partial Reconfiguration. Partial – or reconfiguration in general – is needed by many FPGA applications to provide bug fixes and real-time updates [12]. Recently, partial reconfiguration has been used to update machine learning (ML) parameters in FPGAs following training [47, 20]. Kernel parameters and ML interconnect can be updated to provide improved algorithm performance. A similar approach can be used to update encryption algorithms and keys in FPGAs [28, 31]. The

cryptographic keys can be directly embedded in FPGA logic to obscure their function. Additionally, partial reconfiguration is used in communication coding and network virtualization. For example, to tune communication transmitters and receivers to adapt to changing channel noise [68, 75]. Since FPGAs are a popular choice for network routing, given their parallelism and specialization, partial reconfiguration can be used to reduce router power consumption [56] and provide updated routing information [83]. Finally, signal processing filter coefficients can be changed to adapt to changing operating conditions [64, 16].

2.1.5 Internal Configuration Access Port (ICAP). The ICAP is an internal configuration access port primarily used for partial reconfiguration. It can be instantiated as part of a design like any other primitive located within the FPGA's fabric. Xilinx 7-Series devices employ the `ICAPE2` primitive, while UltraScale(+) devices utilize `ICAPE3`, which includes added status ports. Since ICAP is directly connected to the configuration engine, it is considered a *trusted* configuration port. Hence, netiher encryption nor authentication is needed via ICAP.

2.2 Netlist Reverse Engineering

In some attack scenarios, an FPGA netlist is analyzed to gain design insights or find spots for Trojan insertion. A bitstream is effectively an alternative format of an FPGA design netlist, and converting the bitstream to a human-readable netlist format is possible [3]. After bitstream conversion, a flattened netlist containing no high-level or hierarchical information [1, 7, 26] is obtained. This netlist contains only logic gates and their interconnection. Netlists can be analyzed with (1) static analysis methods, in which the netlist is analyzed without information collected during execution, or (2) dynamic analysis methods that include simulation or on-chip debugging [67].

2.3 Hardware Trojans

Since the report on high-performance microchip supply in 2005 [18], the scientific community has substantially researched offensive and defensive aspects of unauthorized malicious hardware manipulations *a.k.a.* hardware Trojans [26, 76]. Surreptitious manipulations can be inserted during various development phases (e.g., during the design phase by a malicious designer or during the globalized IC production phase) and comprise various trigger and payload designs. Numerous works focus on automated detection to counteract the risks of hardware Trojans, such as identifying trigger mechanisms or focusing on payload features [26]. Here, different static and dynamic analysis approaches have been proposed [34, 77, 82, 62, 15]. Powell et al. [58, 42] developed a framework that can statically detect pre-defined malicious circuitry in FPGA designs while other strategies [62] combine runtime detection strategies.

2.4 Related Work

In Bitman [57] and Byteman [46], the authors determined general knowledge of the FPGA bitstream format to allow them to reallocate, merge and modify regions using this information. The tools operate at the bitstream level. Upon receiving one or more input bitstream(s), different operations, such as merging or reallocating entire blocks, can be applied. Currently, the tools are not based on bitstream databases, so targeted modifications must be crafted by hand. Notably, these modifications are directly applied to the bitstream itself, prior to FPGA initialization. The work focuses on improving the tooling and offering faster workflows rather than on security implications. Stolz *et al.* explored using low-level bitstream features for design obfuscation [67]. For example, they check for targeted low-level routing manipulation via ICAP as a safeguarding mechanism within their obfuscation scheme.

Additional previous work, such as thangrycat [36], has focused on malicious static manipulations to the bitstream. Modifications made by thangrycat were applied to the bitstream itself in a static form by pre-startup bitstream exchange. Fyrbiak et. al. manipulated a LUT to leak a key of an FPGA used for encryption for a USB storage device [69]. Ender et. al. [23] provided insights on how to conduct static bitstream manipulation to bypass a self-check in a cryptographic core.

This work focuses on novel run-time manipulations, in conjunction with an understanding of the bitstream format. Potential modifications are carried out during FPGA operation.

3 THREAT MODEL

In this work, we assume an attacker with access to ICAP and in-depth knowledge about ICAP and the bitstream format. The high-level goal of the attacker is to undermine system security by performing illegitimate hardware access. More precisely, such accesses can be an illegitimate read operation (e.g., to spy out data that the attacker should not have access to) or an illegitimate write operation (e.g., to perform malicious hardware manipulation).

We want to emphasize that our threat model is consistent with prior research on hardware security [67, 35, 72, 40, 41] as detailed in the following:

(1) **Malicious Designers.** A malicious designer has the goal to conceal a hardware Trojan by exploiting partial reconfiguration. Since partial reconfiguration features can be included during design phase, the malicious designer has the capability to perform illegitimate spy or manipulation actions. Note that we will demonstrate a substantial toolkit with `STRAT` to insert and remove a Trojan, that will remain undetected with modern analysis techniques (Section 5). Moreover we want to emphasize that the attacker class of malicious designers can be categorized as follows:
   - **Malicious Designers / Integrator** carefully craft a hardware design with the explicit intent of inserting malicious elements such as a Trojan. To this end, they leverage their system knowledge with the goal to embed harmful components in a covert way to render detection as challenging as possible.
   - **Malicious 3rd-Party IP Cores Designers** builds a malevolent Intellectual Property (IP) core to introduce a Trojan (e.g., an exploit or backdoor) into the overall system, where the IP core is integrated. Similarly to the malicious designer / integrator, the goal is to embed the Trojan in a covert way.
   - **Malicious EDA Tools** refers to manipulated or compromised EDA tools to introduce harmful modifications during design (e.g., a malicious synthesizer adds a Trojan to the design). Similarly to the other malicious designers, the goal is to embed the Trojan in a covert way as well.

(2) **(Multi-Tenant) Cloud FPGAs** become increasingly popular, however, dynamic reconfiguration features / ICAP challenge trust between cloud providers and users as outlined in the following threat scenarios:
   - **Attack towards Customers** can be performed by a malicious tenant or malicious cloud provider by use of partial reconfiguration / ICAP to spy or manipulate victim tenant hardware with the aforementioned security implications.
   - **Attack towards Cloud Provider** can be performed by a malicious customer who leverages partial reconfiguration / ICAP to spy or manipulate hardware of the cloud provider shell (e.g., to overcome security access restrictions).

We want to highlight that numerous designs use ICAP and partial reconfiguration for legitimate reasons (cf. Section 2.1.4), thus usage itself is not suspicious for a security analyst. However, we

demonstrate that designs with ICAP are inherently hard to trust, as they pose potential to disguise malicious intent.

4 SYSTEMATIC ANALYSIS OF (NOVEL) RECONFIGURATION CAPABILITIES

We now provide details on the real reconfiguration capabilities by providing an extension to the vendor-intended reconfiguration flow (Section 4.1). Moreover, we introduce our setup and ICAP prototyping framework that supported our ICAP exploration in a semi-automated way (Section 4.2).

4.1 Extending the Vendor-Intended Reconfiguration Flow

Modern FPGAs provide partial dynamic reconfiguration capabilities, enabling the exchange of components within a programmed device. Support for this functionality is typically facilitated by the vendor's computer-aided design (CAD) tool flow. Designers are required to define logic partitions that are targeted to specific regions (*pBlocks*) of the FPGA chip. This information is provided to the tool flow. Partitions may occupy the same region at different times (Figure 2).

A full bitstream containing the static partition (not changed after initial configuration) and partial bitstreams for the dynamic partition are created.

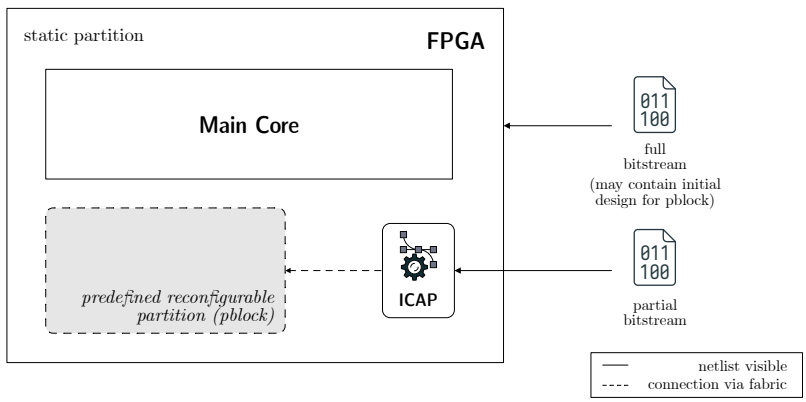
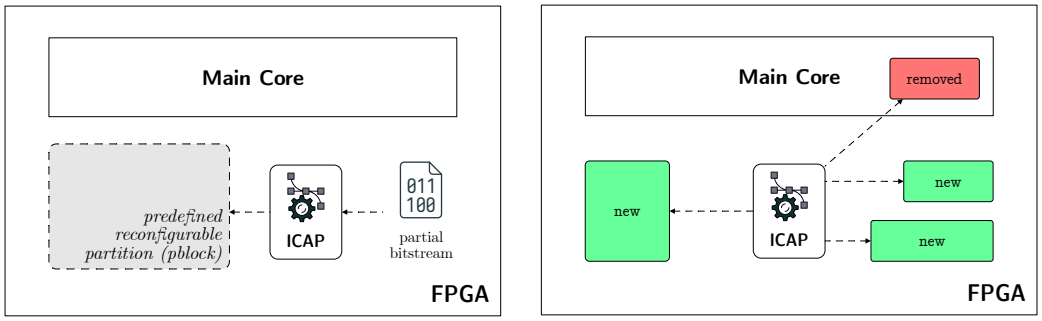


Fig. 2. General Partial Dynamic Reconfiguration Flow. The FPGA is programmed with the full bitstream, which may contain an initial design for the predefined dynamically reconfigurable region. The partial bitstream is supplied to the ICAP controller, which subsequently configures the partial bitstream via the internal fabric configuration access.

If the bitstream format and frame content are understood, the standard vendor tool flow can be extended. Since a bitstream consists of a series of commands, arbitrary command sequences to read or write specific sections of the FPGA fabric can be created. This capability can be used to derive novel primitives that are unsupported by the vendor flow and may be exploitable for malicious purposes. In the following, we introduce novel configuration options.

4.1.1 Create / Delete. By using partial dynamic reconfiguration and bitstream format knowledge, circuits that were not part of the initial design can be created and placed anywhere in the FPGA, augmenting design functionality. Likewise, existing circuitry can be removed even if it was not defined as partially reconfigurable in the initial design flow. In Xilinx's standard Vivado tool flow, only a predefined `pBlock` can be removed by creating and then loading an empty design for a partition.



(a) Intended behavior by the vendor. Only a predefined *pBlock* can be created/updated using a partial bitstream. An empty design can be created and deployed to delete the circuit in a *pBlock*.

(b) Modified behavior with bitstream format understanding. Circuits the FPGA can be deployed or removed anywhere on the FPGA.

Fig. 3. Partial reconfiguration supported by vendor tools versus the use of new primitives fashioned using full bitstream knowledge

4.1.2 (Atomic) Modify. Xilinx support for partial reconfiguration limits configuration writes to predefined *pBlocks*. However, with our approach, *atomic* modifications on single elements can be performed. As a result, every FPGA basic element can be reconfigured, including the following examples.

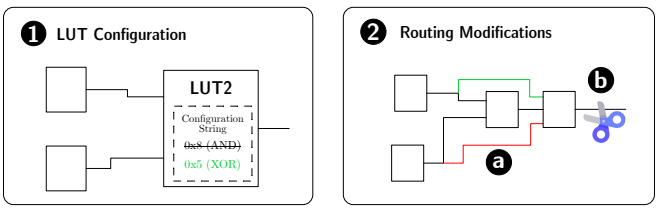


Fig. 4. Capabilities of the *modify* primitive

❶ **LUT Configuration.** Manipulates a single LUT configuration to alter the Boolean function without affecting the surrounding circuitry.

❷ **Routing Changes.** Changes can include ⓐ modifying an input wire or rerouting a wire completely or ⓑ *cutting* connections at a route Programmable Interconnect Point (PIP), causing a signal *output* to take on a logic 1 value as a consequence. This approach can be used to patch certain values to always output a 1 without modifying the combinational circuit.

Conducting Atomic Modifications. The smallest entity that can be replaced in an FPGA is a frame (see Section 2.1.2). However, a single frame may contain more than just a single LUT or PIP for routing, which means that replacing a frame might inadvertently alter unintended circuitry. To address this issue, one must first read the current frame content, apply the desired modifications, and then write the modified frame back. This approach ensures that no unwanted changes occur during the manipulation process.

Register Content Manipulation. While it is possible to perform targeted register value manipulation, this is not feasible in practice. To change the value, the `FF.INIT` value has to be changed in the bitstream. Normally this value is loaded when the FPGA is configured. However, a reset of the associated global set reset (GSR) can also reset the FFs to their initial values. Thus all elements in a region affected by the clock would be reset to their initial values, potentially destroying the current state. Thus, to the best of our knowledge, a targeted change to one FF is impractical.

4.1.3 (Atomic) Read. Access to the configuration port not only allows for the write of configuration but also a readout. Xilinx offers this feature to verify a current configuration. Here, we are also not restricted to any boundaries allowing us to perform targeted readouts.

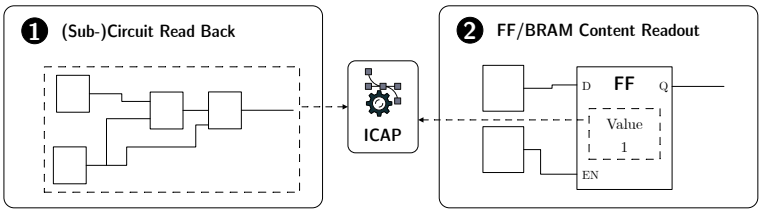


Fig. 5. Capabilities of the *read* primitive

1. **(Sub-)Circuit Read Back.** By reading specific frames, a subcircuit or even the entire design can be read back. The readback circuits can then be converted to a netlist for further analysis.
2. **Register/BRAM Content Readout.** It is possible to read back current FF values or BRAM state. This approach enables register access without a visible connection in the netlist. The live readback is enabled by the capture feature offered by Xilinx, as described below.

Readback of Storage Elements. Xilinx devices provide a *configuration readback capture* feature, enabling live readout of internal Configurable Logic Block (CLB) registers, including FFs and BRAMs [XAPP1230]. For 7-Series devices, the `CAPTUREE2` primitive and/or the `GCAPTURE` command are required, while for UltraScale(+) devices, only the enabling of the capture bit in the `CTL1` register is necessary. The bit location in the fabric can be determined in two ways: (1) by matching the value of the bit to the `FF.INIT` value in the bitstream database used for FF initialization, or (2) by generating a logic localization file using vendor tools to define the exact position of the bit. The latter method can be employed to create a database for devices without an existing bitstream database by placing all FFs and retrieving their positions from the logic localization file. However, for 7-Series devices, activating capture mode overwrites the `FF.INIT` value, causing potential unintended states when a fabric reset/restore (GSR) is triggered for the region. UltraScale(+) devices do not face this issue, as the original `FF.INIT` value is restored after disabling capture mode.

4.2 ICAP Prototyping Framework

We designed and implemented an ICAP prototyping framework to support our experiments. The framework consists of two parts: (1) the hardware design, incorporating the ICAP to send commands to it; and (2) the software that communicates via Universal Asynchronous Receiver Transmitter (UART) with our controller. Our framework is able to receive bitstreams, i.e., a sequence of commands[1] and send them to ICAP. The hardware sets the correct sequence for the ICAP control

[1]Note that *bitstream* refers to the command sequence and does not necessarily mean a write to the fabric itself to configure a design (see Section 2.1.2)

signals, enabling write and read functionality. Thus the framework allows for tests ranging from reads from and writes to all command registers to writes to all parts of the fabric.

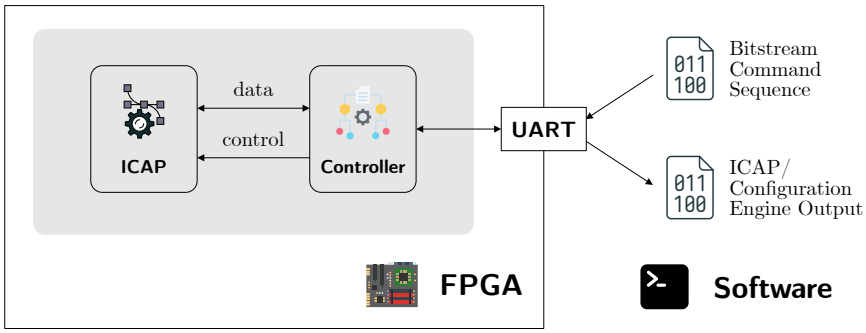


Fig. 6. Overview of ICAP Prototyping Framework.

5 STRAT: STEALTHY RECONFIGURABLE ADAPTIVE TROJAN FRAMEWORK

We now describe the design and implementation of our stealthy reconfigurable adaptive Trojan framework STRAT leveraging the novel primitives from Section 4. First, we motivate and introduce the high-level idea and then the architecture and workflow of STRAT are detailed. We conclude with a Trojan case study, where the Trojan is deployed and removed with STRAT, and a discussion.

5.1 Motivation

With recent advances in reverse engineering, a Trojan designer faces challenges that could lead to Trojan detection. In the following we discuss modern reverse engineering techniques and introduce our high-level idea to counter state-of-the-art detection methods furthering the need for improved detection.

5.1.1 The Threat of Netlist Reverse Engineering from a Malicious Designer Perspective. From the perspective of a Trojan designer, advances in netlist reverse engineering are worrisome, as they allow Trojans to be identified more easily.

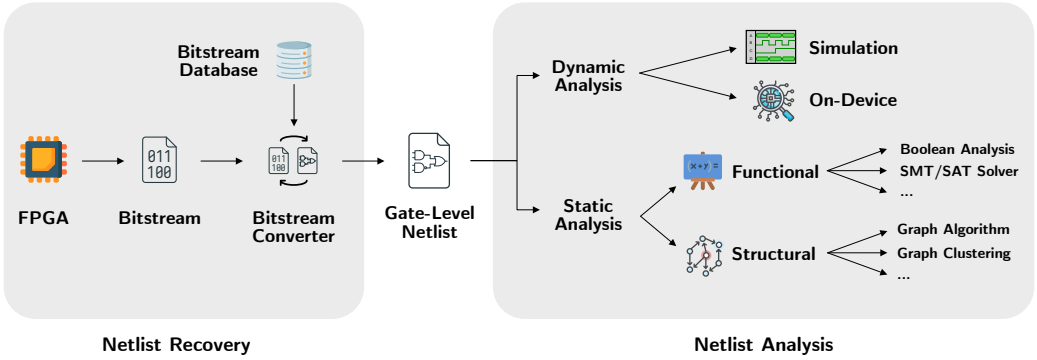


Fig. 7. Netlist Reverse Engineering Overview.

With bitstream databases readily available for many FPGA devices, FPGA netlist reverse engineering can be conducted without the need for expensive equipment. An attacker must extract the netlist, which is often stored in external non-volatile memory, and convert it to a netlist. For ASICs, expensive equipment, extensive know-how from various research areas, and significant time are necessary to achieve netlist extraction. Thus we argue that ASIC reverse engineering is more possible for nation state actors or parties with extensive resources, while FPGA reverse engineering can generally be conducted in a *basic research setting*. Generally, an FPGA Trojan designer must consider netlist analysis an acute and realistic threat, which needs to be taken into account when designing a Trojan, and a countermeasure developer must be able to use it to their advantage.

Once the netlist has been recovered, an attacker can deploy various static and dynamic analysis techniques to analyze the *sea-of-gates*. A popular framework to support these analysis efforts is *HAL - The Hardware Analyzer*, which is available open-source [24, 32]. Generally, we distinguish between dynamic and static analysis methods [67].

Static Analysis. Static analysis facilitates an examination of design functionality at the netlist level, without necessitating design execution. Here, the community often distinguishes between *functional* and *structural* analysis methods [6, 7]. In structural analysis, the netlist can be interpreted as a directed graph, thus allowing the application of graph theory. This approach enables the reverse engineer to leverage detection algorithms to conduct tasks such as identifying the control path [13, 14, 25, 48, 50, 49] or performing data-path analysis [1, 43]. Most recently the use of Graph Neural Networks (GNNs) have gained attention as their applicability to netlist reverse engineering has yielded promising results [4]. In functional analysis, the underlying logic function is analyzed [7]. Here, tools like Satisfiability Modulo Theories (SMT) solvers often find usage, e.g., to match subcircuits against predefined models [27, 43].

Dynamic Analysis. Dynamic analysis enables a reverse engineer to monitor temporal chip behavior. This category of analysis encompasses various types, including simulation-based and on-chip methods [67]. Specifically, within the context of FPGAs, the Joint Test Action Group (JTAG) readout ports could be utilized to extract the current bitstream and ascertain the device's state.

5.1.2 High-level Idea. Since netlist reverse engineering can stymie Trojan insertion, an attacker must hinder static and dynamic analysis methods as much as possible. Given advancements in FPGA analysis and reverse engineering, Trojans must be concealed from techniques such as graph-based, simulation-based, and on-device analysis.

Software attackers often deploy self-modifying code in malicious software to counter state-of-the-art detection methods [74, 9]. We adapt this principle to FPGA deployment by exploiting our novel reconfiguration primitives to counteract static detection strategies. The primitives are incorporated within an automated tool flow to effectively build dynamic hardware Trojan triggers and payloads and thereby circumvent state-of-the-art methods that detect static malicious circuitry [42]. Precise and minimally invasive Trojans are developed using bitstream knowledge, facilitating tailored modifications (e.g., for single LUTs, FFs, or wires) using a custom Trojan-specific bitstream file format. A novel adaptive readout detection and response mechanism is also designed. Thus, our mechanism detects external bitstream readbacks and adaptively responds by removing the Trojan before the readout is completed.

5.2 STRAT - Design

We now detail the design of STRAT and its automated Trojan generation workflow from an attacker perspective. STRAT is build around the *Project X-Ray* database, thus supporting 7-Series devices [3].

Building Blocks. STRAT consists of 3 building blocks: (1) a software part to analyze the bitstream and generate a trojanized version (Section 5.2.1), (2) a hardware runtime fabric manipulation engine to dynamically reconfigure the Trojan into the design (Section 5.2.2), and (3) a hardware readout detection and response engine to detect external readbacks and respond with dynamic removal of the Trojan before the readback occurs (Section 5.2.3). An overview of STRAT's hardware components is shown in Figure 8.

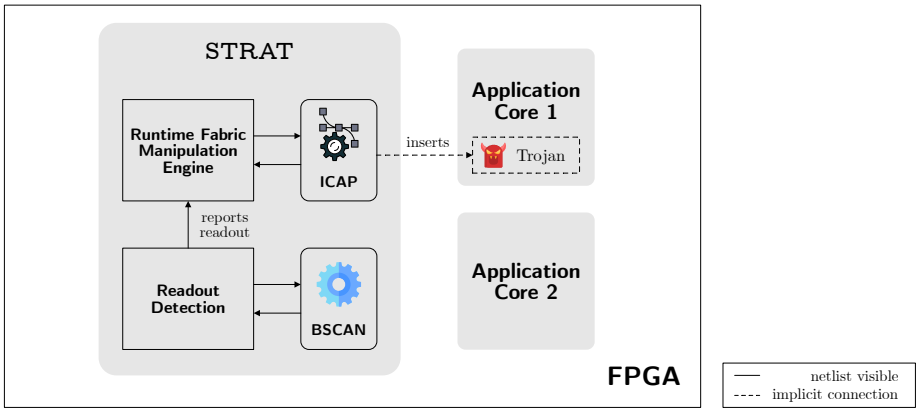


Fig. 8. High-Level Overview of STRAT.

Workflow. We first detail the workflow from a malicious designer's perspective. For simplicity, we assume that the Trojan functionality is given at the netlist level, i.e., the malicious designer knows which gates and wires are targeted, see Section 5.3 for malicious designer case studies. Operation steps are as follows: (1) the malicious designer uses the bitstream analysis and delta generation software together with the current bitstream and Trojans FPGA Assembly (FASM) description to generate our so-called *configuration delta memory* file, i.e., the changes that are required to realize the Trojan functionality in the current bitstream, cf. Figure 9. The delta memory file is then patched into the memory of the runtime fabric manipulation engine by updating the bitstream via the Xilinx memory update tool. (2) Once the runtime fabric manipulation engine receives a signal to perform the dynamic reconfiguration to load the Trojan, it reads back the associated frames in the device via ICAP. It stores them in temporary memory (BRAM). (3) The configuration delta is applied to the retrieved frames during readback, and only specific words are changed. (4) The ICAP is then used to configure the malicious changes in the FPGA by writing back the manipulated frames from the temporary memory.

Suppose the readout detection and response engine detects a readout attempt by the victim at runtime. In that case, the runtime fabric manipulation engine is triggered to revert the changes and re-generate the original configuration data that does not include the Trojan. We want to emphasize that we store the configuration delta as XOR bitmask to ease the reversion as an involution and simply re-use the same bitmasks for both adding and removing the Trojan, as in more detail discussed in Section 5.2.2.

5.2.1 Bitstream Analysis and Delta Generation Software. To generate a Trojan bitstream from a FASM description and subsequently embed the Trojan into a given bitstream, we leverage Project X-Ray [3] bitstream databases. Based on the bit differences between the original and malicious

bitstreams, we generate the delta configuration as lists of triplets of (1) the configuration frame address, (2) the word address in the configuration frame, and (3) the delta data. Finally, the software outputs an updated memory file, i.e., for the memory of the runtime fabric manipulation engine. This output is then passed to the Xilinx memory update tool to generate the malicious bitstream containing the Trojan.

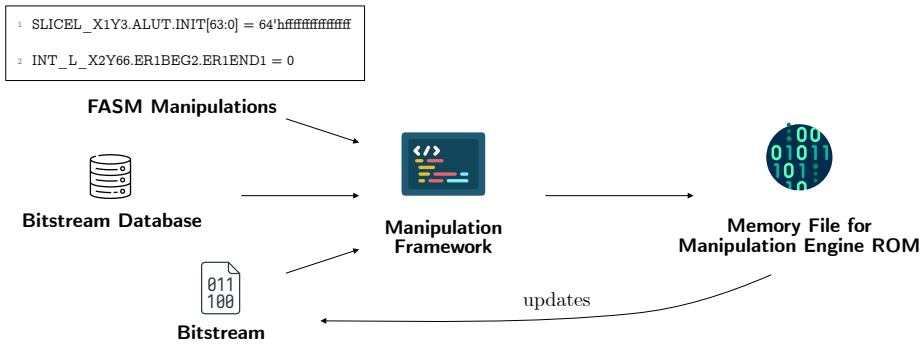


Fig. 9. Workflow for Generating Trojanized Designs with STRAT.

Implementation. Our bitstream analysis and delta generation is written in C++ and can parse Xilinx 7-Series bitstream databases provided by Project X-Ray [3]. The software can handle manipulations for incomplete databases since only the manipulations have to be covered by the bitstream database, i.e., bitstream modifications are inserted at known points, and the rest is left untouched. In addition to memory file generation, the tool can be used to insert modifications into the bitstream directly.

5.2.2 Runtime Fabric Manipulation Engine. We leverage our runtime fabric manipulation engine to apply the insertion and removal of the Trojan functionality from the bitstream configuration at runtime. The operational heart of this engine is a state machine that handles communication with ICAP and a memory block that stores the configuration delta. First, the engine reads the configuration frame via ICAP, i.e., identified by the first entry of a configuration delta triplet. Second, the engine then applies the configuration delta data in the target word, i.e., identified by the second and third entry of a configuration delta triplet. Finally, the engine then writes back the configuration frame via ICAP. As noted before, we store the Trojan configuration delta data as an XOR bitmask, so the same circuitry is used to add and remove the Trojan from the design.

Hardware Implementation. The runtime fabric manipulation engine is implemented using a state machine with 12 states, 1 BRAM used for temporary frame storage, and 1 Read-Only Memory (ROM) containing the bitstream delta. In total, the engine, including the ICAP controller, uses 286 LUTs, 245 FFs and 2 BRAMs (one configured as ROM).

5.2.3 Readout Detection and Response Engine. Even though dynamic reconfiguration hardware Trojans evade static Trojan detection strategies by design, readout of an FPGA configuration at runtime via external ports would reveal its presence, e.g., by applying static Trojan detection strategies on the readout configuration or even simply comparing the readout configuration to the initial bitstream configuration. To this end, we design a readout detection and response engine to detect such readout attempts and remove the Trojan before the readout is completed. Since Xilinx

FPGAs provide readout capabilities using JTAG, we now first briefly present JTAG to understand the mechanics of the inner working of our readout detection and response engine.

Readout Detection. JTAG is a typical debug interface and Xilinx provides the so-called BSCANE2 primitive to access JTAG chain from within the FPGA design. Based on the design of JTAG chains, all clock and data signals are shared between the external interface and all internal BSCANE2 primitives. If any activity is sensed on the shared clock or data signals, it implies that the JTAG interface is in use and thus serves as a straightforward readout detection, cf. [XAPP1084, XAPP1098]

Response. Once we detect a readout via JTAG, we aim to remove the Trojan from the current configuration immediately. However, once a JTAG readout occurs, access to all other configuration ports is locked, including the ICAP used to remove the Trojan. Note that this challenges the *short* time frame from detection that a readout occurs to the response to remove the Trojan using ICAP. In our experiments, the time until the ICAP is locked took 5.350 ms on average (min: 0.348 ms, max: 9.216 ms) for a Basys3 board (Artix-7, FTDI USB to serial) using OpenOCD 0.11 running on a Linux PC (6.1.23-1-MANJARO). For example, since our Trojan from the case study in Section 5.3 is removed within 0.194 ms, thus the response time is sufficient to remove the Trojan before the ICAP is locked.

Our controller which is optimized for area and not speed, needs $522 + 3 \times n$ cycles to manipulate a single frame (with n = number of manipulation within the frame). Considering the minimum time frame of 0.348 ms it is possible to change up to 63 frames with 10 manipulation in each frame and a 100 MhZ clock. However, the actual needed time highly depends on the location of the conducted manipulations, e.g., its placement relative to each other, and thus the placement within the frames. Note as well that for example one LUT configuration string is spread into multiple different frames. An attacker can optimize the manipulation process, by manually placing the LUTs and other desired modifications in the same frames, thus reducing the overall amount of frames being modified.

We want to note that a noise-less JTAG query system (e.g., using another FPGA to communicate with JTAG) may significantly reduce the readout time. Thus we examined an alternative strategy to increase the response time for our Trojan removal. Since the BSCANE2 primitive configuration can disable the JTAG altogether, we can simply disable external JTAG access. Once a readout attempt via JTAG occurs, the external interface will generate an error as JTAG is disabled. Meanwhile, we remove the Trojan and restore JTAG access, so any further JTAG debug attempt will succeed and find the configuration without the Trojan.

Hardware Implementation. The implemented JTAG detection is using 1 LUT, 2 FFs, and 1 BSCANE2 primitive, since a connection to the BSCANE2 clock suffices to detect activity.

5.3 Case-Study: Insert Key-Leakage Trojan in AES with STRAT

In this scenario, we consider a malicious designer who employs our framework to execute their nefarious objectives covertly. The motivation behind this action could be to establish a backdoor, facilitating unauthorized access or control over the system. To evade detection, the Trojan must be expertly concealed. Consequently, we explore how a malicious designer can leverage STRAT to discreetly embed a Trojan within a design. The design uses the ICAP for legitimate purposes, it might be an advanced ML core, network equipment circuit, or soft error mitigation IP core that is part of Xilinx's IP core library. Hence the use of ICAP is not necessarily suspicious to the victim. Attackers can use our STRAT framework to insert hardware Trojans at runtime, thus evading advanced detection mechanisms. In the following, we conduct a case study which inserts a key-leakage Trojan into an AES core.

Attack Outline. In this attack we modify the SBox output of the round function to always output zero. The flow of the last AES round is depicted in Figure 10.

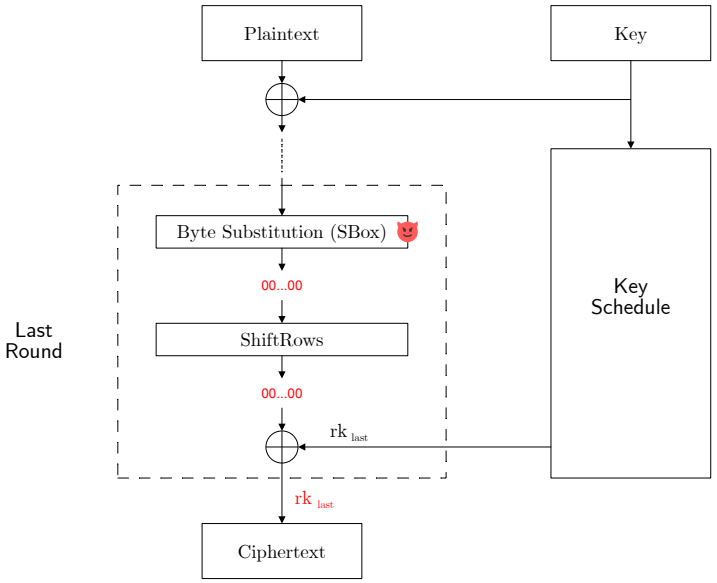


Fig. 10. Attack Outline Showing AES Key Leakage via SBox Manipulation.

With the modified SBox generating zeros as output, the result of the *Byte Substitution* layer is zero. In *ShiftRows* the positions of the bits are only shifted, thus the output is not changed. As a consequence, in the last round we are able to retrieve the last round key (rk) as the ciphertext. With the Trojan inside the ciphertext is computed as follows:

$$\begin{aligned}
\textsc{Ciphertext} &= rk_{last} \oplus \textsc{ShiftRows}_{lastRound} \\
\textsc{Ciphertext} &= rk_{last} \oplus 000...000 \\
\textsc{Ciphertext} &= rk_{last}
\end{aligned}$$

Once the last round key has been retrieved, the key schedule can simply be reverted, as it is a static function, with only the main key as input. Thus, given any round key, the input key can be calculated.

Trojan Design. We use an AES-128 design optimized for space by implementing an SBox lookup in quarter rounds, i.e., four SBoxes enabling a 32-bit lookup in one cycle [66]. Our Trojan manipulates the SBox output of a round function to always output zero, ensuring that the ciphertext corresponds to the last round key, which can then be analyzed to obtain the input key. As a malicious designer, there are two ways to achieve this result, which are illustrated in Figure 11:

❶ Cut PIPs managing the SBox output so that the LUT input signals are consistently set to a constant '1'.
❷ Set the LUT's configuration string to always output '0'.

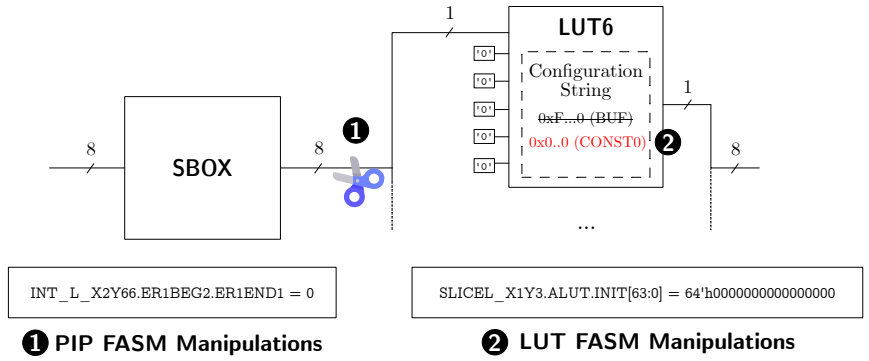


Fig. 11. Overview of possible manipulations to set the SBox output to 0.

To minimize manipulations, we implemented a buffer LUT after each SBox bit, thus requiring the modification of only one LUT. This approach ensures that the SBox output does not spread throughout the remainder of the design, as it could be employed in multiple locations.

Numerous potential triggers for Trojan deployment exist, such as activation based on a specific plaintext input or deployment after a certain number of encryptions. Using our manipulation framework, the trigger condition circuit can also be configured at runtime. For the sake of simplicity, in our proof-of-concept implementation, we trigger the Trojan via a simple button push.

Trojan Implementation. Depending on the placement, multiple frames may require manipulation. In our example design, we extracted the 32 LUT locations for manipulation using a Tcl script within Vivado applied to the generated netlist. Note that if one opts to cut the PIPs, the respective PIPs must be carefully selected, as pseudo-PIPs are often employed on routes, which are not configurable but static. Once the correct locations are extracted, the corresponding FASM file for the 32 SBox buffer output LUTs is created (as mentioned earlier, the AES implementation uses quarter-round states for the SBox lookup). From here, the flow to generate the manipulation ROM, described in Section 5.2.2, is followed.

Evaluation. Our proof-of-concept implementation successfully triggered and removed the Trojan when the readout detection was activated within 0.194 msec (19404 cycles at 100 MHz). This is below the minimum measured readout time we observed for our framework via JTAG. Generally, this time frame heavily depends on the placement and number of LUT changes.

Once the Trojan has been triggered, the attacker must start an encryption and can extract the main key by calculating the key schedule in reverse with the last round key received as ciphertext.

5.4 Discussion

We now briefly discuss the design of STRAT in the context of the vendor's partial reconfiguration flow and the readout detection strategies for other readback FPGA interfaces.

On Vendor Partial Reconfiguration Flows. While our framework can be built using the standard vendor partial reconfiguration flow from a high-level perspective, our framework provides various advantages by using the introduced atomic primitives in Section 4.1. In particular, the vendor flow generates a (large) partial bitstreams for a predetermined *pBlock*. From a malicious designers point of view, the vendor flow has major downsides: (1) the stored partial bitstream can be extracted, easily converted to a netlist, and then analyzed with static analysis techniques to detect a Trojan,

(2) the partial bitstream covers a much larger area, taking up unnecessary space and restricting a larger portion of the design due to the minimum size of the *pBlock*. In our case, first our readout protection must be circumvented and then the atomic modifications must be carefully mapped to the original design to identify malicious changes. Note that such analysis can be further complicated by additionally encoding and decoding the configuration delta, so that only encoded configuration delta data resides in memory.

On SelectMAP Readout. Similar to JTAG, the SelectMAP configuration interface can be used for readback. But as the I/O ports of the SelectMAP interface are shared with common FPGA I/Os, the SelectMAP is usually deactivated and used as standard I/O ports after the initial configuration. If these I/Os are configured to remain as the SelectMAP interface, the ICAP is disabled, as both features are mutually exclusive. In our setting this results in the fact that our Trojan can never be observed via SelectMAP at runtime because either the SelectMAP is deactivated or the ICAP is deactivated and our Trojan cannot be configured.

On ICAP Readout Detection and Response. In case ICAP is used as a readback mechanism by an analyst as well, e.g., to validate the integrity of their configuration at runtime, a malicious designer can simply re-route the wires from ICAP to an attacker-controlled hardware simulated ICAP interface that plays back the original configuration without the Trojan.

6 SECURITY IMPLICATIONS FOR CLOUD FPGAS

In this section, we highlight various security-critical real-world scenarios that may arise from using ICAP, based on our findings. We show that ICAP poses a significant security risk to multi-tenant cloud applications, including any tenant and the cloud provider itself.

6.1 Scenario: ICAP in (Multi-Tenant) Cloud FPGAs

As modern FPGAs contain millions of LUTs and storage elements, not all users may require the full capacity of a single FPGA device. The simultaneous sharing of FPGA resources among multiple tenants provides significant advantages, particularly in cloud environments where individual tenants may not need dedicated access to all FPGA hardware. Thus, resources could be effectively shared. Since commercial FPGAs contain a single on-chip power distribution network (PDN), malicious tenants can monitor or manipulate on-FPGA voltage values. For example, voltage sensors crafted from FPGA logic are used as a side-channel [84, 29, 65] to detect details of the victim tenant's computation, e.g., an encryption core [63]. In the latter case, the attacker deliberately wastes substantial power [60], causing the on-FPGA voltage to drop. This effect can induce faults in the victim tenant [30, 59] or cause an FPGA crash [61]. Thus, FPGA multi-tenancy is currently not supported by commercial cloud vendors, such as Amazon AWS [5] and Microsoft Azure [51] due to security concerns. However, several schemes [39, 38, 61, 44], have been proposed to counter the aforementioned attacks and multi-user co-tenancy is likely in the future [37]. Also, the use of intellectual property cores and the presence of the cloud provider shell already serves as a form of multi-tenancy in cloud FPGAs.

In this scenario, we explore a new, previously-unexplored risk for multi-tenant cloud FPGAs. While there are currently no commercial services available that allow for the purchase of multi-tenant clouds instances, significant research has been performed in the area. In the following we consider a cloud scenario, with multiple tenants sharing an FPGA that is managed by a shell from the cloud provider. We assume that a malicious participant has access to the ICAP interface. Our demonstration will highlight the potential for victim bitstream readback and espionage activities by this malicious entity through ICAP access. This investigation underscores the inherent security

vulnerability of exposing ICAP to tenants, emphasizing the importance of robust access control in future multi-tenancy service offerings.

6.1.1 High-Level Idea. Although a number of proposed implementation approaches exist, the physical layouts of multi-tenant FPGAs are similar. Different regions of the FPGA are typically assigned to distinct tenants. A *management engine* or *shell* [10] establishes external communication by implementing interfaces that enable data transmission to and from the assigned partitions. Tenants are physically isolated from one another, without any connections between them. Figure 12 presents a high-level overview.

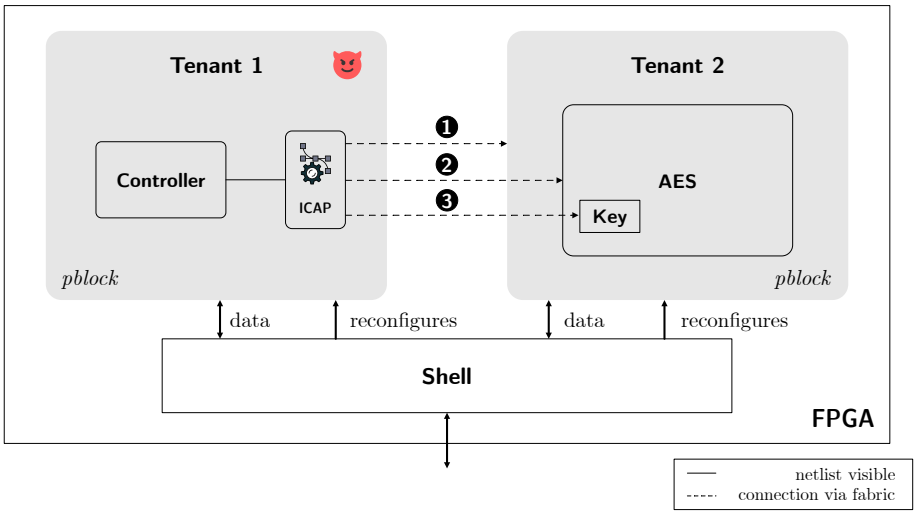


Fig. 12. High-Level Attack Overview in a Multi-Tenant Cloud Scenario.

If a malicious tenant can instantiate an ICAP interface, it poses a risk to all other tenants, including the cloud provider, as an attacker can create/read/update/delete circuits of other tenants based on our primitives discussed in Section 4.1. In the multi-tenant setting, the following attacks are particularly concerning:

1. **Design Readout.** The attacker can read the proprietary circuit of another tenant and analyze it after converting the recovered bitstream to its gate-level netlist.
2. **Design Manipulation.** Once a suitable target in the circuit has been identified, i.e., by means of reverse-engineering, it can be manipulated without the other tenant knowing.
3. **Register Readout.** As an alternative to manipulating, the attacker can access current register values in the victim circuit to leak sensitive information, such as cryptographic key material.

6.1.2 Case-Study: Reading the Key Register of Another Tenant. In this case study, we demonstrate how a malicious tenant can surreptitiously read the key register of another tenant via a fabric readout. In Figure 12, Tenant 2 has implemented an Advanced Encryption Standard (AES) core, while the malicious Tenant 1 can instantiate the ICAP primitive in their region.

Design & Implementation. In our proof-of-concept system, each tenant has a predefined region akin to the one depicted in Figure 12. We chose to utilize Xilinx's dynamic reconfiguration feature via JTAG, leaving all ICAP ports available on the device and minimizing the management engine's complexity. Each simulated tenant has access to dedicated output pins, where a UART connection

can be established for external communication – in a cloud setting, the communication would be established via a shell. We conducted the attack on a Virtex UltraScale+ VCU118 board.
Attack Execution. The goal of the malicious Tenant 1 is to read out the key of the Tenant 2 AES core. We assume that Tenant 1 knows the region where Tenant 2 is located, e.g., due to a previous assignment to that region or by reading the entire bitstream and examining the detailed FPGA utilization.

Note here that the attack is not limited to cryptographic keys, but any internal register state can be read, including unencrypted private data, state machine states obfuscated by logical locking, etc. During read out, it is essential to read back the exact registers and their order as even the complexity of finding the correct order of each register grows factorial (e.g., draw without replacing in order is $n!$).

(1) **Design Readout & Analysis** *(partially omitted).* The first step is to read out the design. The retrieved partial bitstream is converted to a netlist. From here, it can be analyzed to identify points of interest, such as the key register. Note that we partially omitted this step in our proof of concept since our focus is not on netlist analysis. Instead, we use knowledge from the placed and routed netlist and general design information.
(2) **Readout Frames with Activated Capture Mode.** The analyst can identify the correct readout positions in the fabric frames by knowing the points of interest, i.e., the FFs holding the key. The controller then reads the identified frames, as described in Section 4.1.3. To this end, the controller retrieves a list of corresponding frame addresses and transmits the readback frame. Note that the capture mode must be activated by enabling the capture bit in the `CTL1` configuration register (see Section 4.1.3).
(3) **Key Retrieval.** The individual key bits can then be extracted from the returned frames as the positions of the individual bits are known from the analysis in step 1.

pBlock Boundaries. We want to highlight that configuration *pBlocks* do not have strong boundaries and those that exist are enforced by vendor tools in generating partial bitstreams. Thus, we can easily overcome the boundaries using our comprehensive bitstream understanding and ICAP. The severity of this scenario is significant since other tenants have no way of detecting the harm, as the attack occurs entirely *under-the-hood* within the configuration engine, e.g., the fabric has no indication of being attacked. Moreover, there does not have to be a logical connection between the tenants' regions.

Malicious Cloud Provider. In addition to a malicious user, a malicious cloud provider could carry out attacks using this approach. ICAP is often utilized by the shell to enable the fast reconfiguration of tenants, as it allows for high data throughput [33]. Such a setup is employed by Amazon F1 [5] and Azure Cloud FPGAs [8] to reconfigure the customer region. Thus, given the already existing ICAP access in the shell, the cloud provider gains the ability to spy on or manipulate customers' designs.

Stealthiness of Attack. The attack operates discreetly, remaining undetectable by the victim tenant due to its communication with the configuration engine, which occurs *under-the-hood*. Any readout of the configured design goes unnoticed, as it entails a mere extraction of the configured frames without affecting ongoing operations. Furthermore, the extraction of register values remains concealed since there's no direct connection to the target register in the netlist — the readout transpires through the frames via the configuration engine. While design alterations may become apparent if they modify the design's functionality, the act of manipulation itself is mostly covert, as it involves frame reconfiguration *under-the-hood*. Undefined behaviour could occur during the

time period in which the frame is configured. So only manipulations, not readout attempts, would be noticeable by the tenant.

7 INCONSISTENCIES AND MISCONCEPTIONS IN USER GUIDES

In the following, we describe various discrepancies, inconsistencies, and misconceptions regarding security and implementation details found in Xilinx user guides during our research. We want to stress that the user guides provide misleading and false information in regards to ICAP possibilites. We want to highlight that *clear and concise* security documentation is of particular importance as any confusion may have devastating consequences in practice when designing (allegedly) secure systems.

7.1 Dispersed Security Recommendations in Official User Guides.

Before we address documentation details, we emphasize the challenge of dispersing consistent security documentation, cf. the Xilinx FPGA Security Design Hub [17]. The amount of documentation is not the deciding factor in creating complexity. Rather document inter-dependencies and cross-references between old and new documents yield challenges in clarifying assumptions and implementation details.

For instance, this issue is apparent in the documentation for the Starbleed attack [22, 21]. Older specialized security user guides, such as [XCN15038] originall published in 2015, do not mention the 2020 and 2022 Starbleed attacks. Even comprehensive user guides such as [UG470] and [XAPP1084], which have been updated since the attack's publication, do not mention it. Starbleed is only referenced in the *Design Advisory* 73541[80], which is linked only in the *Xilinx Design Advisory Master Answer Record* [78]. However, to the best of our knowledge, none of the relevant documents from the Security Hub link to these design advisory documents.

7.2 ICAP in a Dynamically Reconfigurable Region.

According to the Xilinx Vivado Dynamic Function eXchange User Guide [UG909] "*[...] the configuration components (such as* `BSCAN`, `STARTUP`, `ICAP`, *and* `FRAME ECC`*) must remain in the static portion of the design*" [UG909, p. 11]. While this statement is correct for 7-Series FPGAs, i.e., that include the `ICAPE2` primitive, this statement is not correct for UltraScale(+) FPGAs featuring the `ICAPE3` primitive. To this end, we implemented an `ICAPE3` primitive in a dynamic portion of the design on a VCU118 FPGA. A design rule check error is generated if an `ICAPE2` primitive on a 7-Series FPGA is instantiated, whereas no error is generated for UltraScale(+) FPGAs using `ICAPE3`. Hence, an `ICAPE3` primitive can be instantiated in a dynamic region. We tested this by creating a partial bitstream, that we configured after initial configuration via JTAG, which contains our ICAP prototyping framework. The framework was operating as normal. To the best of our knowledge, this capability is not mentioned in the user guides. Configuring another ICAP via ICAP, however, can cause undefined behaviour, as this is probably the reason, why Xilinx generally rules out the possibility of ICAP in partial reconfiguration.

7.3 Disabling Readback and Reconfiguration.

Xilinx provides numerous bitstream generation options, among others `BITSTREAM.READBACK.SECURITY`. The definition in user guide [UG908] states:

> "*Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback. Specifying Security Level2 disables Readback and Reconfiguration.*" [UG908, p.316]

Based on this statement, one may assume that readback and/or reconfiguration cannot be performed from any port when this option is enabled. However, Xilinx application note [XAPP1239] contradicts this statement:

> "*Partial bitstreams can be delivered unencrypted to the ICAP, or encrypted (with the same AES key) to any configuration port, so long as the latter has not been explicitly forbidden by the designer. Setting Security Level2 [...] prevents partial reconfiguration over external configuration ports.*" [XAPP1239, p.13]

In addition, Xilinx application note [XAPP1098] states that the feature is insecure – only bitstream encryption ensures readout prevention:

> "*A Vivado tools security option via a particular control bit in the bitstream provides a soft means of enabling and disabling readback. This bit can be changed during configuration. Therefore, readback disable is easy to defeat for devices that are not using an encrypted or authenticated bitstream.*" [XAPP1098, p.17]

The attack that Xilinx briefly describes in application note [XAPP1098] refers to the readout bit in the header of the bitstream file, to be more precise, the `SBITS[1:0]` bits in the control register `0 CTL0` can be set to disable a readout. An attacker is indeed able to flip this bit in the bitstream and can then enable the readout feature. To this end, they search for the write to the `CTL0` register in the bitstream header and change the two corresponding bits. Since the bitstream integrity is protected by a CRC32 checksum, the attacker adjusts or deactivates it. Note that this checksum is not a cryptographic checksum and can be calculated without any effort.

Thus, if bitstream encryption is not used, Xilinx does not provide a safe way to circumvent readback and the `BITSTREAM.READBACK.SECURITY` option is rendered insecure, which is not mentioned in the main user guide [UG908], where the option itself is defined.

7.4 Case-Study: Encrypted Bitstream Leakage

Conventionally, bitstream encryption serves as a deterrent against reverse engineering, manipulation, and unauthorized duplication. These bitstream protection schemes are predominantly instituted to safeguard Intellectual Property, achieved through the encryption and validation of configuration data, as discussed in Section 2.1.3.

However, in our considered scenario, we postulate the presence of a backdoor embedded by the attacker within a 3rd party malicious IP core. Should an attacker successfully integrate their compromised design into a product that employs bitstream encryption, they can consequently expose the entire encrypted layout. In the context of cloud infrastructures, where a customer might have ICAP access, the provider may choose to encrypt the shell within the design. Yet, this encryption is rendered moot if an attacker gains access to the ICAP, leading to a full exposure of the design even with bitstream encryption activated.

While we recognize that Xilinx addresses this attack vector in their documentation [79], our intention is twofold: (1) to illustrate the execution of such an attack and (2) to delineate the scenarios where this attack poses a significant threat. Ultimately, this attack underscores our contention that designs incorporating ICAP present considerable trust challenges.

High-Level Idea. The intuition of this attack is based on the readout of the currently configured (decrypted) bitstream configuration data frame by frame. Note that upon FPGA start, the encrypted bitstream is decrypted and stored in plaintext in fabric. While all external ports can be secured when bitstream encryption is used, the internal ports, such as ICAP, have unrestricted access to the device.

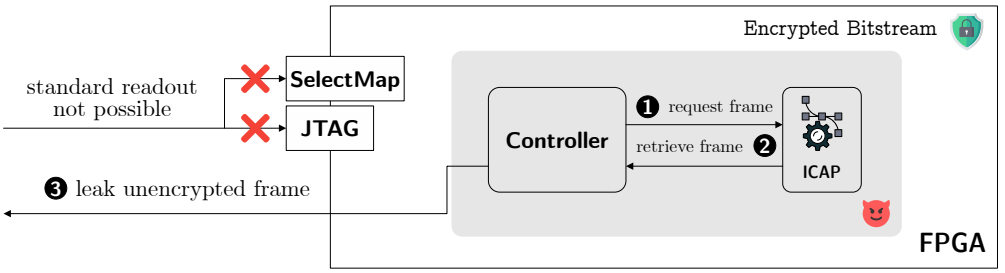


Fig. 13. Leakage of Unencrypted Bitstream via ICAP.

When using encrypted bitstreams, the design is decrypted on the fly when the FPGA is booted. Once the configuration is done, the design is stored in plaintext in fabric. With access to the ICAP, an attacker can readout the bitstream as they normally would:

1. **Request frame(s)** The readback command for one or more frames is issued, requesting the currently configured frame content.
2. **Retrieve frame(s).** The ICAP primitive sends back the regarding frame content in an unencrypted manner to the controller.
3. **Leakage.** The controller is then able to leak the unencrypted frame to the outside.

For a proof-of-concept, we implemented a design that includes ICAP access and enabled bitstream encryption for the entire design. Hence the design cannot be leaked by performing a readback on the external configuration interfaces.

Design & Implementation. The attacker can readout the decrypted bitstream at runtime using our ICAP prototyping framework, see Figure 6, as the readback is not prohibited at the trusted ICAP port. The readback bitstream can then be converted back to a gate-level netlist (e.g., using project X-Ray tools) then to reverse engineering the FPGA design. To improve the attack and hide the malicious intent, the malicious designer can use STRAT as introduced in Section 5.

8 DISCUSSION

We now discuss dynamic reconfiguration, our work's security implications, and potential mitigation strategies for designs requiring partial dynamic reconfiguration.

8.1 Dynamic Reconfiguration

Even though our analysis primarily focuses on Xilinx ICAP, our attack primitives can be applied through any port that accesses the internal configuration engine, see Figure 1.

Other Families and Vendors. For example, on Xilinx Zynq devices, the ICAP is inaccessible after initialization and has to be enabled from the software first. However, these devices feature a Processor Configuration Access Port (PCAP) configuration port, so our proposed attack primitives can be entirely realized from software and do not require an additional malicious hardware design or IP core. Future research should examine the applicability of dynamic reconfiguration-based attacks for other FPGA vendors such as Intel Altera and Lattice. However, the bitstream configuration file format is still proprietary and open-source documentation through reverse engineering is sparse to non-existent, challenging in-depth analysis.

Xilinx Security Considerations on ICAP. Xilinx acknowledges that ICAP is a powerful tool and potentially dangerous [XAPP1098]. While official documentation claims that ICAP is a trusted channel since only the user can instantiate it, our work highlights typical FPGA application scenarios where system designers may not have full control over the final configuration. Thus establishing trust in a design that uses ICAP is virtually impossible.

8.2 Security Implications

We demonstrated various severe security implications of ICAP in malicious hardware designs ranging from surreptitiously leaking cryptographic keys in third-party IP cores, leakage of allegedly encrypted bitstreams, and snooping on sensitive key material in multi-tenant FPGA cloud systems.

Insights on Attack Vectors. In this article, we investigate ICAP as an attack vector from the perspective of malicious designers and (multi-tenant) FPGA cloud systems. While malicious designers represent a significant threat in their own right, it is imperative to understand all potential attack vectors in detail. Such understanding serves a dual purpose: (1) it furnishes insights that can be instrumental in the design of sound defenses and proactive countermeasures, and (2) it amplifies awareness, ensuring that stakeholders are well-informed about emerging attack threats, thereby fortifying the larger ecosystem against potential vulnerabilities.

On Detection of Dynamic Reconfiguration Trojans using ICAP. As noted before, no static analysis technique can effectively detect our proposed hardware Trojans since we do not employ known partial bitstreams but rather a custom configuration format that can be flexibly adapted. However, dynamic analysis techniques are currently limited to both simulation and on-chip debugging: Simulation of dynamic reconfiguration is challenging to support since the entire FPGA fabric, including the configuration plane, has to be mimicked. On-chip debugging requires trust in the execution environment, i.e., that the readout mechanism works as intended and is not surreptitiously manipulated. However, our readout detection and response engine demonstrated how such trust could be significantly reduced.

On Security Documentation. Based on the dispersion and inconsistency of security-related documentation across multiple user guides, application notes, and design advisories, we recommend enhancing and simplifying security documentation. This recommendation does not only hold for Xilinx themselves. Therefore, we advocate for the research (security) community to actively engage in such documentation approaches to enable users to fully understand the security features and potential vulnerabilities of all FPGAs to dynamic partial reconfiguration.

8.3 Mitigations

A straightforward approach to mitigate threats associated with the ICAP interface is to completely restrict its usage in user devices. However, this is not feasible for applications that rely on partial reconfiguration. In such cases, we propose an ICAP firewall that restricts and monitors access to the ICAP interface.

8.3.1 ICAP Firewall. Instead of granting users direct access to the ICAP interface, a parameterized filtering system can be designed to mediate access. This ICAP firewall analyzes incoming commands and only allows access to a predefined fabric region.

Note that Xilinx offers a *Security Monitor IP*, which can be used to monitor partial reconfiguration [81]. However, exact functionality remains unclear, since most information seem to be available after signing an Non-Disclosure Agreement (NDA).

Design. To limit access, users are only granted access to an ICAP firewall primitive managed by a shell. Access restrictions can be easily enforced by providing an address range based on FPGA frame coordinates. If the user wants to access the ICAP, they must go through the ICAP firewall.

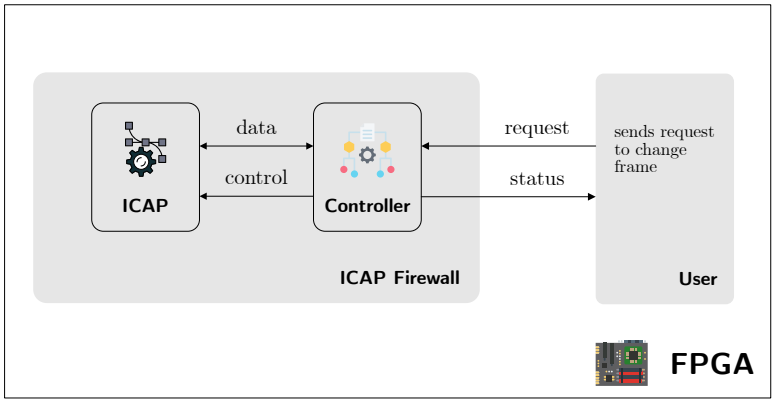


Fig. 14. ICAP Firewall Overview.

The firewall could support frame-based read and write access. For example, if a user wants to re-configure the frame at address 0x01234567, a potential command sequence could appear as follows:

$0x00000001$	read = $0x0$, write = $0x1$
$0x01234567$	frame address
frame content	frame content in case of write

The firewall would conduct checks on each incoming frame address. If the requested frame address is in the allowed frame region of the tenant, the operation is executed. The ICAP firewall can than craft the exact ICAP command sequence to execute the request.

Generally, dynamic configuration would become slower, as normally frames can be written much quicker, since a special bitstream compression format is typically used by Xilinx.

Deployment. In a multi-tenant cloud environment, the cloud provider can employ such a firewall to limit user requests to their assigned frame regions. Similarly, when integrating third-party IP cores, designers can restrict access to the region where the IP core is placed. However, in a cloud scenario, a malicious tenant may still be able to configure malicious circuitry within their own region, e.g., drawing excessive power to harm the cloud provider. Identifying such requests is challenging since it requires an in-depth analysis of the dynamic reconfiguration changes at run-time, e.g., to detect that a reconfiguration change realizes a circuit that draws excessive power. One idea would be to integrate the FPGA virus scanner as introduced by La *et al.* [42]. However, this still presents a significant engineering effort and would probably exceed on-chip compute capabilities. The most secure way to address the issue is to disallow ICAP access.

9 CONCLUSION

We presented a comprehensive security analysis of the Xilinx FPGA (re-)configuration process, with a focus on the Internal Configuration Access Port (ICAP). Based on prior bitstream reverse

engineering, we investigated novel and generic (attack) primitives that exploit dynamic reconfiguration to spy on and manipulate FPGA hardware design circuitry at runtime. Our novel framework STRAT simplifies development of such malicious circuitry, including features to even evade readout detection efforts, as demonstrated in our AES key leakage case study. Moreover, we investigated the risk in (multi-tenant) FPGA cloud environments.

In summary, we demonstrated that designs with ICAP are inherently hard to trust with current analysis methods, thus necessitating a paradigm shift from conventional static analysis methods to dynamic approaches, however, required dynamic approaches are currently not supported by available tools.

ACKNOWLEDGMENTS

This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy (EXC 2092 CASA 390781972). We also want to thank Julian Speith for giving us access to his Bitstream Manipulation Framework, which we used for the development of STRAT.

REFERENCES

[1] Nils Albartus, Max Hoffmann, Sebastian Temme, Leonid Azriel, and Christof Paar. 2020. DANA universal dataflow analysis for gate-level netlist reverse engineering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020, 4, 309–336.

[2] Chips Alliance. [n. d.] Project U-Ray. (). Retrieved Apr. 2, 2023 from https://github.com/f4pga/prjuray.

[3] Chips Alliance. [n. d.] Project X-Ray. (). Retrieved Apr. 2, 2023 from https://github.com/f4pga/prjxray.

[4] Lilas Alrahis, Muhammad Yasin, Hani H. Saleh, Baker Mohammad, and Mahmoud Al-Qutayri. 2019. Functional reverse engineering on sat-attack resilient logic locking. In *IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, Japan, May 26-29, 2019*. IEEE, 1–5.

[5] Amazon. [n. d.] *Amazon EC2 F1*. https://aws.amazon.com/ec2/instance-types/f1/. Amazon AWS.

[6] Leonid Azriel, Ran Ginosar, and Avi Mendelson. 2019. Sok: an overview of algorithmic methods in IC reverse engineering. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2019, London, UK, November 15, 2019*. Chip-Hong Chang, Ulrich Rührmair, Daniel E. Holcomb, and Patrick Schaumont, (Eds.) ACM, 65–74.

[7] Leonid Azriel, Julian Speith, Nils Albartus, Ran Ginosar, Avi Mendelson, and Christof Paar. 2021. A survey of algorithmic methods in IC reverse engineering. *J. Cryptogr. Eng.*, 11, 3, 299–315.

[8] Microsoft Azure. [n. d.] Np-series - azure virtual machines. https://learn.microsoft.com/en-us/azure/virtual-machines/np-series. ().

[9] Sebastian Banescu and Alexander Pretschner. 2018. Chapter five - a tutorial on software obfuscation. In Advances in Computers. Vol. 108. Atif M. Memon, (Ed.) Elsevier, 283–353. DOI: https://doi.org/10.1016/bs.adcom.2017.09.004.

[10] Christophe Bobda et al. 2022. The future of FPGA acceleration in datacenters and the cloud. *TRETS*, 15, 3, Article 34, (Sept. 2022), 42 pages.

[11] Andrew Boutros and Vaughn Betz. 2021. FPGA architecture: principles and progression. *IEEE Circuits and Systems Magazine*, 21, 2, (Apr. 2021), 4–29.

[12] Tom Branca. 1999. How to add features and fix bugs - remotely. *Xcell*, 33, 12–14.

[13] Michaela Brunner, Johanna Baehr, and Georg Sigl. 2019. Improving on state register identification in sequential hardware reverse engineering. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*. IEEE, 151–160.

[14] Michaela Brunner, Alexander Hepp, Johanna Baehr, and Georg Sigl. 2022. Toward a human-readable state machine extraction. *ACM Trans. Design Autom. Electr. Syst.*, 27, 6, 58:1–58:31.

[15] Burçin Çakir and Sharad Malik. 2015. Hardware trojan detection for gate-level ics using signal correlation based clustering. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*. Wolfgang Nebel and David Atienza, (Eds.) ACM, 471–476. http://dl.acm.org/citation.cfm?id=2755860.

[16] Chang-Seok Choi and Hanho Lee. 2006. An reconfigurable FIR filter design on a partial reconfiguration platform. In *Proceedings of the International Conference on Communications and Electronics*. (Oct. 2006).

[17] Xilinx Corporation. [n. d.] Fpga design security. https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0082-fpga-security-hub.html. ().

[18] Defense Science Board Washington DC. 2005. Report of the Defense Science Board Task Force on High Performance Microchip Supply. (2005).

[19] Paul Dempsey. [n. d.] Teardown: HTC Vive Pro VR headset. https://eandt.theiet.org/content/articles/2018/05/teardown-htc-vive-pro-vr-headset/. ().

[20] Chris Edwards. 2015. Growing pains for deep learning. *Communications of the ACM*, 58, 7, (July 2015), 14–16.

[21] Maik Ender, Gregor Leander, Amir Moradi, and Christof Paar. 2022. A cautionary note on protecting Xilinx' UltraScale(+) bitstream encryption and authentication engine. In *30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2022, New York City, NY, USA, May 15-18, 2022*. IEEE, 1–9.

[22] Maik Ender, Amir Moradi, and Christof Paar. 2020. The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series fpgas. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. Srdjan Capkun and Franziska Roesner, (Eds.) USENIX Association, 1803–1819.

[23] Maik Ender, Pawel Swierczynski, Sebastian Wallat, Matthias Wilhelm, Paul Martin Knopp, and Christof Paar. 2019. 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, ASPDAC 2019, Tokyo, Japan, January 21-24, 2019*. Toshiyuki Shibuya, (Ed.) ACM, 112–119.

[24] Marc Fyrbiak, Simon Rokicki, Nicolai Bissantz, Russell Tessier, and Christof Paar. 2018. Hybrid obfuscation to protect against disclosure attacks on embedded microprocessors. *IEEE Trans. Computers*, 67, 3, 307–321.

[25] Marc Fyrbiak, Sebastian Wallat, Jonathan Déchelotte, Nils Albartus, Sinan Böcker, Russell Tessier, and Christof Paar. 2018. On the difficulty of FSM-based hardware obfuscation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018, 3, 293–330.

[26] Marc Fyrbiak, Sebastian Wallat, Pawel Swierczynski, Max Hoffmann, Sebastian Hoppach, Matthias Wilhelm, Tobias Weidlich, Russell Tessier, and Christof Paar. 2018. HAL-The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion. *IEEE Transactions on Dependable and Secure Computing*.

[27] Adrià Gascón, Pramod Subramanyan, Bruno Dutertre, Ashish Tiwari, Dejan Jovanovic, and Sharad Malik. 2014. Template-based circuit understanding. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*. IEEE, 83–90.

[28] Lubos Gaspar, Viktor Fischer, Lilian Bossuet, and Robert Fouquet. 2012. Secure extension of FPGA general purpose processors for symmetric key cryptography with partial reconfiguration capabilities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 5, 3, (Oct. 2012), 1–13.

[29] Ilias Giechaskiel, Ken Eguro, and Kasper Rasmussen. 2019. Leakier wires: exploiting FPGA long wires for covert- and side-channel attacks. *Transactions on Reconfigurable Technology and Systems (TRETS)*.

[30] Dennis R. E. Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B. Tahoori. 2018. An experimental evaluation and analysis of transient voltage fluctuations in FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26, 10, 1817–1830.

[31] Ivan Gonzalez, Sergio Lopez-Buedo, Francisco J. Gomez, and Javier Martinez. 2003. Using partial reconfiguration in cryptographic applications: an implementation of the IDEA algorithm. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. (Sept. 2003), 194–203.

[32] HAL. [n. d.] HAL – The Hardware Analyzer. (). Retrieved Apr. 2, 2023 from https://github.com/emsec/hal.

[33] Simen Gimle Hansen, Dirk Koch, and Jim Tørresen. 2011. High speed partial run-time reconfiguration using enhanced ICAP hard macro. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011 - Workshop Proceedings*. IEEE, 174–180. DOI: 10.1109/IPDPS.2011.139.

[34] Matthew Hicks, Murph Finnicum, Samuel T. King, Milo M. K. Martin, and Jonathan M. Smith. 2010. Overcoming an untrusted computing base: detecting and removing malicious hardware automatically. *login Usenix Mag.*, 35, 6. https://www.usenix.org/publications/login/december-2010-volume-35-number-6/overcoming-untrusted-computing-base-detecting.

[35] Max Hoffmann and Christof Paar. 2021. Doppelganger obfuscation - exploring the defensive and offensive aspects of hardware camouflaging. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021, 1, 82–108. DOI: 10.46586/tches.v2021.i1.82-108.

[36] Jatin Kataria, Rick Housley, Joseph Pantoga, and Ang Cui. 2019. Defeating Cisco trust anchor: A case-study of recent advancements in direct FPGA bitstream manipulation. In *13th USENIX Workshop on Offensive Technologies, WOOT 2019, Santa Clara, CA, USA, August 12-13, 2019*. Alex Gantman and Clémentine Maurice, (Eds.) USENIX Association. https://www.usenix.org/conference/woot19/presentation/kataria.

[37] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. 2018. Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS. In *13th USENIX Symposium on Operating Systems Design and Implementation*. Carlsbad, CA, USA, (Oct. 2018), 107–27.

[38] Jonas Krautter, Dennis RE Gnad, Falk Schellenberg, Amir Moradi, and Mehdi B Tahoori. 2019. Active fences against voltage-based side channels in multi-tenant FPGAs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[39] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. 2019. Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12, 3, 1–26.

[40] Christian Krieg, Clifford Wolf, and Axel Jantsch. 2016. Malicious LUT: a stealthy FPGA trojan injected and triggered by the design flow. In *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*. Frank Liu, (Ed.) ACM, 43. DOI: 10.1145/2966986.2967054.

[41] Tuan La, Khoa Dang Pham, Joseph Powell, and Dirk Koch. 2021. Denial-of-service on FPGA-based cloud infrastructures - attack and defense. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021, 3, 441–464. DOI: 10.46586/tches.v2021.i3.441-464.

[42] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. 2020. FPGADefender: malicious self-oscillator scanning for Xilinx UltraScale FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 13, 3, 15:1–15:31. DOI: 10.1145/3402937.

[43] Wenchao Li, Adrià Gascón, Pramod Subramanyan, Wei Yang Tan, Ashish Tiwari, Sharad Malik, Natarajan Shankar, and Sanjit A. Seshia. 2013. WordRev: finding word-level structures in a sea of bit-level gates. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2-3, 2013*. IEEE Computer Society, 67–74.

[44] Yukui Luo and Xiaolin Xu. 2020. A quantitative defense framework against power attacks on multi-tenant FPGA. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. Virtual Event, USA, (Nov. 2020), 1–4.

[45] Roel Maes, Dries Schellekens, and Ingrid Verbauwhede. 2012. A pay-per-use licensing scheme for hardware IP cores in recent SRAM-based FPGAs. *IEEE Trans. Inf. Forensics Secur.*, 7, 1, 98–108.

[46] Kristiyan Manev, Joseph Powell, Kaspar Matas, and Dirk Koch. 2022. Byteman: A bitstream manipulation framework. In *International Conference on Field-Programmable Technology, (IC)FPT 2022, Hong Kong, December 5-9, 2022*. IEEE, 1–9. DOI: 10.1109/ICFPT56656.2022.9974549.

[47] Mahmoud Masadeh, Yassmeen Elderhalli, Osman Hasan, and Sofiène Tahar. 2021. A quality-assured approximate hardware accelerators-based on machine learning and dynamic partial reconfiguration. *ACM Journal on Emerging Technologies in Computing Systems*, 17, 4, (Oct. 2021), 57:1–57:19.

[48] Travis Meade, Kaveh Shamsi, Thao Le, Jia Di, Shaojie Zhang, and Yier Jin. 2018. The old frontier of reverse engineering: netlist partitioning. *J. Hardware and Systems Security*, 2, 3, 201–213.

[49] Travis Meade, Shaojie Zhang, and Yier Jin. 2016. Netlist reverse engineering for high-level functionality reconstruction. In *21st Asia and South Pacific Design Automation Conference, ASP-DAC 2016, Macao, Macao, January 25-28, 2016*. IEEE, 655–660.

[50] Travis Meade, Shaojie Zhang, Zheng Zhao, David Pan, and Yier Jin. 2016. Gate-level netlist reverse engineering tool set for functionality recovery and malicious logic detection. In *International Symposium for Testing and Failure Analysis (ISTFA)*.

[51] Microsoft. 2023. *Microsoft Azure Machine Learning*. https://azure.microsoft.com/en-us/pricing/details/machine-learning/. Microsoft.

[52] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. 2011. 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, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*. Yan Chen, George Danezis, and Vitaly Shmatikov, (Eds.) ACM, 111–124.

[53] Amir Moradi, Markus Kasper, and Christof Paar. 2012. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the Xilinx Virtex-4 and Virtex-5 bitstream encryption mechanism. In *Topics in Cryptology - CT-RSA 20W12 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings* (Lecture Notes in Computer Science). Orr Dunkelman, (Ed.) Vol. 7178. Springer, 1–18.

[54] Amir Moradi, David F. Oswald, Christof Paar, and Pawel Swierczynski. 2013. Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering. In *The 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, Monterey, CA, USA, February 11-13, 2013*. Brad L. Hutchings and Vaughn Betz, (Eds.) ACM, 91–100.

[55] Amir Moradi and Tobias Schneider. 2016. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers* (Lecture Notes in Computer Science). François-Xavier Standaert and Elisabeth Oswald, (Eds.) Vol. 9689. Springer, 71–87.

[56] Juanjo Noguera and Irwin O. Kennedy. 2007. Power reduction in network equipment through adaptive partial reconfiguration. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. (Sept. 2007), 240–245.

[57] Khoa Dang Pham, Edson L. Horta, and Dirk Koch. 2017. BITMAN: A tool and API for FPGA bitstream manipulations. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*. David Atienza and Giorgio Di Natale, (Eds.) IEEE, 894–897. DOI: 10.23919/DATE.2017.7927114.

[58] Joseph Powell, Kaspar Matas, Kristiyan Manev, and Dirk Koch. 2022. FPL demo: FPGA bitstream virus scanning. In *32nd International Conference on Field-Programmable Logic and Applications, FPL 2022, Belfast, United Kingdom, August 29 - Sept. 2, 2022*. IEEE, 469. DOI: 10.1109/FPL57034.2022.00085.

[59] George Provelengios, Daniel Holcomb, and Russell Tessier. 2019. Characterizing power distribution attacks in multi-user FPGA environments. In *International Conference on Field Programmable Logic and Applications (FPL)*, 194–201.

[60] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power wasting circuits for cloud FPGA attacks. In *International Conference on Field Programmable Logic and Applications (FPL)*, 231–235.

[61] George Provelengios, Daniel E. Holcomb, and Russell Tessier. 2021. Mitigating voltage attacks in multi-tenant FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, (July 2021), 9:1–9:24.

[62] Lampros Pyrgas and Paris Kitsos. 2018. A hybrid FPGA trojan detection technique based-on combinatorial testing and on-chip sensing. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications - 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings* (Lecture Notes in Computer Science). Nikolaos S. Voros, Michael Hübner, Georgios Keramidas, Diana Goehringer, Christos P. Antonopoulos, and Pedro C. Diniz, (Eds.) Vol. 10824. Springer, 294–303. DOI: 10.1007/978-3-319-78890-6_24.

[63] Chethan Ramesh, Shivukumar Basanagouda Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. 2018. FPGA side channel attacks without physical access. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. (Apr. 2018), 45–52. DOI: 10.1109/FCCM.2018.00016.

[64] J. Sheeba Rani and C. Sai Phalghun. 2014. FPGA based partial reconfigurable FIR filter design. In *Proceedings of the IEEE International Advance Computing Conference*. (Feb. 2014).

[65] Falk Schellenberg, Dennis RE Gnad, Amir Moradi, and Mehdi B Tahoori. 2021. An inside job: remote power analysis attacks on FPGAs. *IEEE Design & Test*.

[66] SecWorks. [n. d.] Secworks AES Core with On-The-Fly Keygen. [Online]. Available: https://github.com/secworks/aes/tree/on_the_fly_keygen. ().

[67] Florian Stolz et al. 2021. Lifeline for FPGA protection: obfuscated cryptography for real-world security. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021, 4, 412–446.

[68] Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson. 2002. A dynamically reconfigurable adaptive Viterbi decoder. In *Proceedings of the 10th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. (Feb. 2002), 227–236.

[69] Pawel Swierczynski, Marc Fyrbiak, Philipp Koppe, and Christof Paar. 2015. FPGA trojans through detecting and weakening of cryptographic primitives. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34, 8, 1236–1249.

[70] Pawel Swierczynski, Amir Moradi, David F. Oswald, and Christof Paar. 2015. Physical security evaluation of the bitstream encryption mechanism of altera stratix II and stratix III fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 7, 4, 34:1–34:23.

[71] Shahin Tajik, Heiko Lohrke, Jean-Pierre Seifert, and Christian Boit. 2017. 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, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, (Eds.) ACM, 1661–1674.

[XAPP1230] Stephanie Tapp. [n. d.] Xapp1230 - configuration readback capture in UltraScale FPGAs. https://docs.xilinx.com/v/u/en-US/xapp1230-configuration-readback-capture. ().

[72] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A survey of hardware trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27, 1, 10–25. DOI: 10.1109/MDT.2010.7.

[73] Aaron Tilley. [n. d.] This mysterious chip in the iPhone 7 could be key to Apple's AI push. https://www.forbes.com/sites/aarontilley/2016/10/17/iphone-7-fpga-chip-artificial-intelligence/?sh=4158ca3d3c69. ().

[74] Tayssir Touili and Xin Ye. 2017. Reachability analysis of self modifying code. In *22nd International Conference on Engineering of Complex Computer Systems, ICECCS 2017, Fukuoka, Japan, November 5-8, 2017*. IEEE Computer Society, 120–127. DOI: 10.1109/ICECCS.2017.19.

[75] Shikha Tripathi, Rishi Mathur, and Jyoti Arya. 2010. Unified 3GPP and 3GPP2 turbo encoder FPGA implementation using run-time partial reconfiguration. In *Wireless Telecommunications Symposium (WTS)*. (Apr. 2010).

[76] Timothy Trippel, Kang G. Shin, Kevin B. Bush, and Matthew Hicks. 2021. Bomberman: defining and defeating hardware ticking timebombs at design-time. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 970–986. DOI: 10.1109/SP40001.2021.00052.

[UG002] Xilinx Corporation. 2005. *UG002 - Virtex-II Platform FPGA User Guide 002*. Xilinx Corporation. (Mar. 2005).

[UG908] Xilinx Corporation. [n. d.] *UG908 - Vivado Design Suite User Guide Programming and Debugging*. Xilinx Corporation.

[77] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. 2013. FANCI: identification of stealthy malicious logic using boolean functional analysis. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, (Eds.) ACM, 697–708. DOI: 10.1145/2508859.2516654.

[XAPP1098] 2021. Xapp1098 - developing tamper-resistant designs with UltraScale and UltraScale+ FPGAs. https://docs.xilinx.com/v/u/en-US/xapp1098-tamper-resist-designs. Xilinx Corporation, (Mar. 2021).

[XAPP1239] Xilinx Corporation. 2021. *XAPP1239 - Using Encryption to Secure a 7 Series FPGA Bitstream*. Xilinx Corporation. (Mar. 2021).

[XCN15038] Xilinx Corporation. 2015. *XCN15038 - RSA Authentication and Supporting Configuration Modes*. Xilinx Corporation. (Dec. 2015).

[78] Xilinx. 2023. *42946 - Design Advisory Master Answer Record for Kintex-7 FPGA*. Xilinx Corporation. (Mar. 2023).

[79] Xilinx. 2023. *52881 - Configuration - BitStream Encryption - How to create and program an encrypted bitstream*. Xilinx Corporation. (Feb. 2023).

[80] Xilinx. 2023. *73541 - Design Advisory for 7 Series/Virtex-6 FPGAs: Defeating Bitstream Encryption*. Xilinx Corporation. (Feb. 2023).

[UG470] Xilinx. [n. d.] Ug470 - 7 series FPGAs configuration. https://docs.xilinx.com/r/en-US/ug470_7Series_Config. ().

[UG570] Xilinx. [n. d.] Ug570 - ultrascale architecture configuration. https://docs.xilinx.com/v/u/en-US/ug570-ultrascale-configuration. ().

[UG909] Xilinx. [n. d.] Ug909 - vivado design suite user guide: dynamic function exchange. https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration. ().

[81] AMD Xilinx. [n. d.] SECURITY MONITOR IP. [Online]. Available: https://www.xilinx.com/support/documents/product-briefs/security-monitor-ip-core-product-brief.pdf. ().

[XAPP1084] Ed Peterson Xilinx. [n. d.] Xapp1084 - developing tamper resistant designs with Xilinx Virtex-6 and 7 series FPGAs. https://docs.xilinx.com/v/u/en-US/xapp1084_tamp_resist_dsgns. ().

[82] Jie Zhang, Feng Yuan, Lingxiao Wei, Yannan Liu, and Qiang Xu. 2015. Veritrust: verification for hardware trust. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 34, 7, 1148–1161. DOI: 10.1109/TCAD.2015.2422836.

[83] Xuzhi Zhang, Xiaozhe Shao, George Provelengios, Naveen Kumar Dumpala, Lixin Gao, and Russell Tessier. 2020. CoNFV: a heterogeneous platform for scalable network function virtualization. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 14, 1, (Nov. 2020), 1–29.

[84] Mark Zhao and G. Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *Proceedings of IEEE Symposium on Security and Privacy*. San Francisco, CA, USA, (May 2018), 805–20.