Malicious Routing: Circumventing Bitstream-level Verification for FPGAs

Qazi Arbab Ahmed, Tobias Wiersema, and Marco Platzner
Department of Computer Science, Paderborn University, Germany
{qazi, wiersema, platzner}@mail.upb.de

***Abstract*—The battle of developing hardware Trojans and corresponding countermeasures has taken adversaries towards ingenious ways of compromising hardware designs by circumventing even advanced testing and verification methods. Besides conventional methods of inserting Trojans into a design by a malicious entity, the design flow for field-programmable gate arrays (FPGAs) can also be surreptitiously compromised to assist the attacker to perform a successful malfunctioning or information leakage attack. The advanced stealthy malicious look-up-table (LUT) attack activates a Trojan only when generating the FPGA bitstream and can thus not be detected by register transfer and gate level testing and verification. However, also this attack was recently revealed by a bitstream-level proof-carrying hardware (PCH) approach.**

In this paper, we present a novel attack that leverages malicious routing of the inserted Trojan circuit to acquire a dormant state even in the generated and transmitted bitstream. The Trojan's payload is connected to primary inputs/outputs of the FPGA via a programmable interconnect point (PIP). The Trojan is detached from inputs/outputs during place-and-route and re-connected only when the FPGA is being programmed, thus activating the Trojan circuit without any need for a trigger logic. Since the Trojan is injected in a post-synthesis step and remains unconnected in the bitstream, the presented attack can currently neither be prevented by conventional testing and verification methods nor by recent bitstream-level verification techniques.

***Index Terms*—FPGA security, hardware Trojans, EDA tools, FPGA design flow**

I. Introduction

Reconfigurable systems, in particular, field programmable gate arrays (FPGAs) are nowadays largely used as acceleration platforms for diverse applications in, for example, cloud computing, high-performance computing, autonomous driving, and also for military purposes. Compared to application-specific integrated circuits (ASICs), FPGAs benefit from a flexible and re-programmable computation fabric and short time-to-market and have thus become a fundamental part of the embedded systems space, including the Internet of Things (IoT) and end-user electronics.

To avoid any malfunctioning during operation, it is of utmost importance to provide security guarantees for these reconfigurable systems used by the customer. In FPGAs, the implemented design is loaded onto the manufactured and tested device in form of a configuration bitstream. This process resembles software programs but differentiates FPGAs from ASICs, where the functionality of the manufactured device cannot be changed once it is fabricated. FPGAs thus offer the flexibility to update erroneous designs or configure new designs on-the-fly. While hardware tampering attacks by untrusted foundries, such as the insertion of hardware Trojans into silicon, are possible, FPGAs can be considered more resistant to such attacks compared to ASICs since at manufacturing-time an attacker would have no knowledge about the design which would later be implemented on the FPGA device. Nevertheless, a bitstream configuration file may be susceptible to reverse engineering attacks, which would allow an attacker to steal the implemented design or insert malicious circuitry by regenerating the bitstream [1], [2].

Due to their re-programmable fabrics, FPGAs are very well suited for implementing cryptographic algorithms where extensive bit and byte level computations are required, such as Advanced Encryption Standard (AES), that is used to allow for secure data transmission in many critical applications [4]. Subsequently, such devices are responsible for the secret flow of information, which may be directly related to the user's privacy. Consequently, there is a high risk of secret information being stolen unnoticed through inserted hardware Trojans.

Using side channel analyses to leak secret information has been proven to be a practical attack for FPGAs where the attacker can gain physical access to the FPGA and use power side channels to retrieve, e.g., a secret key [3], [5]. Sophisticated attackers might even attempt to insert malicious circuitry that remains inactive until it is triggered by a specific condition or external input to circumvent all the design-time testing and verification processes. Such an attacker could be a dishonest employee in the design house who inserts the Trojan in a design, or there may be a Trojan in a third-party Intellectual Property core (IP) provided by an untrusted vendor. Besides that, there is also a chance that both, the design house and IP vendors, are trustworthy, but the EDA tools maintained by the vendors are undermined, resulting in an enhanced power to the attack by infecting a bundle of designs in just one go. The authors in [6] recently presented an attack by introducing a stealthy "malicious LUT" hardware Trojan inserted during design flow that employs a two-stage mechanism of insertion and activation. This Trojan remains dormant throughout the design phase and is activated when the bitstream is written, thus circumventing design-time verification techniques such as [7], [14]. However, a bitstream-level proof-carrying hardware (PCH) technique presented in [8] was able to reveal the attack. Supplementary to this attack, we propose a novel attack that

This work has been partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre 901 "On-The-Fly Computing" under the project number 160364472 and "HEC/DAAD Pakistan".

can intensify the stealthy nature of the inserted Trojan, such that the Trojan remains dormant in the bitstream as well and is only activated when actually configuring the device. The Trojan circuitry is introduced after synthesis, i.e., when the netlist is being read by the placement and routing tool. As the presented novel Trojan is activated after the bitstream generation step, the certificates generated by the PCH technique would lead to false-negatives.

Our novel contributions in this paper are as follows:

- We present an FPGA Trojan that remains inert throughout the design flow, even in the bitstream, which to the extent of our knowledge is the first to circumvent even bitstream-level verification techniques. Trojans inserted by EDA design tools so far are inserted either at a design stage or into the bitstream.
- Our attack is based on the malicious routing of a Trojan circuit that is unconnected from the actual user design before configuring the target device, and that is reconnected by a malicious programming tool to activate the Trojan.

The remainder of this paper is structured as follows: In Section II we will introduce some related work on hardware Trojan insertion and detection in FPGAs and then present our threat model in Section III. In Section IV we will detail our Trojan insertion methodology. We demonstrate our attack with examples in Section V, followed by a discussion of our approach in Section VI. In Section VII we will then conclude the paper.

II. Related Work

Hardware Trojans are malicious circuits purposely inserted into the original design to distract the intended functionality or disclose confidential information. There are a variety of ways to insert and implement hardware Trojans, depending upon their design, characteristics, activation, and actions. An FPGA bitstream configuration file containing a particular design information may also be susceptible to hardware Trojans. The three major entities during the bitstream generation processes that may insert malicious logic / functionality in the legitimate design are a) a design house or a malicious designer in the design house, b) compromised EDA tools, and c) malicious communication channels, i.e., via man-in-the-middle (MiM) attacks. Recently, Duncan et al. [9] classified the different threats to a bitstream at various stages during the FPGA design flow. In the first stage of this taxonomy, the threat to a bitstream generation by the design house and third-party IPs are categorized into malicious and non-malicious intent, where the latter refers to tools-induced vulnerabilities. The introduction of vulnerabilities or malicious circuitry through design tools could be more interesting for the attacker due to the simplicity of the attack and the implanted malicious circuitry can also be more devious. The activation of dormant logic inserted at any point could force the device during operation to go to certain undesired states such as the denial of service, change of functionality, and secret information leakage.

In 2013, Chakraborty et al. [10] presented a mechanism to taint the FPGA bitstream that was first to insert a Trojan directly in the bitstream. The Trojan is inserted in an unencrypted bitstream using an add-on program that modifies the bitstream configuration file. Based on their connectivity to the original circuit, two types of Trojans have been proposed that can be inserted: A type-I Trojan has no connection to the original circuit (hence non-functional Trojans) and is inserted to the free resources of FPGAs. A Trojan circuit consisting of ring oscillators has been implemented to increase the temperature of the FPGA, which causes reliability issues and early aging. Such kinds of attacks are denial of service attacks. The success of the attack relies on the availability of resources in the proper locations of the FPGA. However, such kinds of Trojans may be difficult to insert if a) the bitstream is encrypted or b) the unused FPGA resources are filled up with dummy logic in a bitstream. Trojans that have a connection to the original circuit are considered as type-II and require sufficient design knowledge to implement.

In 2016, Krieg et al. [6] presented an FPGA design flow attack that has been carried out by subverting the vendor's provided EDA tools. Their attack works in a two-stage manner by using compromised design tools. The malicious circuitry, i.e., Trojan, is first inserted by the synthesis tool while reading the design and activated when writing the bitstream. The implementation of a smart trigger using a malicious look-up table (LUT), which remains dormant throughout the design phase and activated when the final bitstream is written, helps the Trojan to evade all the design-time verification techniques relying on functional verification or rare events occurrence. In both the attacks it is believed that due to the lack of a verification mechanism for the bitstream configuration file it would be enormously difficult to counter such attacks. However, in their recent work Ahmed et al. [8] demonstrated a bitstream-level verification technique using proof-carrying hardware (PCH) which effectively detects the stealthy two-stage malicious LUT hardware Trojan attack presented in [6].

III. Threat Model

We consider the foundry as a trusted entity in our threat model, since even though some FPGA vendors are fabless and outsource device fabrication to a third party [2], attacking the FPGA fabric itself is less effective than for ASICs as the design to be implemented is loaded after the device is fully tested and shipped.

Trojan insertion by the design house and a third-party IP provider are considered as major threats for FPGA systems-on-chip (SoC), however, in our attack scenario we consider that both the design house and the IP vendor are trustworthy, but the EDA tools used by the design house are compromised by an attack; either by reverse-engineering the binaries of commercial EDA tools to insert malicious code, or via an insider in the EDA tools provider who maliciously swaps the legitimate binaries with malignant ones used for compilation by the design houses. We follow the threat scenario presented in [6], where the malicious code is inserted into an open-source tool that is then compiled to a binary version which in turn is used to intrude on the design house, ideally over the Internet, in order to replace the legitimate binary of the tool in the design house

with the malicious one to infect multiple machines of the design house in one go. However, the compromised EDA tools are not only limited to synthesis and place-and-route tools, but the tool that programs the FPGA could also be subverted to activate a Trojan inserted in the earlier stages. Fig. 1 highlights the entities involved for the development of the hardware module specified by the consumer. It can be seen that the design house itself is trusted but the EDA tools used by the design house are subverted by an attacker, as explained above, to gain control over the device when it is configured without being noticed by the producer or the consumer. In our threat model, the place-and-route tool and the FPGA programming tool are compromised to inject and activate the Trojan, and thus their binaries are marked as a red dotted box in the compromised EDA tool-chain in Fig. 1.

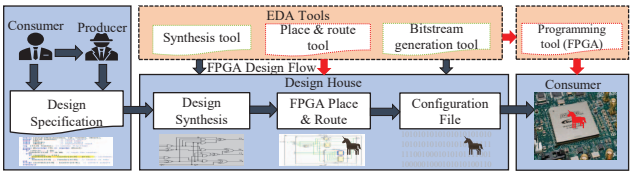

Fig. 1. Threat model: The red dotted boxes indicate compromised tools

We would also like to stress that our attack only activates the Trojan if the FPGA programming tool used by the consumer to program the FPGA device is compromised along with the place-and-route tool used by the design house and the configuration bitstream is either un-encrypted or the programming tool is capable to decrypt it, otherwise, the output of the infected tools will behave similarly as the original ones when the FPGA is programmed with the genuine programming tool. In this way, the attacker can conduct multiple targeted attacks by infecting only the programming tools of intended targets, which implies that the inserted Trojan will remain inert and therefore virtually undetectable in most of the customers' designs, thus there are fewer chances of the attack being revealed by chance.

IV. Methodology

A. Overview of Malicious Design Flow Attack

In our attack model, the Trojan is inserted in the second stage of the FPGA design flow, i.e., when the design's netlist is read by a malicious place-and-route tool (PnR) and is activated only in the FPGA device itself. The novelty of our attack lies in the fact that the routing tool disconnects the Trojan circuit from the original circuit before writing the bitstream and the FPGA programming tool again connects the Trojan circuit with the original circuit, thus activating the Trojan.

The general design flow of our attack for malicious insertion, routing, and activation is shown in Fig. 2. The attack works in two phases, i.e., in the first phase the Trojan circuit is injected, attached and then disconnected at one of the FPGA's programmable interconnect points (PIP) by the PnR tool. We call this temporary breaking point Trojan PIP (TPIP). This step is marked with a dotted red box in Fig. 2, to indicate that it is modified. In the second phase, i.e., the last stage in the design flow, the TPIP is activated again by the modified FPGA programming tool to connect the Trojan circuit to the original circuit.

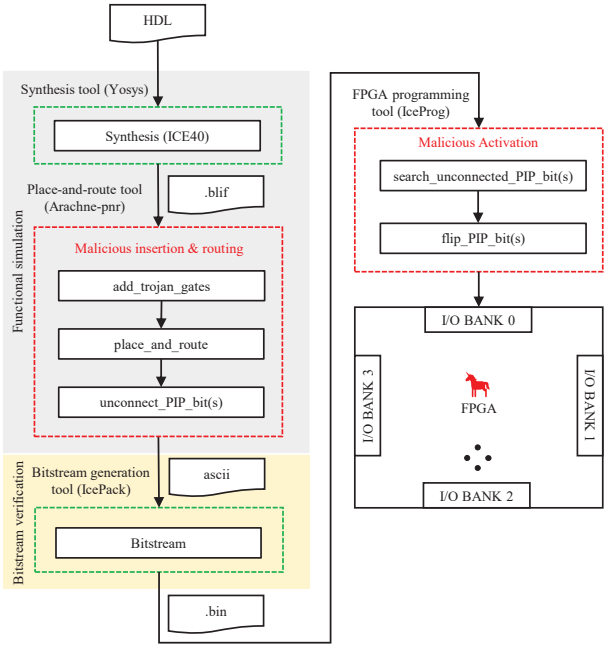

Fig. 2. FPGA design flow for malicious insertion and activation of a Trojan

Each of the stages of the design flow is explained in the following: The first step in our design flow, the synthesis of an HDL design by a synthesis tool, is not infected, and therefore the generated netlist remains unchanged compared to the design flow using pristine EDA tools. The next step of the design flow is compromised such that the Trojan circuitry is added to the synthesized netlist when it is read by the PnR tool, e.g., barrier gates who route secret information to primary outputs, but prevent it from leaking until the Trojan is activated. After placement and routing, the connection of the Trojan to the original circuit is removed by flipping the TPIP configuration bit to "0" so that the output at this stage behaves functionally equivalent to the original design, thus avoiding detection by any functional simulation and verification methods.

Note that in contrast to compromised tools targeting the RT level, the benefit of the post-synthesis Trojan insertion is that it still works if an IP core provided by a third party to a design house is a gate-level netlist. Even if the design is already a verified synthesized netlist, it can be infected in the next step. Furthermore, machine learning approaches based on reverse-engineering the bitstream to obtain a gate-level netlist for feature extraction mainly consider trigger nets [15], and thus cannot detect our Trojan circuit for two reasons: a) The information-leaking version is trigger-less, and b) in the general case, the payload of the Trojan is unconnected from its trigger in the bitstream, hence a reverse-engineered netlist would result in false negatives.

In the next step, a bitstream configuration file is generated

by the unmodified bitstream generation tool which carries the Trojan payload that stays unconnected at this stage, hence evading any bitstream verification mechanism such as PCH. In the last stage of the FPGA design flow, when the bitstream is loaded onto the FPGA by the programming tool, the connection of the Trojan to the original circuit is re-established by flipping the TPIP configuration bit again to "1". For functional Trojans, the circuit diagram of a simple design with inserted Trojan trigger and payload shown in Fig. 3 describes the internal schematic view of the design in a bitstream and the FPGA to better understand the attack. The switch between the trigger and the payload refers to the TPIP used to disconnect and connect the trigger, thus rendering the payload inactive and the Trojan dormant.

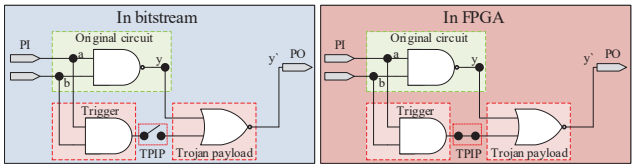

Fig. 3. Circuit schematic of an infected design in a bitstream and FPGA

Though our malicious flow is generic and any kind of Trojans can be implemented, an intelligent attacker would be interested to get higher level attacks and much control to the design such as secret key leakage from a cryptographic circuit without external resources. This can be done if the enable pin of the barrier gates is maliciously routed to one of the unused I/O pins and the connection through the TPIP is removed when the tool writes the output into a bitstream. The information of the unused I/O pin can be obtained by scanning the constraints file, therefore making this pin available for activating the Trojan in the final stage. After the connection of the I/O pin is re-established by the modified programming tool and the FPGA is configured, the attacker can apply voltage to the pin to activate the Trojan circuit which leaks the key through the output pins used by the original design instead of cipher text. The key leakage of an AES core by a Trojan payload is explained in Section V.

B. Flow of information between the compromised tools

A prerequisite for a successful attack is that the FPGA programming tool knows the location of the TPIP in the infected design. The attacker therefore has to communicate this location from the tool that chooses it to the tool where it is used, requiring them to create a hidden communication channel between the design house and the consumer site. The attacker can realize this communication explicitly, i.e., over a new channel, or implicitly, i.e., by hiding the information among the regularly transmitted data.

With explicit communication, the TPIP location is decoupled from the bitstream, and hence needs to be related within the programming tool in order to apply the correct TPIP flip for the loaded design. Depending on whether the attacker wants to target specific individual designs created at the design house, or rather infect all designs originating from there, they have to either transmit one location or create and query a database that maps design identifiers (e.g., design hashes) to TPIP locations. The former would require the attacker to closely monitor the designs that should be written in the near future, which would likely necessitate the continued presence of an agent on-premises, who could then also be leveraged to exfiltrate the information via human communication. The widespread infection of many designs, on the other hand, would generate more traffic over the hidden channels, increasing the chance of the channel being detected by the victims. However, to scale such an attack to multiple design houses and consumers, an attacker could then leverage existing botnet solutions to create a command-and-control infrastructure akin to modern software trojans, resulting in exactly the same advantages and disadvantages as in the software world, i.e., fast and easy scalability, high degree of automation and flexibility, versus vulnerability to automated, pattern-based communication-detection methods and a complete failure of the attack in a high-security environment, where the programming tool is not connected to the internet and thus cannot query TPIP locations.

Using implicit communication instead, the attacker can only rely on the one communication that the victims cannot avoid, which is the transmission of the bitstream. By reverse engineering the bitstream format used by both parties, the attacker could leverage unused portions of the stream to hide the TPIP location in plain sight, e.g., pockets of legacy information that are no longer in use for modern devices, or information that follow the end-of-stream and which would thus be ignored by the regular tools. This form of communication implicitly matches the TPIP location to a design and therefore does not require any method of design identification afterwards, but in order for the attacker to sustain such an attack, they would need to continue to reverse engineer bitstream formats and update their transmission code every time that a format is updated or the involved parties switch to a new format. However, since the attack itself also relies on modified binaries within the EDA suite, the attacker would have to update the compromised tools anyway in these cases.

V. Experimental Validation

A. Secret key leakage of an AES Core

As a proof of concept, we demonstrate our attack by implementing an AES core with an 8-bit data interface from [11] to an iCE40HX-1k device iCEstick Evaluation Kit provided by Lattice semiconductor. We use the open-source design flow from the project *Icestorm* [12] for iCE40 FPGAs, which consists of the *Yosys open synthesis suite* for hardware synthesis, *Arachne-pnr* for placement and routing, *IcePack* and *IceProg* for bitstream generation and programming the FPGA respectively.

To circumvent verifications at the gate level, we did not modify *Yosys*; hence the output at this stage is a legitimate synthesized netlist *(.blif)* of the AES design synthesized for iCE40 FPGAs, i.e., using `synth_ice40` command. After that when the compromised *Arachne-pnr* reads the netlist by invoking `read_blif`, it inserts the Trojan circuit into the

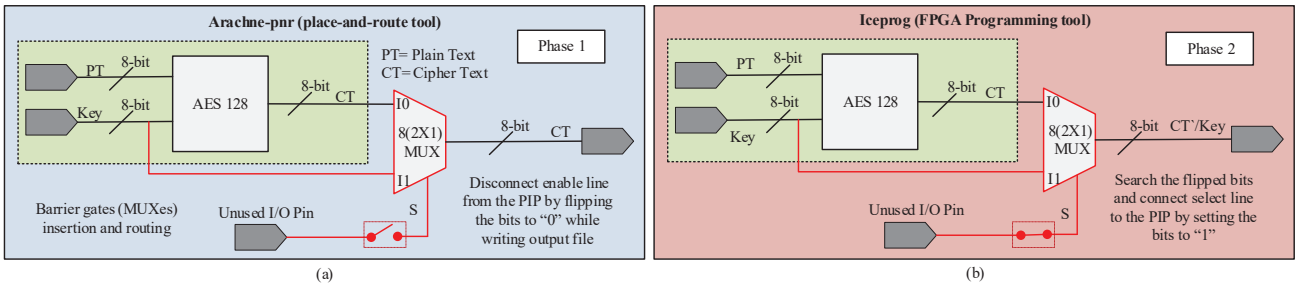

Fig. 4. Phases of Trojan insertion and activation in AES core by the compromised design flow tools

original circuit. The commands used in our flow are given in Listing 1, where the commands in rows three and five call the malicious script from the tool to infect the design.

An example for the two-phased process of inserting and activating an I/O-triggered key-leaking Trojan based on barrier gates into an AES core is shown in Fig. 4. Phase 1 in subfigure a) shows the barrier gate insertion and routing of the enable signal to an unused I/O pin, which is disconnected at a PIP, the TPIP, while writing the design output. Our barrier gates consist of eight `2X1 MUXes` connected to the very last register before the primary output.

```
1 yosys -p read_verilog example.v
2 yosys -p synth_ice40 -blif example.blif
3 arachne-pnr -d 1k -o example.asc icestick.pcf example.blif
4 icepack example.asc example.bin
5 iceprog example.bin
```

Listing 1. Commands used by the malicious design flow in our examples

The inputs to each `MUX` is the output of the register containing cipher text, and the secret key to be leaked. The key from any of the rounds can be leaked, however for the sake of simplicity, we take the key that is used as an input to the AES module.

Depending on the attacker's intentions the enable input (S) of the barrier gates in Fig. 4 could be attached in two different ways: a) A constant "1", or b) an unused I/O pin which is added by *Arachne-pnr* when it reads the constraints file (*.pcf)*. In the first case, after placement and routing, the TPIP connecting the enable line to the barrier gates is flipped when the output file is being written. In the second case, the infected design is placed and routed making sure that the enable line (S) of the barrier gates is routed to the newly assigned input pin and the address of the I/O tile containing the input pin is stored.

When the routed design is being written to a text file *(.ascii)*, the modified `write_text` function is invoked in the backend, which accesses the corresponding I/O tile and its connection to the barrier gates via the TPIP is removed. In the architecture of the iCE40HX-1k device, each I/O tile can have two I/O blocks, one for inputs and another for outputs with two local tracks. Each I/O block in a tile is connected to the other I/O or logic blocks with the help of PIPs called vertical and horizontal spans in the iCE40 family. To remove the connection of the enable line from the input pin, the PIP that connects the enable signal to the I/O tile is flipped to "0". Likewise, when the FPGA programming tool, *IceProg* configures the FPGA, the corresponding TPIP bit in the bitstream, communicated by the design flow, is flipped again to make the connection of the enable line to the barrier gates. In our example, the TPIP is located in I/O tile $(10, 17)$, corresponding to the bit at address `0x4743` in the bitstream.

Once the connection is made, the cyclic redundancy checksum (CRC) is updated and the FPGA is configured with the infected bitstream. Since one of the inputs to the barrier gates is a cipher text bit and the other is a secret key bit, the barrier gates act as buffers that pass the cipher text to the outputs when the enable line is disconnected or low. In the first case, the Trojan circuit will be activated and start leaking the key straight away, and can hence be accessed remotely or locally by the attacker. However, in the second case, the attacker needs physical access to the device to activate the Trojan circuit by giving voltage to the specified I/O pin used by the malicious tools, thus triggering the key leakage.

Although physical access is required, the attacker, in this case, has more control over the device, i.e., switching between the original circuit and the Trojan circuit using an input pin to leak the secret key. Therefore, the attack is more difficult to be revealed in the field as compared to the first case where the output of the implemented design will always be a malicious one. Nevertheless, in both cases, we have successfully performed the attack to leak the 8-bit secret key in our example. In order to illustrate, we have used the five available LED's on the device and connected three external LED's to read out the leaked key byte.

B. Demonstration example

To assess that our barrier gates inserted and connected properly, we have synthesized the example design given in Listing 2 with Yosys and then gave it to the compromised PnR tool. We have read out the example design in a text file from *Arachne-pnr* before enabling the malicious back-end script that is used to disconnect the TPIP. The left part of Fig. 5 shows this design as visualized by the ICE40 layout viewer [13]. Next, we enabled the back-end script and compiled the tool to a final compromised version. Using a synthesized netlist of an AND-gate as an example design for the compromised PnR tool, we have obtained the design depicted on the right-hand side of Fig. 5, confirming the successful removal of the enable line after the TPIP has been flipped. After this successful subversion

of the PnR tool, we have modified the programming tool and then have programmed the FPGA with it to see the output of the design. We have verified the output with both, the original version of the programming tool and the compromised version, where the output of the former is not affected when the enable line is high while the latter then leaks the input "*b*" at the outputs.

```
1 module top (input a, b, output y);
2 assign y = a & b;
3 endmodule
```

Listing 2. Original Verilog code for AND-gate example

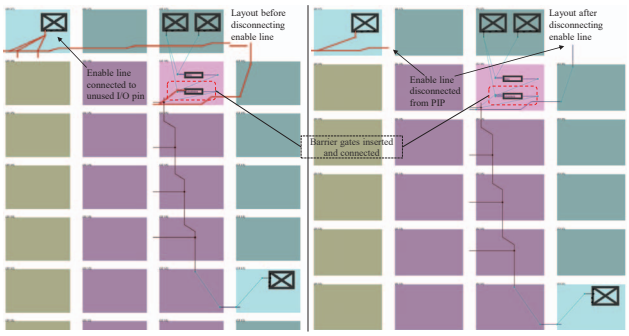

Fig. 5. ICE40 Layout Viewer to evaluate the barrier gates insertion and disconnection by a compromised PnR tool in AND-gate design

VI. Discussion

When we talk about Trojans in FPGAs, the common question discussed is the way Trojans are inserted into the design such that it cannot be detected by conventional testing methods. In this regard, we present a hardware Trojan attack employing compromised EDA tool flow in which the Trojan is inserted into a design during the place-and-route step and is activated while loading the bitstream configuration file into the FPGA. Note that we use an open-source tool flow to present our attack, however, in principle, the attack would work for the commercial tools provided by other FPGA vendors, e.g., Xilinx, if there is access to the code. Our attack is versatile and can be implemented according to the attacker's desire and suited scenario, for example, the attack presented in [6] can also use our compromised tool flow to hide the malicious LUT Trojan even in the bitstream to evade bitstream-level detection mechanisms. Also, in literature, a lot of effort has been given to design a Trojan which could assist in leaking the secret key through side channels [5]. However, our attack is subtle yet effective in providing the attacker with a means to leak the secret key directly through the primary outputs at runtime, i.e., opening a covert channel. On the other hand, if we discuss the detection of sneaky Trojans, a trigger-less implementation of a Trojan in our attack renders pre-and-post-synthesis detection mechanisms based on static or dynamic trigger characteristics ineffective [7], [14]. Power side channel analysis techniques inspect and visualize the power traces obtained from the design running on the device with the golden one to detect the Trojan. In our key leakage example no logic is implemented as trigger, and the payload consists of only a few gates, i.e., eight `2X1 MUXes`, hence the extra power consumption by an infected design should be negligible. To reveal our attack, post-configuration methods such as the `ReadBack` feature of the FPGA [3], if enabled, can be used at the consumer side to read out the bitstream and apply verification mechanisms such as functional equivalence checking using PCH.

VII. Conclusion and Future Work

We have presented a new attack in this paper that is based on the malicious routing of an inserted Trojan circuit to retain a dormant state even in the bitstream for a reconfigurable hardware device. For the information-leaking variant, the inserted Trojan circuit in our approach does not even need any trigger logic as the payload is maliciously routed to the primary outputs. In all variants, the last programmable interconnect point (PIP) connecting the payload to a trigger or an enable signal is removed during place-and-route and is established again only at the very last step when the FPGA is being programmed, thus activating the Trojan circuit. We have demonstrated our attack on an 8-bit AES design to successfully read out the first eight key bits of an AES module. The conventional testing and verification methods presented at the RTL or gate level so far cannot prevent or detect our proposed attack as the Trojan is injected in a post-synthesis step of the design flow, thus circumventing conventional testing and verification methods. Furthermore, we have shown that our Trojan circuit stays unconnected in the bitstream, thereby evading even bitstream-level verification techniques. In our future work, we aim to investigate possible countermeasures for the presented attack such as post-configuration verification with the help of readback feature in FPGAs.

References

[1] S. Wallat et al., "A look at the dark side of hardware reverse engineering - a case study", In IEEE 2nd IVSW, 2017, pp. 95-100.
[2] S. Bhunia et al. "Hardware trojan attacks: Threat analysis and countermeasures," In Proceedings of the IEEE 102(8), 1229-1247 (Aug 2014).
[3] S. M. Trimberger et al., "FPGA Security: Motivations, Features, and Applications," In Proceedings of the IEEE 102(8), 1248-1265, Aug(2014).
[4] S. S. Mirzargar, et al., "Physical Side-Channel Attacks and Covert Communication on FPGAs: A Survey," In (FPL), 2019, pp. 202-210.
[5] M. Ender et al., "The First Thorough Side-Channel Hardware Trojan," In Advances in Cryptology – ASIACRYPT 2017.
[6] C. Krieg et al., "Malicious LUT: A stealthy FPGA Trojan injected and triggered by the design flow," In ICCAD, 2016, pp. 1–8.
[7] Adam Waksman et al., "FANCI: identification of stealthy malicious logic using boolean functional analysis," In Proceedings of the 2013 ACM SIGSAC (CCS '13), 697–708.
[8] Q. A. Ahmed et al., "Proof-Carrying Hardware Versus the Stealthy Malicious LUT Hardware Trojan," In (ARC) 2019.
[9] A. Duncan et al., "FPGA Bitstream Security: A Day in the Life," In (ITC), 2019, pp. 1-10.
[10] R. S. Chakraborty et al., "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream," In IEEE Design Test, 2013.
[11] "8bit datapath AES," https://github.com/ChengluJin/8bit_datapath_AES.
[12] C. Wolf. et al., "Project icestorm," http://www.clifford.at/icestorm/.
[13] K. Nielsen, "ICE40 Layout Viewer," https://github.com/knielsen/ice40_viewer.
[14] J. Zhang et al., "VeriTrust: Verification for hardware trust," 50th ACM/EDAC/IEEE (DAC), 2013, pp. 1-8.
[15] T. Zhang et al.,"A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code," in IEEE Access, 2019.