Analysis of Attack Surfaces and Practical Attack Examples in Open Source FPGA CAD Tools

Sandeep Sunkavilli, Zhiming Zhang, and Qiaoyan Yu
Dept. of Electrical and Computer Engineering, University of New Hampshire
Durham, NH 03824, USA
Email: Qiaoyan.Yu@unh.edu

Abstract—**FPGAs gain increasing utilization in system prototyping, low-volume products, and obsolete component replacement. Driven by high profits, FPGA deployment is suffering from various attacks, such as reverse engineering bitstream, functionality tampering via hardware Trojans, information leaking through covert channels, and denial-of-service attacks. Typically, the investigation of security threats on FPGA deployment is tied with a specific FPGA chip and its design suite version, either the discovered attacks or the developed countermeasures are not easy to migrate to other FPGAs. Thus, the utilization of open source FPGA CAD tools becomes increasingly attractive. This work analyzes the new attack surfaces on two open source FPGA CAD tools: VTR and Symbiflow. The case studies in this work indicate that practical attacks in open source FPGA computer-aided-design (CAD) tools can be implemented with minor changes on the intermediate files generated by the CAD toolchain.**

Index Terms—**FPGA, open source CAD tool, place and route, hardware security, hardware Trojan, bitstream.**

I. Introduction

FPGAs are prevalent in system prototyping, hardware implementation for low-volume products, and the replacement of obsolete components in legacy systems [1], [2]. As highlighted in the work [3], FPGA security gains increasing attention [4]. There are extensive research efforts on the secure operations conducted by FPGA devices and safe bitstream delivery. Driven by the large market profit, attackers are motivated to degrade the performance of FPGA based systems by using counterfeit FPGA chips [5], pirate the hardware description of FPGA configuration by reverse engineering the bitstream [6], or tamper with the FPGA configuration via malicious FPGA Computer-Aided-Design (CAD) tools.

Among the various attacks on FPGAs, the one conducted in untrusted FPGA CAD tools are the most challenging to address. As the FPGA bitstream generation process is not transparent to FPGA users, malicious modifications made in the intermediate stages of the bitstream generation process are difficult to notice and detect. Moreover, for a given FPGA device, there is no golden bitstream reference available for verification. The security threats originated from vulnerable FPGA CAD tools have been reported in handbooks, case demonstrations, and research articles. For instance, untrusted FPGA CAD tools can be exploited by attackers to insert hardware Trojans [7], [8]. The work [9] showcases three attack surfaces on Xilinx ISE, including the output port of mapping, place & route, and bitstream generation. That work also indicates that the Altera FPGA design suite, Quartus, leaves similar backdoors for hardware Trojan insertion. The work [10] proves that it is also practical to extract FPGA IPs without interrupting the original logic function from commercial FPGA CAD tools.

As the investigation of security threats on FPGAs is typically tied with a specific FPGA chip and its design suite version, either the discovered attacks or developed countermeasures are not easy to migrate to other FPGAs. Thus, the study of security threats on open source FPGA CAD tools will be more valuable to promote the development of generalized methods for attack-resilient FPGA designs. To fulfill this purpose, this work analyzes the new attack surfaces on open source FPGA CAD tools and provides some attack examples. More specifically, this work makes the following contributions:

- A generalized attack flow is proposed to implement stealthy attacks on open source FPGA CAD tools. We expect that our attack flow will inspire more researchers to be aware of the new emerging attack methods in both commercial and open source FPGA CAD tools.
- We analyze the open source FPGA tool, *Verilog to Routing (VTR)*, to reveal the potential attack surfaces. With practical and successful attack examples, we showcase the process of altering the place and blif output files to harm the original delay and logic allocation on FPGAs.
- We further extend the security vulnerability analysis to another open source bitstream generation tool, *Symbiflow*. The impact of the proposed attacks on the critical and non-critical path delay is assessed.

The rest of this work is organized as below. Section II introduces two open source FPGA CAD tools, VTR and SymbiFlow. Section III presents the main challenges on analyzing the security threats from FPGA CAD tools and also characterizes the attacks implemented on open source tools. Sections IV and V propose the detailed attack flow and examples designed in VTR and SymbiFlow. In Section VI, we suggest possible defense strategies. This work is concluded in Section VII.

II. Preliminaries

A. *Open Source FPGA CAD Tool: VTR*

Open source FPGA CAD tools enable us to investigate the impact of various FPGA architectures and CAD algorithms on FPGA configuration. VTR is open source software that generates net, place and route files. The design flow of VTR needs two inputs: the hardware description of a circuit (Verilog) and an FPGA architecture description file (EARCH.xml). As

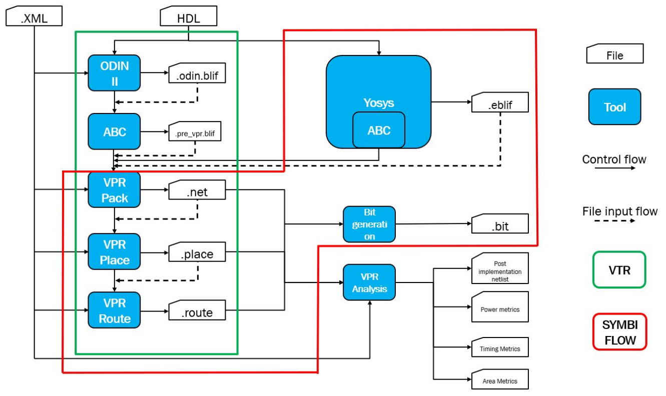


Fig. 1. Open Source CAD flow.

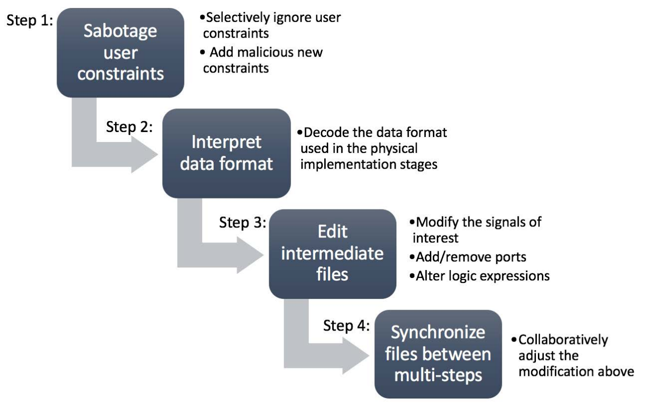

Step 1:
Sabotage user constraints
•Selectively ignore user constraints
• Add malicious new constraints
Step 2:
Interpret data format
•Decode the data format used in the physical implementation stages
Step 3:
Edit intermediate files
•Modify the signals of interest
•Add/remove ports
•Alter logic expressions
Step 4:
Synchronize files between multi-steps
•Collaboratively adjust the modification above

Fig. 2. Proposed general attacks on open source FPGA CAD tools.

shown in Fig. 1, the entire toolchain of VTR is composed of three tools: ODIN II, ABC, and Versatile Place and Route (VPR). Among those tools, ODIN II is equivalent to Yosys used in the toolchain of Symbiflow.

As an elaboration and synthesis tool, ODIN II reads the Verilog file and synthesize it into a netlist (a file ends with the extension *.odin.blif*). The tool ABC performs logic optimization and technology mapping based on the *.odin.blif* file, producing two blif files *.abc-no-clock.blif* and *.pre-vpr.blif*. The latter one is the final and complete netlist file for the given Verilog description. At the final stage of VTR, the tool VPR is responsible for packing, placement, routing, and timing analysis. The output of packing is saved in a *.net* file. The placement information is available in a *.place* file. The routing output is stored in *.route* file. In this work, we study possible attack surfaces in VTR that could be exploited by attackers to harm the integrity of a general FPGA design flow.

B. Open Source FPGA CAD Tool: Symbiflow

Symbiflow is an end-to-end open source FPGA synthesis tool that converts a Verilog file to a bitstream file for FPGAs. Currently, Symbiflow can support only Xilinx 7-Series, Lattice iCE40, and Lattice ECP5 FPGAs. The tools Yosys, ABC, VPR, nextpnr, and open FPGA assembler are used to produce the bitstream file of the desired FPGA board. Yosys is a synthesis tool used to convert a Verilog description to the corresponding hardware netlist and write it to a *.eblif* file. Next, the Place and Route (PnR) tool processes the .eblif file to generate a physical implementation for a specific FPGA device. There are two PnR tools in Symbiflow: VPR for Xilinx-7 Series and nextpnr for Lattice iCE40. All the practical attacks performed in this work are based on the Xilinx-7 Series FPGA. Both VPR and nextpnr tools write their results to a *.fasm* file, which will be further used to generate a bitstream file. The VPR tool also produces .net, .place, and .route files. In addition, Symbiflow allows users to perform simple analysis on critical-path delays.

III. Proposed General Attack Flow on Open Source FPGA CAD Tools

A. Importance and Challenges

The investigation on open source CAD tools will provide FPGA users with better understanding on the potential security threats originated from the untrusted or third-party CAD tools. With the highlighted caution, FPGA users should reconsider their FPGA deployment (from specification, system architecture, implementation, to verification and authentication strategies) and develop necessary proactive defense mechanisms.

The main challenges of revealing the security vulnerabilities on FPGA CAD tools include the facts as follows.

(1) Limited disclosure is available due to the protection for commercial profits. This is especially the case for the commercialized FPGA vendors.
(2) The nontransparent synthesis, placing, and routing algorithms adopted in the CAD tools make it difficult for FPGA users to differentiate the malicious design modifications induced by the untrusted CAD tools from normal optimizations.
(3) Most of the intermediate outputs in the FPGA compiling and configuration flow are not readable. This fact increases the difficulty for FPGA users to timely perceive the abnormal operations embedded by malicious FPGA tools.
(4) To generate a bitstream file through open source tools, the toolchain may use multiple open source software. Some malicious software from untrusted third parties may be mingled with the official FPGA CAD tools. The use of multiple open source software could render to new attack surfaces other than the known security vulnerabilities of each individual tool.
(5) As FPGA CAD tools are often updated, it is difficult to keep the same update pace to investigate the new security threats and revise the existing countermeasures accordingly.
(6) Different with the commercial FPGA design suites, the current open source FPGA CAD tools only have very limited functionality for power and timing analysis tools. Consequently, open source tool users will not be able to zoom in the security issues and assess the potential security risk.

B. Generalized Attack Flow in Open Source FPGA CAD Tools

Despite diverse FPGA CAD tools using different interfaces in the process of design compiling and bitstream generation, we abstract the common steps that a typical attack will take

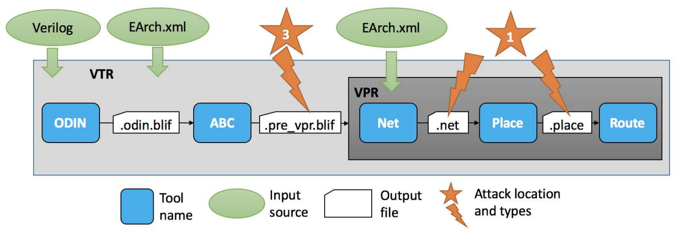


Fig. 3. Overview of attack surfaces on VTR.

and summarize them in Fig. 2. In step 1, an attack performed via FPGA CAD tools will selectively incorporate the user constraints, either ignoring the user's constraints or stealthily adding new constraints, so that the FPGA configuration could be modified covertly. In step 2, attackers need to fully understand the format of the intermediate output files (e.g., .blif) and foresee what changes on the intermediate files can fulfill the intended attack purpose. The core attacks on FPGAs will take place in step 3. One could modify I/O and also alter the Boolean expressions indicated by the hardware description language. Step 4 is the most challenging one as the success of bitstream generation requires the modified intermediate files to be acceptable in other phases of the CAD flow. Attackers need to collaboratively adjust the malicious modifications so that the intermediate files are synchronized in the entire FPGA configuration flow. In an open-source FPGA CAD tool, that fact that all the intermediate stages whose input or output file open for editing will introduce potential attack surfaces.

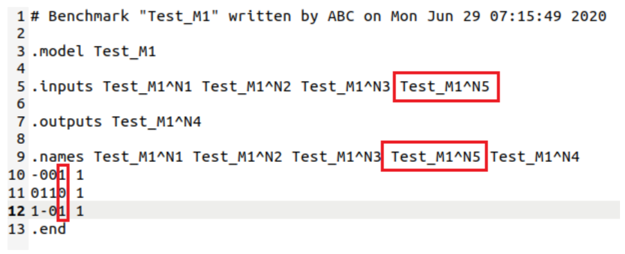
IV. Attack Surfaces on VTR

In this section, we use VTR as an example to show what potential attack surfaces that can be exploited to implement practical attacks on open source FPGA CAD tools. After understanding the file format of each tool in the VTR toolchain, we identify three attack surfaces as shown in Fig. 3: *.blif*, *.net*, *.place* files after **ABC**, **Net** and **Place**, respectively. More precisely, we have successfully realized three types of attacks to alter input ports, output ports, and logic truth tables.

A. Potential Attack Surfaces

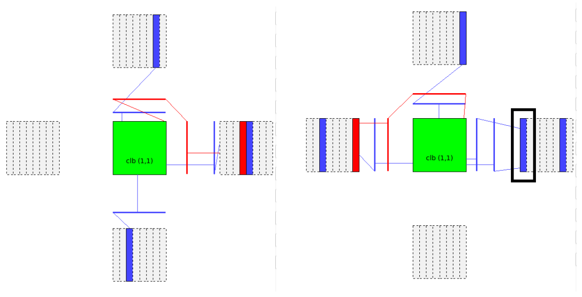
1) .blif File: A Berkeley Logic Interchange Format file (.blif) describes the logic level hierarchical circuit in a textual format. Figure 4(a) shows an example of a .blif file for a design under attack. The .blif file consists of four important parts: *.model* (the module name, e.g., Test_M1), *.inputs* (all the input pins for the module), *.outputs* (all the output pins), and *.names* (the complete list of signals involved in a particular output logic e.g., N4). All those four parts can be the target of attacks conducted on the .blif file.

2) .net File: A .net file keeps the same information indicated in a .blif file and further includes block names, subblocks, instances, modes, and clocks. The information inside the block is populated based on the Architecture file (EArch.xml) being used. If only the .net file is modified, the FPGA CAD tool will detect the mismatches between the original .blif and the tampered .net files, generating an error message to warn

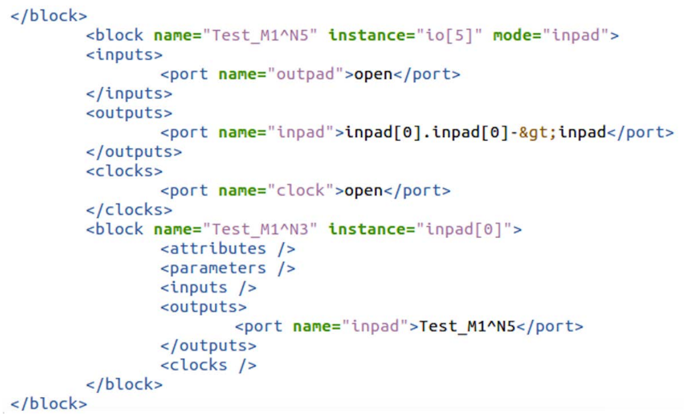


```
# Benchmark "Test_M1" written by ABC on Mon Jun 29 07:15:49 2020

.model Test_M1

.inputs Test_M1^N1 Test_M1^N2 Test_M1^N3 Test_M1^N5

.outputs Test_M1^N4

.names Test_M1^N1 Test_M1^N2 Test_M1^N3 Test_M1^N5 Test_M1^N4
-001 1
0110 1
1-01 1
.end
```

(a)



(b) (c)



```
</block>
        <block name="Test_M1^N5" instance="io[5]" mode="inpad">
        <inputs>
                <port name="outpad">open</port>
        </inputs>
        <outputs>
                <port name="inpad">inpad[0].inpad[0]-&gt;inpad</port>
        </outputs>
        <clocks>
                <port name="clock">open</port>
        </clocks>
        <block name="Test_M1^N3" instance="inpad[0]">
                <attributes />
                <parameters />
                <inputs />
                <outputs>
                        <port name="inpad">Test_M1^N5</port>
                </outputs>
                <clocks />
        </block>
</block>
```

(d)

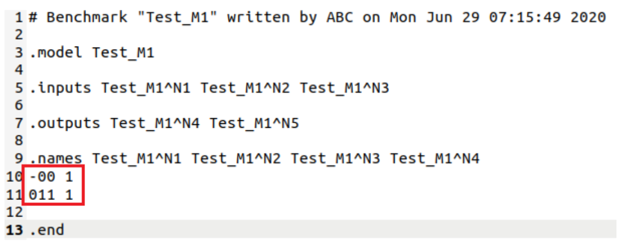
Fig. 4. Implementation of the input attack on the .blif and .net files. (a) Tampered .blif file (modified portion highlighted in red boxes), (b) graphical view of the mapped circuit before attack, (c) graphical view of the mapped circuit after attack, and (d) equivalent modification on the .net file.

FPGA users. As a result, the following placing and routing will be halted. However, if the .blif and .net are corrupted collaboratively, the attack will go through the integrity check built in the CAD tool.

3) .place File: A .place file has information about where the blocks will be placed in the FPGA fabric. Three types of attacks could be conducted by altering the .place file: X and Y coordinates for the position of the block, and subblock number. The tampered coordinates will guide the VPR to re-route the input and output pins to some places, where it will be easier for attackers to probe and develop covert channels later.

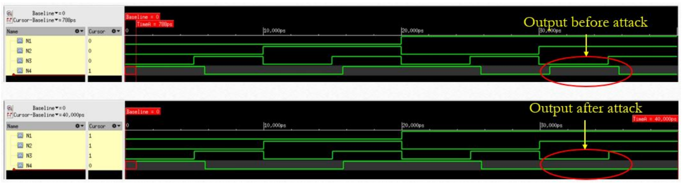
B. Basic Attack Implementation

1) Attack on Inputs and Outputs: In this attack, the .blif file could be modified to create a new input pin in the circuit to alter the original functionality or add new logic. This attack will be succeed if the attacker changes .inputs and .names as shown



```
1 # Benchmark "Test_M1" written by ABC on Mon Jun 29 07:15:49 2020
2
3 .model Test_M1
4
5 .inputs Test_M1^N1 Test_M1^N2 Test_M1^N3
6
7 .outputs Test_M1^N4 Test_M1^N5
8
9 .names Test_M1^N1 Test_M1^N2 Test_M1^N3 Test_M1^N4
10 -00 1
11 011 1
12
13 .end
```

(a)



(b)

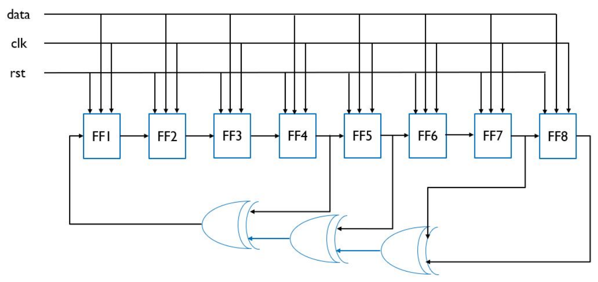
Fig. 5. Implementation of the logic attack on .blif file. (a) Modified .blif file due to the attack on logic description, and (b) waveform showing the change on logic table leading to tampered output.

in Fig. 4(a). The fourth column is added to the lines 10 to 12. Although the VTR tool checks for the file integrity when that .blif file is sent to VPR for packing, placement, and routing, the attack described above can successfully render to the result shown in Fig. 4(c). Comparing the baseline circuit mapping before the attack shown in Fig. 4(b), we can see that the attack introduces one more input pin (we have four blue blocks now, instead of three). The number of output pins (in red) remains the same but the location is shifted, as well. Attackers can follow the similar procedure to add new output pins to the .blif file. The tampered .net file will be similar with the one shown in Fig. 4(d). A block will be added for the newly introduced pin. The attack via the .net file is less readable (thus more stealthy) than the attack via the .blif file.

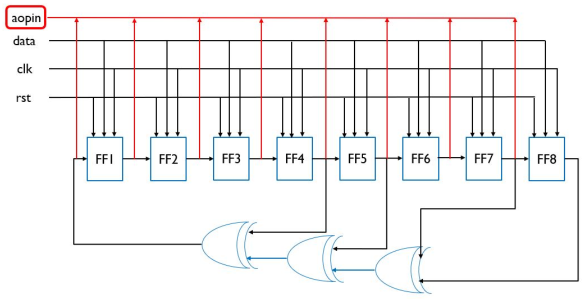
2) Attack on Logic Truth Table: Another modification adversaries could perform is sabotaging the original logic truth table. This is practical for an attacker who have a good understanding on the target design. The truth table is defined in the .blif file under the line starting with the keyword *.names*. Theoretically, one can remove/add one row or revise the logic in the original table in the attack. Due to the built-in integrity check in VTR, the output of VPR will only be accepted if the attack on the logic truth table removes some rows, instead of adding new rows. We resume the same baseline example used in Fig 4(a) to implement the attack on logic description. As shown in Fig. 5(a), the logic expression on line 12 is removed. Consequently, the output of N4 is altered by the attack. The red circles in Fig. 5(b) highlight the change in outcome of the proposed attack. Another interesting observation we notice is, VTR does not have a capability to check if the logic description is modified. Functional verification (conducted in other tools) is necessary to detect the attack on logic truth table.

C. Practical Attack on 8-bit Linear Feedback Shift Register

Linear Feedback Shift Register (LFSR) is often used to generate random numbers for cryptographic modules. Depending on the feedback paths, different random number sequences can be shifted from the serially connected registers. We employ the basic attack implementation introduced in Section IV-B to perform a practical attack on a 8-bit LFSR such that the generated random numbers are confined in a limited range.

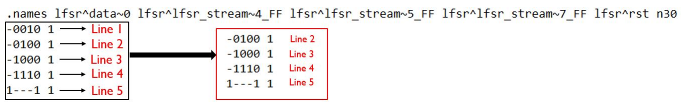


(a)



(b)

Fig. 6. Impact of FPGA CAD attacks on an 8-bit LFSR circuit schematic. (a) LFSR before attack and (b) LFSR after attack.

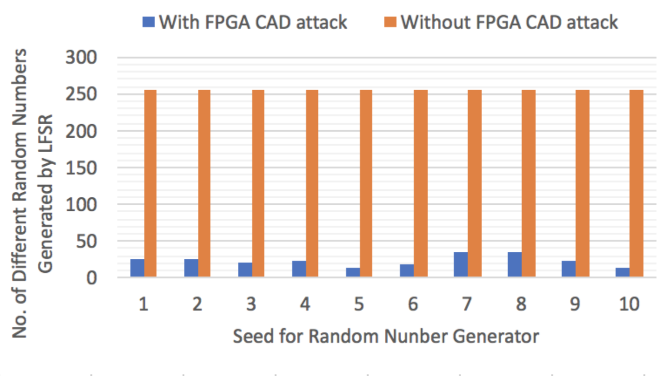


```
.names lfsr^data~0 lfsr^lfsr_stream~4_FF lfsr^lfsr_stream~5_FF lfsr^lfsr_stream~7_FF lfsr^rst n30
-0010 1 ⟶ Line 1
-0100 1 ⟶ Line 2
-1000 1 ⟶ Line 3
-1110 1 ⟶ Line 4
1---1 1 ⟶ Line 5

-0100 1   Line 2
-1000 1   Line 3
-1110 1   Line 4
1---1 1   Line 5
```

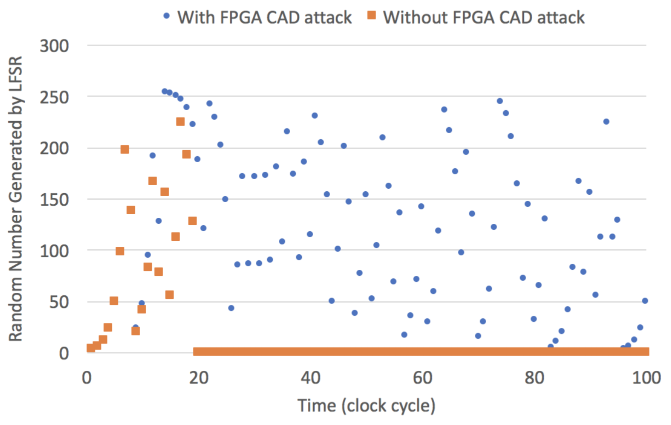
Fig. 7. LFSR feedback logic described in the .blif file. Note that the attack from the FPGA CAD tool removes Line 1, a part of the feedback loop logic.

Figure 6 shows the schematic of the LFSR design before attack. Normally, this LFSR can produce 255 different random numbers. Through the tool VTR, we can successfully implement the pin creation attack and logic attack. The former attack adds a new output pin in the .blif file for the LFSR to leak the random numbers being generated. The pin creation attack on the .blif file changes the LFSR circuit, the schematic of which is shown in Fig. 6(b). Note that the new output pin, *aopin*, is added in the LFSR module to manipulate the feedback loop. The logic attack sabotages the normal function of the LFSR. Figure 7 shows that Line 1, which specifies the feedback logic, is modified by the malicious FPGA CAD tool.

The consequence of the combination of pin insertion and logic modification attacks is illustrated in Fig. 8. As shown in Fig. 8(a), the dynamic range of the random numbers generated by the tampered LFSR is significantly smaller than that for the original LFSR. Moreover, the diversity of the random numbers due to the FPGA CAD attack is decreased dramatically. As shown in Fig. 8(b), the LFSR suffering from the FPGA CAD attack only generates a limited number of distinct random



(a)



(b)

Fig. 8. Impact of pin addition and logic modification attacks from the malicious VTR on the random numbers generated by the 8-bit LFSR.

numbers; in contrast, the LFSR without the attack can produce evenly distributed random numbers in the range of 0 and 255.

V. Attack Surfaces on Symbiflow

A. Potential Attack Surfaces

The FPGA CAD tool *SymbiFlow* has more capabilities than VTR and it is capable of generating executable bitstreams. We have identified two attack surfaces in the Symbiflow toolchain, as shown in Fig. 9. The *.eblif* (generated by Yosys) and *.place* files (produced by the Place tool in VPR) are vulnerable to the attacks from FPGA CAD tools. An Extended Berkely Logic Interchange Format (.eblif) file is similar with a .blif file except for a few changes in the structure. All the attacks performed on .place file in the VPR tool are also implementable in Symbiflow since VPR is embedded in the Symbiflow toolchain.

B. Basic Attack Implementation on .place

The basic attack implementations discussed in Section IV-B are applicable in Symbiflow. Another basic attack can be realized in SymbiFlow is tampering with the .place file. The FPGA fabric location for the blocks can be modified by changing the coordinates in the .place file. For instance, as shown in Fig. 10, the y coordinates of a block N1 are changed from 104 to 124. Although Symbiflow does not have graphical view, it can report holding and setup timing for the generated bitstream. As shown in Table I, the malicious modification on the N1 block's Y-coordinates in the .place file leads to different time slacks on holding and setup time. The affected time slack could be positive or negative, compared to the original ones. A negative time slack is difficult to detect since the attack does not influence the worse-case delay. We also examined the impact of this attack on two benchmark circuits s298 and s15480. The results shown in Table II indicate that our attack has negligible impact on the critical-path delay but could result in large changes on the non-critical paths.

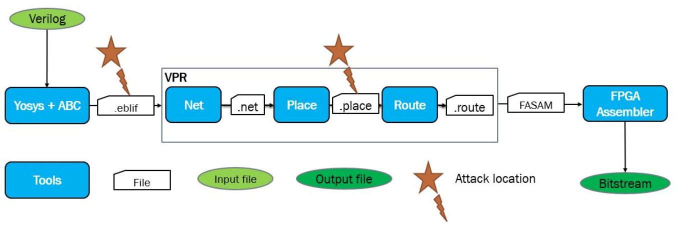
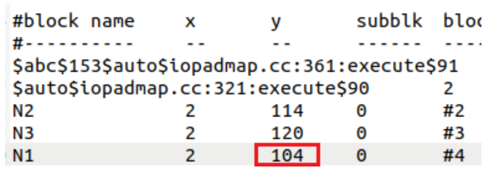
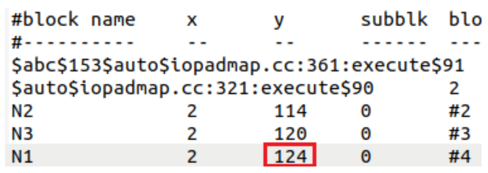


Fig. 9. Overview of attack surfaces on Symbiflow.



```
#block name     x       y       subblk  bloc
#----------     --      --      ------  ----
$abc$153$auto$iopadmap.cc:361:execute$91
$auto$iopadmap.cc:321:execute$90        2
N2              2       114     0       #2
N3              2       120     0       #3
N1              2       104     0       #4
```

(a)



```
#block name     x       y       subblk  blo
#----------     --      --      ------  ---
$abc$153$auto$iopadmap.cc:361:execute$91
$auto$iopadmap.cc:321:execute$90        2
N2              2       114     0       #2
N3              2       120     0       #3
N1              2       124     0       #4
```

(b)

Fig. 10. Implementation of an input attack on .place file. (a) Original .place file before attack, (b) the .place file after attack.

TABLE I
Timing slack affected by the attacks on Symbiflow.

Case ID	*Y-Coordinates*	*Holding time*	*Setup time*
Original	104	2.214 ns	2.418 ns
Attack 1	124	2.418 ns	-3.011 ns
Attack 2	134	2.214 ns	-3.424 ns
Attack 3	144	2.214 ns	-3.631 ns
Attack 4	154	2.214 ns	-3.913 ns

TABLE II
Timing results for different circuits after attacking .place file

Circuit	**.place File**	**Holding Time**		**Setup Time**	
		Critical	Non Critical	Critical	Non Critical
S298	Normal Operation	3.046 ns	1.731 ns	-4.201 ns	-0.464 ns
	Under Attack	3.046 ns	1.415 ns	-4.185 ns	-0.637 ns
	Under Attack	3.046 ns	1.371 ns	-4.185 ns	-0.464 ns
S15850	Normal Operation	-0.585 ns	0.75 ns	-8.966 ns	-3.046 ns
	Under Attack	-0.708 ns	0.345 ns	-8.966 ns	-2.938 ns
	Under Attack	-0.908 ns	0.343 ns	-8.966 ns	-2.923 ns

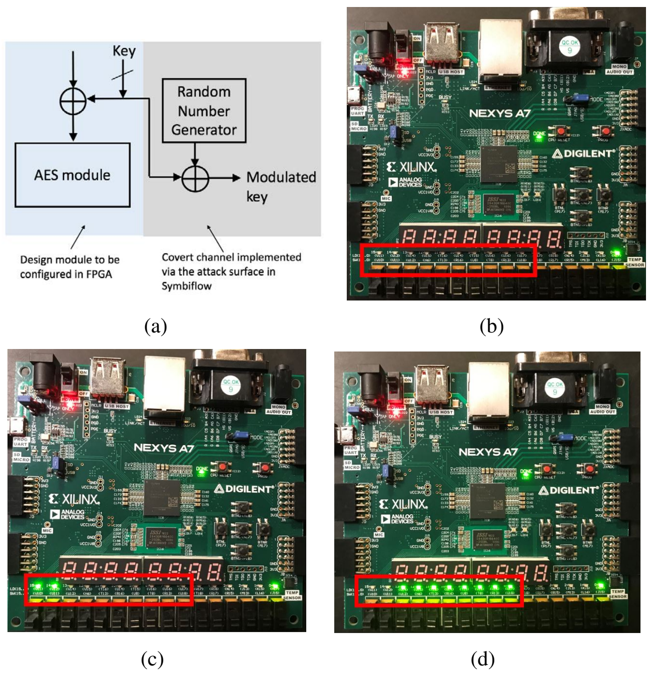

Key
Random Number Generator
AES module
Modulated key
Design module to be configured in FPGA
Covert channel implemented via the attack surface in Symbiflow
(a)
(b)
(c)
(d)

Fig. 11. FPGA implementing a covert channel on AES. (a) Schematic of tampered AES design, (b) FPGA running the normal AES operation, (c) and (d) FPGA running the AES with a covert channel leaking the cryto key information at different moments.

C. Practical Attack on AES

In this section, we demonstrate how the malicious FPGA CAD tool can implement a covert channel in an AES encryption module as shown in Fig. 11(a). We assume that the CAD tool has an internal library, which includes the Trojan design that modulates the secret key used in the encryption module. The CAD tool scans the .eblif file for the AES module and then adds the Trojan logic to that .eblif to form a covert channel. Once the modified .eblif is ready, two more files are changed to ensure the successful generation of bitstream file for the targeted FPGA (In our experiment, we used Digilent Nexys Artix-7 FPGA board). The constrains file (.pcf) and the make file are changed to assign all the ports to the FPGA board and required instructions are passed to the toolchain to generate the bitstream file. All the old .net, .place, .route, .FASAM and .bit files are removed from the directory. Figure 11 (b) shows the normal operation of AES encryption module. Figures 11(c) and (d) show the information leaked through the covert channel implemented through Symbiflow.

VI. Suggestion for Defense Strategies

To mitigate the potential attacks in the open source FPGA CAD tools, both FPGA software developers and FPGA users need to employ some proactive defense mechanisms. The FPGA software developers can design some security features in the CAD tools to protect the integrity of intermediate files. For example, the intermediate files can be encrypted or randomized. Without the knowledge of internal encryption key and randomization seed, the malicious software attached to the original FPGA CAD tools cannot precisely perform modifications to achieve the intended attack purpose.

FPGA users can obfuscate the Verilog-level designs to mislead attackers. For example, dummy logic or dummy components can be inserted to the original design. Without a deep understanding on the original design, the malicious modifications might land in the dummy logic and thus the attack from the FPGA software will not cause any malfunctions. Manipulating the dummy components, such as fake pins, cannot leak information successfully.

VII. Conclusion

FPGA security becomes a big concern in the FPGA deployment. As more and more third-party FPGA IPs and their associated tools are integrated into the FPGA CAD toolchain, more security threats will challenge the integrity and attack resilience of FPGA designs. Due to the limited disclosure on the security vulnerabilities of FPGA CAD tools, the countermeasure development for FPGA security grows slowly. To facilitate more efficient and effective countermeasure designs, we analyze two representable open source FPGA CAD tools to identify the potential attack surfaces, which could be exploited by attackers to harm the integrity of FPGA compiling and configuration process. Furthermore, we propose the basic attack implementation methods on those attack surfaces and provide practical attack examples. Our case studies confirm the feasibility of our generalized attack flow. In future work, we will explore feasible countermeasures to assure the integrity of the FPGA intermediate configuration files and investigate effective defense methods to thwart the attacks from malicious FPGA CAD tools.

References

[1] D. Hallmans, K. Sandström, T. Nolte, and S. Larsson, “A Method and Industrial Case: Replacement of an FPGA Component in a Legacy Control System,” in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 208–214.

[2] Z. Zhang, L. Njilla, C. Kamhoua, K. Kwiat, and Q. Yu, “Securing FPGA-Based Obsolete Component Replacement for Legacy Systems,” in *Proc. ISQED18*, 2018, pp. 401–406.

[3] S. Drimer, “Security for Volatile FPGAs,” University of Cambridge, Computer Laboratory, Tech. Rep., 2009.

[4] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “FPGA-oriented Security,” *Introduction to Hardware Security and Trust*, pp. 195–231, 2011.

[5] M. M. Alam, M. Tehranipoor, and D. Forte, “Recycled FPGA Detection Using Exhaustive LUT Path Delay Characterization and Voltage Scaling,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2897–2910, 2019.

[6] H. Yu, H. Lee, Y. Shin, and Y. Kim, “FPGA reverse engineering in Vivado design suite based on X-ray project,” in *2019 International SoC Design Conference (ISOCC)*, 2019, pp. 239–240.

[7] M. Tehranipoor and F. Koushanfar, “A Survey of Hardware Trojan Taxonomy and Detection,” *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[8] J. A. Roy, F. Koushanfar, and I. L. Markov, “Extended Abstract: Circuit CAD Tools as a Security Threat,” in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 65–66.

[9] Z. Zhang, L. Njilla, C. A. Kamhoua, and Q. Yu, “Thwarting Security Threats From Malicious FPGA Tools With Novel FPGA-Oriented Moving Target Defense,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 665–678, 2019.

[10] V. Mirian and P. Chow, “Extracting Designs of Secure IPs Using FPGA CAD Tools,” in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 293–298.