DPReDO: Dynamic Partial Reconfiguration enabled Design Obfuscation for FPGA Security

Sandeep Sunkavilli, Nishanth Goud Chennagouni, and Qiaoyan Yu
Dept. of Electrical and Computer Engineering
University of New Hampshire

***Abstract*—FPGA security is being challenged by reverse engineering, Trojan, and side-channel analysis attacks. Although the existing static obfuscation methods can protect the integrated circuits from IP piracy and hardware tampering, they are still vulnerable to persistent attacks in FPGAs. In this work, we leverage the advanced function of FPGA CAD tools to develop a Dynamic Partial Reconfiguration enabled Design Obfuscation (DPReDO) method. FPGA emulation results show that, for each obfuscation variant update, the proposed dynamic obfuscation method only consumes 1.5% of the FPGA reconfiguration time required by an existing static obfuscation. Our case study shows that the netlist synthesis time of our method is 26% less than the baseline.**

***Index Terms*—FPGA, hardware security, partial reconfiguration, obfuscation, hardware Trojan.**

I. Introduction

The increasing FPGA market share has motivated more and more hardware hackers to attack FPGA based systems. Various cloud FPGA services further open the door to numerous side-channel attacks through covert communication channels [1]. Even if lacking the explicit mapping of memory addresses to resources and with data encryption, the reverse engineering on FPGA bitstreams is still considered a powerful attack [2]. Hardware Trojan insertion is a significant security threat to FPGA security throughout its life cycle [3]. A hardware Trojan could introduce some malicious chip functionality or leak sensitive data [4]. Malicious intellectual property (IP) could be integrated into FPGA chips during the device manufacturing stage [5]. Other security threats [6], [7] could be originated from fraudulent FPGA Computer-Aided-Design (CAD) tools, which stealthily sabotage the integrity of FPGA bitstream.

Bitstream encryption is one of the most important security measures for FPGA security. Symmetric and asymmetric cryptographic systems have been developed and widely utilized in FPGAs. Symmetric encryption techniques, such as Advanced Encryption Standard (AES) [8], encrypt and decrypt data using a key that is identical to the transmitter and receiver. Asymmetric cryptography techniques such as Elliptic Curve Cryptography (ECC) encrypt and decrypt data using different keys [9]. Logic locking and design obfuscation [10], [11] are another kind of encryption, which blocks the normal logic operations if the authorized key is not available. The widespread machine learning techniques have been applied to address the FPGA security issues [12].

This work was partially supported by NSF Award #1652474.

Despite static defense methods such as encryption and side-channel signal analysis can provide certain attack resilience, they are not sufficient to thwart persistent attacks. As attackers gain more and more knowledge of the static defense mechanism, the protection offered by the static defense method will eventually diminish. Thus, it is imperative to develop countermeasures that can upgrade themselves if necessary. In software, it is easy to patch the existing defense solution. In contrast, hardware is not patchable after fabrication. Fortunately, FPGAs provide users with some flexibility to advance the hardware defense mechanism through the mean of reconfiguration. However, most of the existing defense methods for FPGAs still belong to a static solution, not being designed to support the flow of dynamic modification in future. Toward this need, we propose a dynamic defense method to strengthen the attack resilience of the design mapped to FPGA chips.

The main contributions of this work are as follows:

- We propose a dynamic partial reconfiguration (DPR) based obfuscation method to obscure the hardware design mapped to FPGAs. To the best of our knowledge, this is the first work that leverages the partial reconfiguration function offered by the FPGA CAD tool to strengthen the countermeasure's attack resilience. By using partial reconfiguration, our defense method improves the unpredictability of countermeasures.
- A new FPGA design flow is proposed in this work to enable design partition and dynamic obfuscation.
- The attack resilience achieved by the proposed dynamic defense method is analyzed and validated with using two ISCAS benchmark circuits.

II. Related Work

The defense methods for FPGA security can be categorized in many ways. Depending on whether the method can be upgraded in future or not, we summarize the existing effort into static and dynamic defense groups.

A. Static Defense Methods

The watermarking technique [13] manipulates the state transition graph to generate a rare topological feature presented in a sequence of inputs that traverse specified finite state machine. Hardware IP protection techniques that need to store a permanent key is prone to memory leak attack. To address this security threat, the work [14] introduces a scheme that links

the IP to specific FPGA devices by communicating with physical unclonable functions (PUF) on the FPGA. The method MUTARCH [15] relies on architectural diversity, prohibits invalid upgrades to the mutated FPGA devices, and lowers the economic incentive for attacks. These FPGA security measures only protect the FPGA implementation during the bitstream generation stage. The design after FPGA deployment is still vulnerable to the threats from the compromised FPGA devices. Without relying on a golden version, the work [16] detects anomalies in the FPGA physical layer by examining if the fundamental building block on the FPGA die has different physical statistical parameters in surrounding blocks.

B. Dynamic Defense Methods

Moving target defense (MTD) and partial reconfiguration are the two primary dynamic defense mechanisms. The work [17] executes dynamic defense with Instruction Set Randomization (ISR) and Address Space Randomization (ASR) techniques in microsystems. That method lowers the probability of a successful reverse engineering attack and the risk of being discovered via minimizing the adversary's reconnaissance knowledge. The principle of MTD is exploited by the work [18], [19] to thwart the Trojan insertion attack from malicious FPGA CAD tools. The MTD based defense method creates ambiguity in the FPGA's place and routing stage, thereby reducing the hardware Trojan hit rate and safeguarding the bitstream from harmful modification. However, the methods in [18], [19] do not protect the FPGA design once the bitstream is downloaded to the FPGA. To mitigate the Trojans induced in fabrication or design time, the MORPH architecture [20] combines multiple protection schemes, including morph operation, onion encryption, replication, partial run-time reconfiguration, and hardware abstraction layer, to prevent information from leaking. That work does not perform idea validation. The recent work [21], [22] provides insight into dynamic partial reconfiguration and proposes the challenges and opportunities of FPGA DPR.

III. Preliminaries

In the traditional FPGA design flow, an FPGA user needs to halt the current FPGA operation and then reconfigure the FPGA fabric with a new bitstream. In contrast, the partial reconfiguration (PR) function allows the FPGA user to change the logic inside reconfigurable modules without reprogramming the entire top module or disturbing the active design in the rest of the FPGA chip. The FPGA design flow that supports partial reconfiguration is depicted in Fig. 1. In step 1, the design is first divided into static logic and dynamic logic. The static logic consists of a top module and any other submodules in the design that do not require dynamic programming, such as clock trees. The dynamic logic is the submodules for future partial reconfiguration and comes with variants. In step 2, all design modules are synthesized separately, and the synthesis design checkpoints are generated for all the module variants and static logic. In step 3, the static logic will be linked with one variant of all the dynamic modules. Note, we will need

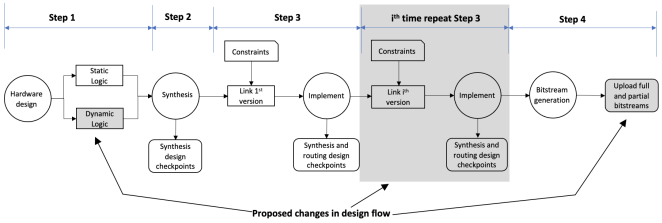


Fig. 1. FPGA design flow that supports partial reconfiguration.

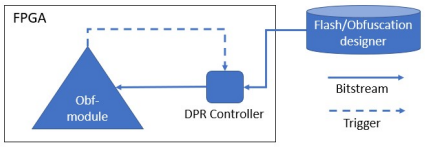


Fig. 2. Control flow of DPR in FPGA.

to create a *Pblock* and then assign each dynamic module to the Pblock in this step. We can repeat this step if multiple variants are available to update the design. At the end of Step 3, post-implementation routing checkpoints are generated for the entire design variant. In step 4, the bitstreams for the full static design and dynamic variants are formed. To support the dynamic obfuscation defense, the blocks highlighted in Fig. 1 will be involved. A Xilinx DPR controller can be customized to deploy bitstreams without designer intervention. As shown in Fig 2, when a trigger signal is sent to the controller, a partial bitstream of a different variant of obfuscation module is read from an external flash memory or designer into the FPGA.

IV. Proposed DPR-enabled Design Obfuscation

A. Method Description

Aiming to improve the attack resilience, we exploit the dynamic partial reconfiguration function (DPR), which is provided by the FPGA CAD tool (e.g., Xilinx Vivado), to develop a DPR-enabled design obfuscation (DPReDO) method. This method inherently has a dynamic feature as the obfuscated module can be updated as a partial bitstream at runtime. The conceptual view of the proposed DPReDO is shown in Fig. 3. Assume the top design has two submodules being obfuscated (i.e., BX1 and BX2) and the entire project is run in a cloud FPGA. The incomplete top design that misses BX1 and BX2 is described in the static bitstream and uploaded to the cloud FPGA first. A series of BX1 (i.e., BX1' and BX1") and BX2 (i.e., BX2' and BX2") are implemented as multiple FPGA bitstream variants, which will be sent to the cloud later for dynamic obfuscation. The complete design for the top module is formed in the cloud and the bitstream for the entire top unit varies with the deployed partial bitstream. From the attacker's point of view, the top design is not only obfuscated but also the obfuscation is not fully predictable in terms of time and function. Thus, the extra uncertainty offered by the proposed DPReDO strengthens the attack resilience of the design obfuscation.

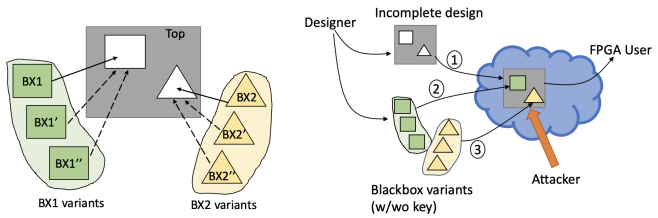


Fig. 3. Proposed dynamic partial reconfiguration enabled design obfuscation: (a) top module with reconfiguration variants (b) design integration.

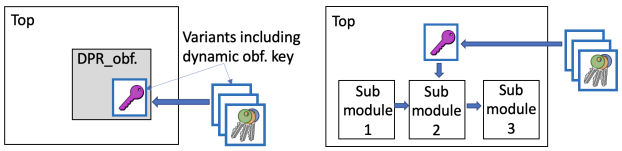


Fig. 4. Two styles of dynamic obfuscation: (a) hierarchical and (b) parallel.

The dynamic obfuscation module can be a unit with two hierarchical layers, one for obfuscation logic and another for obfuscation key management, as shown in Fig. 4(a). With the support of multiple FPGA variants, we have the flexibility to create as many obfuscation key sets as needed. This method eliminates the need of hardcoded key or limited key options in hardware. The obfuscation key can also be provided as a direct replacement via dynamic partial reconfiguration, which is depicted in Fig. 4(b). Regardless which dynamic obfuscation style is applied in the DPReDO, our method improves the unpredictability of the design and achieves better attack resilience than the static obfuscation.

Meanwhile, the DPReDO has potential to reduce the total time that the FPGA design upgrading needs. Figure 5 illustrates the time difference between static and dynamic FPGA reconfiguration. In the static reconfiguration, we will accumulate the time for netlist compilation time, bitstream programming, and operation for functional assessment per each design revision. The total elapsed time is proportional to the number of iterations we have in the FPGA application development cycle. In the dynamic partial reconfiguration, we save the time from multiple stages: (1) do not need to repeat the compilation of static logic of the top design, (2) only compile the variants for dynamic obfuscation, and (3) hide the FPGA mapping time underneath the functional assessment time. Besides offering better attack resilience, our method can also reduce the development time of the entire project.

The pseudo code for our DPReDO method is described in Algorithm 1. The dynamic obfuscation module **DPR_obfuscation** is instantiated in the highest level of the design module **TopDesign**. The inputs for the obfuscation modules are only internal signals from the top module such that no risk of leaking the locking key, including key size and the key input portal. The bitstream for **DPR_obfuscation** is provided as a blackbox after the main design IP is delivered. Furthermore, that bitstream is in a format of partial bitstream and varies with each variant, which changes during the dynamic reconfiguration. The key vector for obfuscation is handled by a **KeyManagement** module (another dynamic FPGA variant module). Thus, the vulnerability of key leaking is significantly reduced.

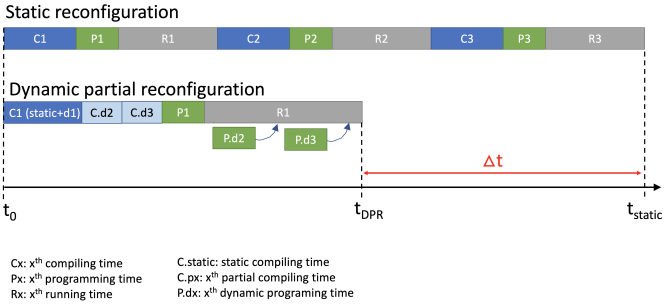


Fig. 5. Conceptual illustration of design reduction achieved by proposed DPReDo.

Algorithm 1: Pseudo Code for Proposed DPReDO.

```
1  module TopDesign (port list);
2     Declaration of input ports;
3     Declaration of output ports;
4     DPR_obfuscation Inst(internal nets);
5     Original logic statements;
6  endmodule
7  module DPR_obfuscation (port list);
8     Declaration of input ports;
9     Declaration of output ports;
10    Declaration of obfuscation key vector;
11    /*Example of an obfuscation logic statement*/
12    if ((in1 & (~ in2)) == Key[0])
13       NextState = CORRECT_NextState;
14    else
15       NextState = WRONG_NextState;
16    Other obfuscation logic statements;
17 module KeyManagement (Key);
18    Hierarchical obfuscation;
19    Parallel obfuscation;
20 endmodule
```

TABLE I
COMPARISON OF FPGA RESOURCE UTILIZATION IN STATIC AND PROPOSED DYNAMIC OBFUSCATION METHODS.

Method	FPGA Utilization			
Baseline [11]	# of LUTs		# of slices	
	1871		537	
Proposed	Static logic (# of LUTs)	Static logic (# of slices)	Dynamic logic (# of LUTs)	Dynamic logic (No. of slices)
	1753	486	27	33

B. A Practical Example

An ISCAS benchmark circuit, s5378, is used in this case study. In the baseline, we obfuscated the circuit with the method in [11]. Next, the same circuit was obfuscated by

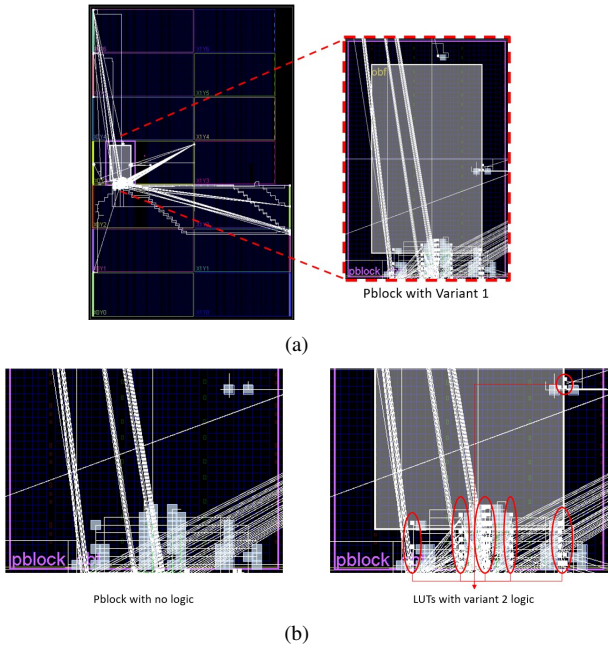


Fig. 6. FPGA floor plan post implementation. (a) s5378 with obfuscation module (b) Obfuscation Pblock with and without logic.

the proposed DPReDO, which generated one static bitstream for the s5378 top module and three variant bitstreams for the obfuscation submodules. Here, parallel style of dynamic obfuscation was used in DRPeDO. In step 3 of the FPGA design flow shown in Fig. 1, we created a pblock with isolation property to host the obfuscated module logic as shown in Fig. 6(a). The static logic plus one variant of the obfuscation submodule forms a full design. As this case study had three variants of the obfuscation submodule, step 3 was repeated three times. The implementation of the obfuscation submodule was removed from the pblock and a new variant logic was added and implemented as shown in Fig. 6(b). The FPGA utilization for two obfuscation methods is shown in Table I. The proposed method consumes comparable LUTs and FPGA slices that the static obfuscation needs.

C. Attack Resilience of Proposed DPReDO

1) Key Leaking Attack: The existing static obfuscation methods (e.g., logic locking and state obfuscation [10], [11], [19]) all assume that a secure key management is available before the deployment of the design obfuscation. Unfortunately, separating key handling from the design obfuscation is not desirable since the encryption key is critical to ensure the success of obfuscation. The proposed DPReDO method integrates the design obfuscation and key management into a unified framework, in which the encryption key can be provided and erased through the process of dynamic partial reconfiguration. Because of the run-time modification feature in DPReDO, the risk of leaking the encryption key is significantly reduced over the static obfuscation and meanwhile the overhead on complicate key management is saved.

2) Reverse Engineering Attack: Although static obfuscation obscures the design, the obfuscated netlist is released entirely. If adversaries conduct persistent reverse engineering attacks, the protection from obfuscation will eventually diminish. Our DPReDO enables the runtime update of multiple (theoretically infinite) variants, thus maximizing the dynamicity and unpredictability of the design under obfuscation. For example, a physical unclonable function (PUF) unit is used in the dynamic obfuscation and the corresponding partial reconfiguration bitstreams are sent to overwrite the FPGA chip with an arbitrary time interval. As we can use different PUFs in the process of obfuscation or leave the PUF variant as a blank place holder, the randomness from the PUF further raises the bar for reverse engineering attack (even if machine learning algorithms are used in the attack).

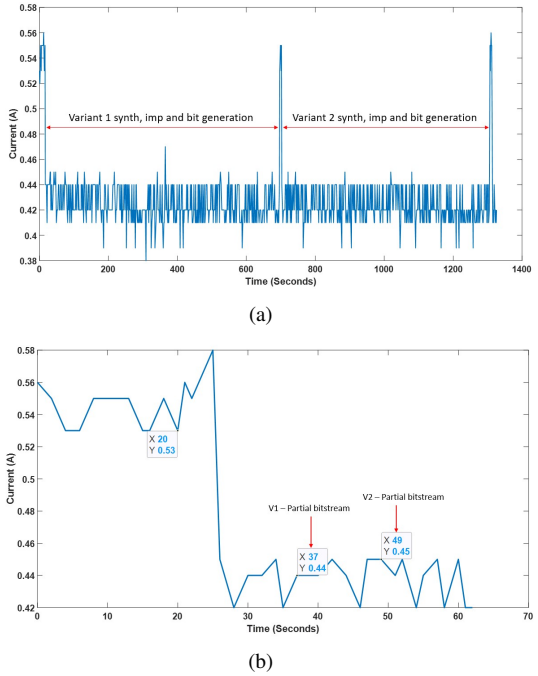
3) Hardware Trojan Attack: Hardware Trojans could sabotage the integrity of the original design, leak information, or consume extra power. The dynamicity introduced by DPReDO increases the resilience against the Trojan attack with the following two reasons: (1) Since the complete design is not fixed, the Trojan trigger condition is more difficult to meet in DPReRO than in static obfuscation. Thus, the deactivated Trojan payload will be benign. (2) If the hardware Trojan could hit the exact FPGA location where the target logic is, that location could move as we replace the critical variants. As a result, the Trojan insertion cannot always succeed and the attack is not persistent anymore.

4) Side-Channel Analysis Attack: FPGAs are prone to side-channel analysis attack. The availability of multiple variants (i.e., partial reconfiguration bitstreams) will break the consistent correlation between the obfuscation submodule and the total power consumption (or critical path delay) of the full design. We envision that DPReDO has a potential to strengthen the obfuscated design from side-channel analysis attack.

V. Experimental Results

A. Experimental Setup

Two testbench circuits ISCAS s298 and s5378 were used in the following assessment. Both the circuits were modified to host an obfuscation submodule, in which a correct key must be provided for the proper function. Different variants of the obfuscation submodules were designed to obfuscate the critical state transition in the benchmark circuits. The small circuit s298 has five obfuscation variants with different locking logic function and key size. The large circuit s5378 is provided with three obfuscation variants. The FPGA board used in this work is Xilinx KC705 evaluation kit hosting Kintex-7 FPGA. The compilation, implementation, and bitstream generation were performed in Vivado 2019.2, which was installed in Intel(R) Core(TM) i7-8565U CPU @1.80GHz 1.99 GHz using x64-based processor. A TI USB-TO-GPIO interface adapter was used to monitor the voltage buses of the FPGA at runtime. The communication between USB interface adapter and PMBus on the FPGA happened using I^2C protocol and communication between the adapter and host computer happened via USB.



(a)

(b)

Fig. 7. Current profile of the FPGA receiving (a) static obfuscation and (b) proposed DPReDO bitstream.

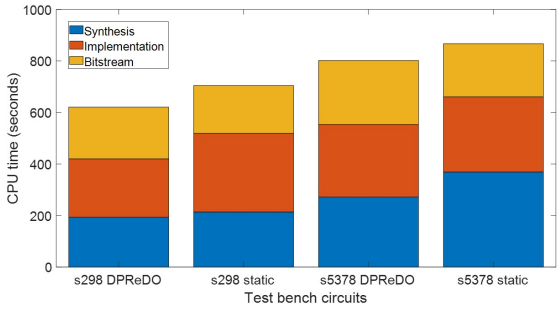
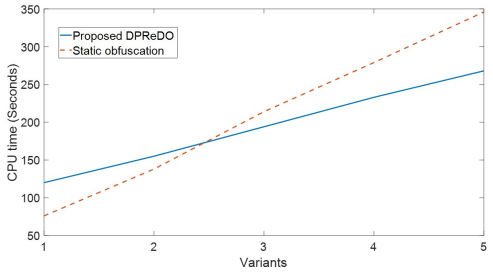
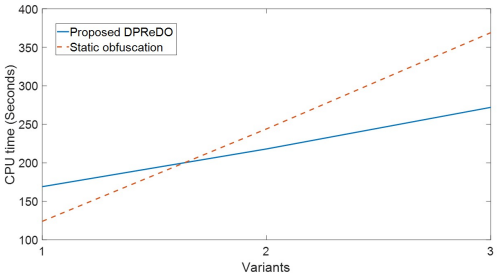


Fig. 8. Time consumed by s298 and s5378.



(a)



(b)

Fig. 9. Synthesis time for dynamic design variants. (a) DPR and static reconfiguration synthesis for s298 (b) DPR and static reconfiguration for s5378.

B. FPGA Obfuscation Speed

The proposed dynamic obfuscation achieves a better FPGA reconfiguration speed than the static obfuscation [11]. In this section, we present the real-time current profile of the FPGA chip and also show the impact of benchmark circuits under obfuscation and the number of variants used in obfuscation on the netlist compilation and FPGA programming time.

1) Demonstration of Current Profile: To be fair, we assume that the static obfuscation was executed twice and two variant bitstreams for partial reconfiguration were programmed to the same FPGA. Figure 7 shows the current flowing through the Kintex-7 FPGA while we were compiling the netlist of s5378 obfuscated statically/dynamically and downloading the corresponding bitstreams to the FPGA chip. Each peak current in Fig. 7(a) indicates the starting point of transferring a new full bitstream. The time interval between two current peaks represents the total time (∼400 seconds) that is required by programming one complete design protected by static obfuscation. As our DPReDO method only updates the obfuscation submodules via partial reconfiguration bitstream, the time interval between two variants is 6 seconds, which is 1.5% of the time used in static obfuscation. If the design revision is taking place in the FPGA cloud, the reduction on the FPGA reconfiguration time will be more significant than the standalone FPGA platform.

2) Impact of Circuit Size on FPGA Obfuscation Time: We followed the same current monitoring approach to further compare the FPGA design time that the static obfuscation and proposed DPReDO methods need when we apply them to s298 and s5378. The total FPGA design time is the sum of the synthesis time, place and routing time, and the bitstream generation time (all reported by Vivado). For each benchmark circuit, three obfuscation cases were tested. As shown in Fig. 8, our method consumes 11.9% and 7.4% less CPU time than the static obfuscation for s298 and s5378, respectively. This experiment result indicates that the reduction on the FPGA obfuscation time increases as the size of the circuit under protection increases with respect to the obfuscation circuit.

3) Impact of Number of Obfuscation Variants on FPGA Synthesis Time: The more variants available for DPReDO will make the obfuscated design achieve stronger attack resilience. In this section, we perform quantitative assessment on the impact of the number of dynamic variants on the FPGA synthesis time.

As shown in Fig. 9, the time for synthesizing one variant in the proposed DPReDO design flow is higher than the time for the static design. This is because the dynamic partial

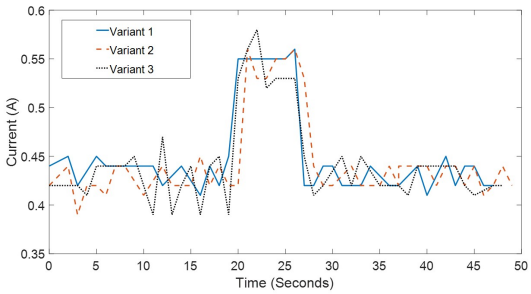


Fig. 10. FPGA current variation.

TABLE II
CURRENT STATISTICS FOR THREE DYNAMIC VARIANTS IN FIG. 10.

Obfuscated variants	#1	#2	#3
Mean	0.45	0.44	0.44
Standard deviation	0.04	0.04	0.04
Correlation coefficient	0.7	0.7	0.8

reconfiguration has overhead time on separately synthesizing static logic, saving checkpoints, Pblock allocation, and partial reconfiguration. Fortunately, The top logic is synthesized only once in DPR whereas in static reconfiguration top logic is synthesized anytime there is a change in the obfuscation variant. This fact helps us to reduce the synthesis time for DPReDO over the static defense. The case study results shown in Fig. 9 indicate that dynamic partial reconfiguration will enable us to reduce the FPGA synthesis time by 2.54% for s298 (with 5 variants) and 26.28% for s5378 (with 3 variants).

C. Variation on Side-channel Signal

We monitored the current through the power line of the FPGA chip and observed that different obfuscation variants yield different current traces. Figure 10 shows the current variation of three obfuscation designs. The statistics of the current profiles for these three designs are listed in Table II. As can be seen, the correlation between any two obfuscation designs is between 0.7 and 0.8, which means that using dynamic obfuscation variants has the potential to increase the difficulty of the side-channel analysis attack.

VI. CONCLUSION

This work introduces a dynamic defense method (DPReDO) to address the persistent hardware attacks on FPGAs. The detailed design flow for dynamic defense in the FPGA platform is proposed, as well. The case studies performed on a Xilinx KC705 FPGA evaluation board prove the feasibility of DPReDO design flow. The FPGA emulation results show that our method not only significantly reduces the FPGA reconfiguration time (as low as 1.5% of static obfuscation), but also saves the netlist compilation time in Vivado and FPGA resource utilization. We further analyze the attack resilience that can be improved by DPReDO. In the future, we will perform more assessments on diverse benchmark circuits.

REFERENCES

[1] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "Fpga side channel attacks without physical access," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 45–52, 2018.

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

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

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

[5] 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.

[6] S. Sunkavilli, Z. Zhang, and Q. Yu, "Analysis of attack surfaces and practical attack examples in open source fpga cad tools," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 504–509, 2021.

[7] S. Sunkavilli, Z. Zhang, and Q. Yu, "New security threats on fpgas: From fpga design tools perspective," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 278–283, 2021.

[8] M. AbuTaha, M. Farajallah, R. Tahboub, and M. Odeh, "Survey paper: cryptography is the science of information security," 2011.

[9] M. Ebrahim, S. Khan, and U. B. Khalid, "Symmetric algorithm survey: A comparative analysis," 2014.

[10] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Hardware-Oriented Security and Trust (HOST)*, pp. 137–143, May 2015.

[11] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 1493–1502, Oct 2009.

[12] X. Xu and J. Zhang, "Rethinking fpga security in the new era of artificial intelligence," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pp. 46–51, 2020.

[13] A. Oliveira, "Techniques for the creation of digital watermarks in sequential circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1101–1117, 2001.

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

[15] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, and S. Bhunia, "Mutarch: Architectural diversity for fpga device and ip security," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 611–616, 2017.

[16] Y. Pino, V. Jyothi, and M. French, "Intra-die process variation aware anomaly detection in fpgas," in *2014 International Test Conference*, pp. 1–6, 2014.

[17] B. Potteiger, Z. Zhang, and X. Koutsoukos, "Integrated moving target defense and control reconfiguration for securing cyber-physical systems," *Microprocessors and Microsystems*, vol. 73, p. 102954, 2020.

[18] Z. Zhang, L. Njilla, C. Kamhoua, K. Kwiat, and Q. Yu, "Securing fpga-based obsolete component replacement for legacy systems," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pp. 401–406, 2018.

[19] 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.

[20] G. Bloom, B. Narahari, R. Simha, A. Namazi, and R. Levy, "Fpga soc architecture and runtime to prevent hardware trojans from leaking secrets," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 48–51, 2015.

[21] W. Lie and W. Feng-yan, "Dynamic partial reconfiguration in fpgas," in *2009 Third International Symposium on Intelligent Information Technology Application*, vol. 2, pp. 445–448, 2009.

[22] S. Sunkavilli and Q. Yu, "Security threats and countermeasure deployment using partial reconfiguration in fpga cad tools," in *2022 IEEE International Workshop on Hardware-Oriented Security and Trust*.