Multi-Tenant FPGA-based Reconfigurable Systems: Attacks and Defenses

Rana Elnaggar†, Ramesh Karri ‡ and Krishnendu Chakrabarty†

†Department of Electrical and Computer Engineering, Duke University
‡Department of Electrical and Computer Engineering, New York University

***Abstract*—Partial reconfiguration of FPGAs improves system performance, increases utilization of hardware resources, and enables run-time update of system capabilities. However, the sharing of FPGA resources among various tenants presents security risks that affect the privacy and reliability of tenant applications running in the FPGA-based system. In this study, we examine the security ramifications of co-tenancy with a focus on address-redirection and task-hiding attacks. We design a countermeasure that protects FPGA-based systems against such attacks and prove that it resists these attacks. We present simulation results and an experimental demonstration using a Xilinx FPGA board to highlight the effectiveness of the countermeasure. The proposed countermeasure incurs negligible cost in terms of the area utilization of FPGAs currently used in the cloud.**

I. Introduction

There is a growing interest in integrating field-programmable gate-arrays (FPGAs) in many applications. In 2015, Intel acquired Altera to integrate FPGAs into Intel platforms. Integration of FPGAs in high-end platforms enables domain-specific customization. For example, they are useful in autonomous driving, IoT and machine-learning applications and acceleration in datacenters [1]. Amazon AWS has introduced FPGA-as-a-service where users can request FPGA resources and deploy their applications on the cloud [2].

FPGAs offer high computational capacity while consuming less power compared to many-core and GPU-based systems [3]. FPGA reconfiguration reduces area overhead; the same FPGA resources are time-multiplexed between various applications. Partial reconfiguration is one of the key features supported by the current generation of FPGAs which permits the reconfiguration of specific regions of an FPGA at run-time. Dynamic partial reconfiguration of a hardware task at run-time allows the addition of new design features or the repair of flaws in the task, e.g., a hardware bug that escapes the verification phase can be updated in-field [4]. In partial reconfiguration, the design is partitioned into static and reconfigurable tasks [5]. Static tasks are fixed during run-time, while the reconfigurable tasks are updated at run-time without interrupting other tasks.

Partial reconfiguration enables the realization of different deep-learning accelerators as requested for by the software application [6]. Software-defined radios use partial reconfiguration to support multiple communication channels on the same FPGA without implementing the processing logic for each channel on dedicated hardware [7].

Moreover, partial reconfiguration of FPGAs promotes co-tenancy where multiple users (tenants) deploy their software applications and hardware tasks on the same FPGA-based programmable logic. Each one of these co-tenant tasks represents intellectual property (IP) of diverse stakeholders. The IPs are however vulnerable to attacks. Consider an FPGA-based system where numerous deep-learning accelerators are employed to support different software applications. A malicious attacker can steal the co-tenant deep-learning IP running in a neighboring reconfigurable region. This attacker can gain unauthorized access to such an IP by launching address-redirection attack [8], i.e., by altering the memory-address mapping to load bitstreams from unauthorized memory locations. Next, the attacker can extract the deep-learning IP without getting direct access to the bitstream [9]. Attackers can redirect the messages between the application and the hardware task to an unauthorized hardware task to steal sensitive data.

Task hiding is another threat. Attackers can sidestep the reconfiguration manager and reconfigure the FPGA with a malicious bitstream [10]. This attack is analogous to process hiding by kernel rootkits in processors [11].

Countermeasures ensure that bitstreams are obtained from authorized sources and are not modified during the transfer to the FPGA [12]. They statically evaluate the hashes of the incoming bitstreams, and compare them to golden values. However, these countermeasures do not protect the system against attacks that modify the expected value, the memory address to fetch the bitstream from, or exploit the configuration access ports (e.g., the internal configuration access ports (ICAP) and the processor configuration access port (PCAP)) [13]. We address these limitations and develop remedies to detect address-redirection and task-hiding at run-time. The major contributions of this study are as follows:

1) It pinpoints the risks of memory-address redirection and task hiding in FPGA-based reconfigurable systems.
2) It secures FPGA-based systems against memory-address redirection and task-hiding attacks. The scheme is backed by theoretical proofs.
3) It highlights the effectiveness of the method through experimental demonstration using a Xilinx FPGA board.

The structure of the paper is as follows. Section II summarizes FPGA-based reconfigurable system architectures introduced in the literature and previous security strategies. Section III defines the threat model. Section IV describes the suggested countermeasure and theoretical analysis. Section V presents simulation results and the FPGA-based demo. We conclude the study in Section VI.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Preliminaries

A. FPGA-based Partially Reconfigurable Systems

Fig. 1 (enclosed in the dotted rectangle) shows the baseline FPGA-based reconfigurable system architecure. This architecture consists of the following components:

- A software layer to co-ordinate the execution of the multiple software applications.
- On-chip processing system (PS) using CPU core(s).
- An on-chip programmable logic (PL) using one or more FPGAs with multiple reconfigurable regions (rrx), where x is the index of the reconfigurable region.
- On-chip/Off-chip memory that stores the bitstreams that belong to different users.
- Configuration access ports, e.g., ICAP and PCAP.
- A reconfiguration manager (RM) implemented either in software or in hardware. It manages run-time reconfiguration of the various tenant applications on the reconfigurable regions. It allocates a hardware task and the related software application to a reconfigurable region.

This FPGA-based system architecture reflects the various reconfigurable architectures proposed in the literature. We assume that there are multiple users that share their computations on the same FPGA-based computing platform (programmable logic). The FPGA-based reconfigurable system operates as follows: (1) The bitstreams of all users are saved in memory. (2) Users submit requests to the RM to use FPGA resources. (3) The RM schedules and allocates the hardware tasks (bitstreams) to the partial reconfigurable regions of the FPGA. (4) The RM loads the target bitstream from memory and controls the configuration access port to reconfigure the FPGA.

The design in [14] proposes an FPGA-based system, which has an interface layer between the operating systems and the hardware tasks implemented in the reconfigurable regions. The status of the reconfigurable regions is monitored by a "hardware controller' implemented on the FPGA. Hardware tasks are scheduled and assigned to the reconfigurable regions by a "hardware task manager", which is a micro-kernel service in software on an isolated virtual machine. The hardware task manager receives feedback about the reconfigurable regions from the hardware controller.

B. Prior Security Methods in FPGA Reconfigurable Systems

A prior security scheme calculates the hash value of the configured bitstream and compares it with an expected value [15]. However, an attacker can modify the expected hash value so that it matches the hash of the malicious bitstream. For example, if the expected hash value is loaded from memory, the attacker can modify the loaded hash value or if the hash value is appended to the bitstream, the attacker can fetch a different bitstream. In another approach a Trust Platform Module (TPM) is implemented in the static region of the FPGA to support on-chip cryptographic operations and to evaluate the hash of the system state. The TPM evaluates the hash of the system state (hardware + software) and sends the system state to the update server before running the system updates [16].

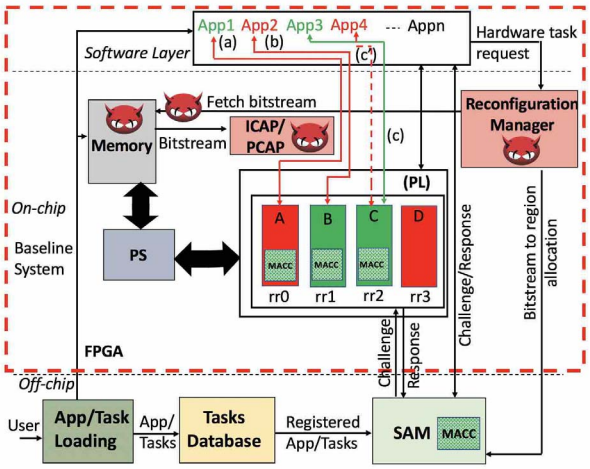

Fig. 1: The baseline FPGA-based reconfigurable system and the threat model. Arrows labelled 1 to 6 are the steps in the defense.

Soft embedded cores implemented on the FPGA can be used as a configuration manager to ensure secure internal FPGA reconfiguration [17]. However, this scheme is not secure when the attacker compromises the FPGA, including the soft cores.

Remote side-channel attacks [18] and remote fault attacks [10] are other emerging threats to multi-tenant FPGA-based reconfigurable systems. These attacks are launched by malicious designs with ring oscillators (ROs) on the FPGA. ROs are used as power sensors to steal secret data processed by other designs running on the same FPGA. ROs can also cause sudden fluctuations in the FPGA power distribution network, which can lead to timing faults.

III. Threat Model

Consider the FPGA-based system shown in Fig. 1. We assume that the FPGA vendor is trusted. However, the FPGA can be exploited by one of the malicious software applications or those that have exploitable vulnerabilities[1]. Malicious software can launch a variety of attacks, including two described below.
Address redirection: In this attack, the attacker does the following: (i) modifies the memory address to load an incorrect bitstream (e.g., in Fig. 1, hardware task A is incorrectly fetched from memory and configured in $rr0$ to illegitimately access data sent by App1); (ii) configures the bitstream in a reconfigurable region other than the one assigned by RM (e.g., in Fig. 1, hardware task B is configured in $rr1$ instead of $rr2$ to grant App2 illegitimate access to hardware task B); (iii) redirects the authenticated communication between the applications and the hardware tasks to malicious entities (e.g., in Fig. 1, when App3 and hardware task C are activated and exchange messages, the attacker redirects the messages exchanged to App4.).

This attack allows malicious software to steal the IP implemented in the reconfigurable regions. An example of this is the model-extraction attack [9]. The malicious software passes input data to the implemented machine learning model and reads the generated output by the model. Next, the attacker uses the input-output pairs to replicate the implemented model.

[1]The vulnerabilities can be exploited by adversaries to launch kernel rootkits.

Malicious hardware tasks can leak sensitive data or modify the output result leading to catastrophic consequences, for example, incorrect classification for deep learning model in autonomous driving leading to life-threatening accidents.

Task Hiding: The attacker exploits the RM or the configuration access ports to load malicious, unregistered bitstreams (e.g., hardware task D in Fig. 1). These bitstreams can short-circuit the programmable logic to overheat it [19].

IV. Countermeasures

Our key idea as shown in Fig. 1 is to integrate a secure authentication module (SAM)—external to the reconfigurable system—to continuously perform the following tasks: (1) check the identity of hardware tasks (e.g. tasks A, B, C, and D) and the software applications that request access to them; (2) ensure that all hardware tasks and software applications are authorized to run in the system (precluding task-hiding attacks); (3) ensure that software applications that request access to hardware tasks are authorized to do so. We propose to add the following modules to the baseline system: (i) secure authentication module (SAM); (ii) Task/App loading module; (ii) Secure task database. The SAM must be implemented off-chip (i.e., external to the FPGA fabric that is used by multiple users) to ensure that it remains secure even if the entire FPGA fabric is compromised.

A message authentication code checker (MACC) is integrated in each software application and each hardware task [20]. MACC operates in parallel and is independent of the main functionality of the software application/hardware task. The MACC integrated in each software application or hardware task uses a message authentication code algorithm (MAC) to compute a unique response to the challenge sent by the SAM. The MAC is used to authenticate the identity of the configured hardware task and the software application that attempts to access this task [21]. The symmetric key used by the MAC in MACC is generated by the SAM and distributed to registered users.

The approach is resilient to attacks even if the RM is compromised and the communication between the SAM and the application, and between the SAM and the partial reconfigurable regions are insecure. The chain of trust is maintained between the SAM and the software and hardware tasks by using encrypted messages. The approach operates as follows:

1) Upon user registration within the system, the SAM generates a symmetric secret key per user and shares it with the user to integrate it within its software application (e.g., feed it as an input to the software application) and within the register transfer level (RTL) of the hardware task, and generate the corresponding bitstream. This secret key is used by MACC to evaluate the response of the challenge that is sent at run-time by the SAM. The identities of the registered users and their corresponding secret keys are recorded in a secure database. The bitstreams and software applications of the registered hardware tasks are stored in on/off-chip memory for subsequent processing.

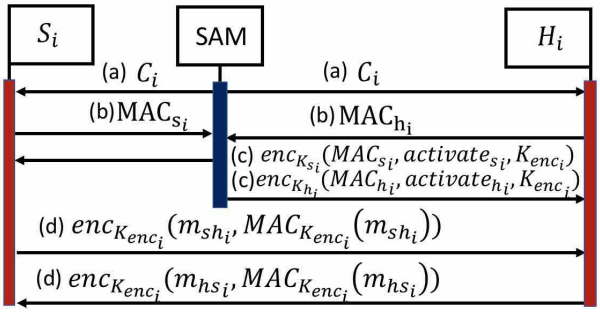

Fig. 2: Communication between SAM, S_i and H_i

2) A software application requests the RM to access a hardware task. The RM allocates the requested hardware task to a partial reconfigurable region and fetches its bitstream from memory.

3) ICAP/PCAP reconfigures the partial reconfigurable region.

4) After configuration, the SAM exchanges message MAC with the configured hardware tasks and software application to verify their identities and confirm the legitimacy of the interactions between them. As shown in Fig. 2(a), it sends a challenge C_i to all the software applications and the reconfigurable regions. In Fig. 2(b) each software S_i responds with MAC_{s_i} equals to $MAC_{K_{s_i}}(C_i)$, where K_{s_i} is the shared key between software S_i and the SAM, and $MAC_{Ks_i}(C_i)$ is the keyed-hash of C_i. Similarly, each hardware task H_i responds with MAC_{h_i} equals $MAC_{K_{h_i}}(C_i)$, where K_{h_i} is the shared key between hardware task H_i and the SAM, and $MAC_{K_{h_i}}(C_i)$ is the keyed-hash of C_i.

5) SAM checks that the software applications and the hardware tasks are authorized to operate in the system and interact with each other. Next, it sends an encrypted message $active_{si}$ to software S_i which is equal to $enc_{K_{s_i}}(MAC_{s_i}, activate_{s_i}, K_{enc_i})$ and $active_{hi}$ to hardware task H_i which is equal to $enc_{K_{h_i}}(MAC_{h_i}, activate_{h_i}, K_{enc_i})$ (see Fig. 2(c)). Note that we can let users integrate their own secret key within their applications and the corresponding hardware tasks. However, key distribution by the SAM ensures that authorized software applications and hardware tasks that run in the system cannot have, by coincidence, a matching secret key with any unauthorized entities.

6) The software application and the hardware task exchange encrypted messages $enc_{K_{enc_i}}(m_{sh_i}, MAC_{K_{enc_i}}(m_{sh_i}))$ and $enc_{K_{enc_i}}(m_{hs_i}, MAC_{K_{enc_i}}(m_{hs_i}))$ using secret key K_{enc_i} (see Fig. 2(d)). The unused partial reconfigurable regions are configured with a dummy hardware task that is verified by the response $MAC_{K_{h_0}}(C_i)$. It is important to note that the SAM can check the identity of the running software applications and hardware continuously at run-time to ensure that only registered hardware tasks and software applications run on the FPGA fabric at any time. This check can be performed as the MACC is designed to operate in parallel with the original working design. In future work, we will investigate how frequently the identity of the hardware tasks and the software applications must be checked.

A. Attack Localization Assuming a Secure RM

The approach localizes the source of the attack as follows if RM is secure: (1) SAM receives an authenticated input from the RM, indicating the allocation of hardware tasks to the partial reconfiguration regions and the identity of software applications that requested access to each hardware task. (2) SAM checks the identity of the hardware tasks in the partial reconfiguration regions and the identity of the software applications as shown in Fig. 2. (3) if the implemented hardware tasks in the partial reconfiguration regions do not match the hardware task allocation indicated by the RM, either the ICAPs or the communication link with the memory are compromised. If a software application requested an unauthorized access to a hardware task, then the application is malicious.

B. Analysis of the Countermeasure

In this section, we prove that the countermeasure ensures the following: (1) authenticates the identity of each running software application and each hardware task implemented in the partial reconfiguration regions. Only registered bitstreams are allowed to run in the system (one cannot hide tasks); (2) bitstream modifications in memory do not affect the overall system security; (3) each software application can only access authorized hardware tasks; (4) the method is resistant to insecure communication links between the SAM and hardware tasks and between the SAM and software; (5) the communication between the software and hardware tasks is private even if the data leaks.

Lemma 1. *The identity of each software S_i in the FPGA-based system is unique, and can be identified by the SAM.*

Proof. Recall that SAM sends a non-repeatable challenge C_i to all active software applications. Suppose that the identity of S_i in our system is not unique. This is true if and only if the response MAC_{s_i} of S_i ($MAC_{K_{s_i}}(C_i)$) is equal to the response MAC_{s_j} of S_j ($MAC_{K_{s_j}}(C_i)$). For this to be true, one of the following conditions must be true: (1) $K_{s_i} = K_{s_j}$, (2) C_i causes key clustering, or (3) an attacker can predict MAC_{s_i}. These conditions cannot be met because: (1) the secure SAM ensures that the generated keys are unique and secret for each software application, therefore, K_{s_i} is not equal to K_{s_j}; (2) the secure SAM evaluates the expected responses of all S_i where $i = \{1, 2.., n\}$ before sending C_i to ensure that C_i does not cause any key clustering; (3) MAC_{s_i} is generated by MACC, therefore according to the basic tenets of message authentication algorithms, even if the attacker has access to previous challenges and the corresponding response of S_i, it cannot predict the response MAC_{s_i} without knowing the secret key K_{s_i} [22]. Therefore, according to the theory underlying MACs, the SAM identifies the unique identity of each application. □

Lemma 2. *The identity of each hardware task H_i that is configured in the partial reconfigurable region PRR_i, where $i = \{1, 2.., n\}$, is unique and can be identified by the SAM.*

Proof. Similar to the proof of Lemma 1, by design, the response MAC_{h_i} of H_i ($MAC_{K_{h_i}}(C_i)$) is not equal to the response MAC_{h_j} of H_j ($MAC_{K_{hj}}(C_i)$) in our proposal. Therefore, the identity of each hardware task is unique. □

Theorem 1. *A software application S_i cannot access hardware task H_i if S_i is not authorized to, even in the presence of a memory address redirection.*

Proof. Suppose that S_i can have unauthorized access to H_i. This is possible if and only if the response MAC_{s_i} of S_i ($MAC_{K_{s_i}}(C_i)$) is in the list of software authorized to access H_i. According to Lemmas 1-2, each hardware task and software application can be uniquely identified and attackers cannot impersonate the identity of authorized applications or hardware tasks. We assume that the SAM is secure and maintains a list of authorized applications that can access each hardware task. Therefore, MAC_{s_i} cannot indicate that S_i belongs to the list of authorized software that can access H_i. □

Theorem 2. *Software application S_i never gains unauthorized access to hardware task H_j even if an attacker intercepts the communication between SAM and S_i, or SAM and H_j.*

Proof. Assume that the SAM sends a non-repeatable challenge C_i, and S_i responds with a unique response MAC_{s_i} where MAC_{s_i} equals ($MAC_{K_{s_i}}(C_i)$). Let the attacker intercept the connection between S_i and SAM and send the response MAC_{s_j} of S_j ($MAC_{K_{s_j}}(C_i)$) instead. The attacker needs SAM to grant access to S_i to access H_j. Thus, SAM identifies MAC_{s_j} to belong to the list of authorized software of H_j and allows S_i to access H_j. SAM grants S_i the permission to operate on H_j by sending the encrypted message $active_{s_j}$ which is equal to ($enc_{K_{s_j}}(MAC_{s_j}, activate_{s_j}, K_{enc_j})$) where MAC_{s_j} is the response of S_j, $activate_{s_j}$ is the activate signal sent by the SAM to enable the software to access H_j, and K_{enc_j} is the encryption key that should be used in S_j/H_j communication. The secret key between SAM and S_j is used to encrypt $active_{s_j}$. S_i has access to H_j, if and only if the decryption of $active_{s_j}$ retrieves MAC_{s_i}, $activate_{s_j}$, and K_{enc_j}. This is not possible because the shared secret key between S_i and the SAM K_{s_i} is not equal to K_{s_j}. Therefore, S_i cannot successfully decrypt the received message. S_i realizes it cannot decrypt the message as it cannot recognize the sent response MAC_{s_i}. Similarly, we can prove that H_j never gains access to S_i even when an attacker intercepts the communication between SAM and H_j, and SAM and S_i. □

Theorem 3. *Software S_i never gains unauthorized access to hardware task H_j even in the presence of an attacker that redirects communication of S_i to another hardware task H_j, during the interaction between S_i and H_i.*

Proof. Suppose an attacker is successful in redirecting messages from S_i to H_j. Assume that the message sent by S_i is $enc_{K_{enc_i}}(m_{sh_i}, MAC_{K_{enc_i}}(m_{sh_i}))$. For the attack to be successful, H_j must successfully (1) decrypt $enc_{K_{enc_i}}(m_{sh_i}, MAC_{K_{enc_i}}(m_{sh_i}))$; (2) evaluate MAC of the message m_{sh_i} and get an output that equals $MAC_{K_{enc_i}}(m_{sh_i})$. For this to be achieved, decryption of $enc_{K_{enc_i}}(m_{sh_i}, MAC_{K_{enc_i}}(m_{sh_i}))$ must be successful. This can be satisfied if and only if K_{enc_j} is equal to K_{enc_i}. This is impossible as the SAM ensures that K_{enc_i} and K_{enc_j} are different for each software and hardware task pair. Therefore,

H_j cannot authenticate the message sent by S_i and blocks the interaction with S_i. □

Theorem 4. *An unauthorized hardware task H_i can never run in the FPGA-based secure reconfigurable system.*

Proof. The proof is by contradiction. Suppose the unauthorized hardware task H_i can run in this system. This is true if and only if the response MAC_{h_i} of H_i to C_i is recognized by SAM. Since H_i is not registered, MAC_{h_i} is not recognized by the SAM. Thus, an unauthorized task cannot run in the system. □

Theorem 5. *An unauthorized software S_i can never run in the FPGA-based secure reconfigurable system.*

Proof. The proof similar to that for Theorem 4. □

V. Experimental Platform and Results

A. Hardware Demo

We evaluated the effectiveness of the proposed countermeasure on Zedboard with a Xilinx Zynq-7000 Programmable SoC. We modified the partial reconfiguration demo in [23] to demonstrate our proposal. In our demo, a C-code source file implements the software application layer and the RM. In this file, the user can request the RM to configure either a Sobel or Gaussian filter in the partial reconfiguration region to process an input image. The software executable runs on the arm core. We considered two different categories of images to represent two different software applications. One input image is captured indoors, while the other is captured outdoors [24]. We added the SipHash [25] to the Sobel and Gaussian filters designs to check their identity using the SAM. We implemented SAM on the same FPGA for simplicity. We specified an access policy as follows: indoor images are processed by Sobel filters while outdoor images are processed by a Gaussian filter. We demonstrated an attack where the attacker attempts to disrupt the specified access policy by modifying the bitstream memory address to configure the Gaussian filter instead of the Sobel filter. The demo shows that the countermeasure stops the software when it attempts to process an indoor image by a Gaussian filter. The demo is available in [26].

Resource Utilization: In order to demonstrate practicality, we evaluated the FPGA resource usage in terms of the number of LUTs and registers utilized. We implemented the SipHash [27] and AES [28], XTEA [29] and Trivium [30] symmetric key crypto cores on the Zynq-7000 SoC. The results are shown in Table I. Cloud FPGA instances have millions of LUTs and registers. For example, the Amazon AWS offers instances with up to eight Xilinx Virtex UltraScale+ (VUP) FPGAs (at the 16 nm technology node) per instance with pricing that starts from $1.65 per hour for one FPGA per instance to $13.65 for 8 FPGAs per instance [2]. Each Virtex Ultrascale+ FPGA has $\approx$ 1.2 M LUTs and 2.4 M registers. In a nutshell, the crypto components used in our countermeasure per user's design consume a negligibly small fraction of the overall instance size (less than 0.3% in the worst case, and as low as 0.0077% in typical cases) as shown in Table I.

Design	# LUTs	# Reg	% VUP LUTs	% VUP Reg
SipHash	907	789	0.076	0.03
AES	3251	2989	0.27	0.12
XTEA	769	162	0.064	0.0068
Trivium	98	184	0.008	0.0077

TABLE I: The number of LUTs and registers utilized by three crypto cores and the SipHash core are reported. These consume negligible amount of resources of a Xilinx Virtex Ultrascale Plus (VUP) FPGA offered by Amazon AWS.

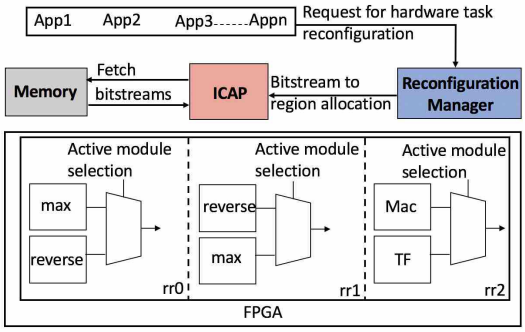

Fig. 3: The architecture of the FPGA-based partial reconfigurable system in ReSim. Max and reverse are implemented in $rr0$; reverse and max are implemented in $rr1$; MAC and TF are implemented in $rr2$. The active task in each region is selected by the active task selection multiplexer.

B. Simulation Environment

In addition to the hardware demo introduced in Section V-A and available online in [26], we demonstrate the behaviour of the proposed countermeasure as well as the concept of attack localization presented in Section IV (A) using a functional verification framework for partial reconfiguration called "ReSim" [31]. ReSim simulates and verifies partial reconfiguration. We use ReSim with three partial reconfigurable regions ($rr0$, $rr1$, $rr2$). To simulate partial reconfiguration, ReSim generates all possible hardware tasks that can be implemented in a certain region[2]. During partial reconfiguration, only one task is enabled using a multiplexer as shown in Fig. 3. In ReSim, $rr0$ can configure two tasks, Max and reverse. Max is initially active in $rr0$ when simulation starts.

Similarly, $rr1$ can configure Max and reverse, and reverse is initially active when simulation starts. In $rr2$, two tasks of the low pass finite impulse response filter (FIR) are implemented; the transfer function (TF) and the multiply and accumulate task (MAC). MAC is initially active when simulation starts.

C. Simulation Results

Address Redirection Attack: We simulate the address redirection attack where MAC is reconfigured in $rr2$ instead of TF. In the specific case when RM is assumed to be secure as described in Section IV (A), RM sends to SAM a list that contains the hardware task requested by each software application and the correct expected allocation of hardware tasks in the partial reconfigurable regions. Fig. 4 shows that the reconfiguration manager Mgr_0 sends $rrid = 2$ and $rmid = 1$, which indicates that TF should be configured in $rr2$. SAM expects that the response of the task implemented

[2]Verilog does not support run-time task instantiation.

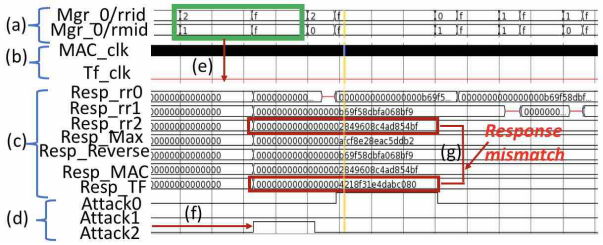

Fig. 4: Simulation snapshot of the FPGA-based reconfigurable system state when address redirection attack affects the hardware task configured to run in $rr2$; (a) signals sent by RM to SAM; (b) clock of tasks MAC and TF implemented in $rr2$; (c) expected and received challenge response in SAM; (d) output signals from SAM indicate the launch of an attack; (e) TF should be configured in $rr2$ but clk of TF is disabled; (f) $attack2$ signal is asserted to represent an attack on $rr2$; (g) received response from the active task in $rr2$ does not match the expected response of TF.

in $rr2$ ($Resp\_rr2$) is equal to the expected response from TF. However, SAM detects a different response. The attack signal ($attack2$) is asserted. This indicates that an attacker might be exploiting the memory, the ICAP, or the communication link between the memory and the ICAP. In the more general case when RM cannot be trusted, SAM identifies the task configured in $rr2$ to belong to a registered user (TF), however, it needs to verify that the software application that attempts to access the hardware task is authorized to access the configured task in $rr2$ as demonstrated in the hardware demo (Section V-A). Moreover, if the communication channels between the software applications, the hardware tasks, and SAM cannot be trusted, SAM sends the activate signal, along with an encryption key to the hardware task and any authorized software application as explained earlier in Section IV.

Task Hiding Attack: In Fig. 5, we demonstrate the task-hiding attack. We modify the key of the hardware task configured in $rr2$ so that the response does not match any of the expected responses by the registered users of the FPGA platform. Therefore, the SAM raises a task-hiding alert and indicates that the attack occurs in $rr2$.

VI. Conclusion

We have highlighted the security risks of co-tenancy in FPGA-based reconfigurable systems. We utilized message authentication codes to identify active tasks and applications running in the system. This protects the system against address redirection and task hiding attacks. Encrypting the communication between the software and the hardware modules prevents the attacker from understanding the data leaked. We have implemented the reconfigurable system with message authentication code added to the software applications and the hardware tasks to show that our proposed countermeasure can be practically deployed on FPGAs. We have investigated the FPGA utilization of the proposed countermeasure to ensure that the added security components consume a negligible fraction of the FPGA resources available for the user's design.

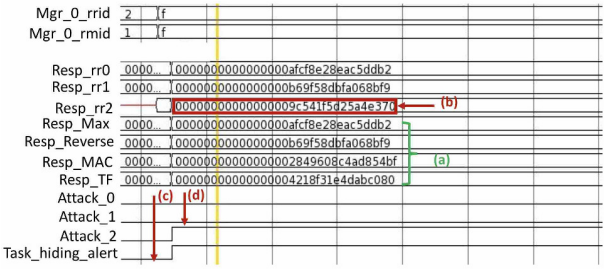

Fig. 5: Simulation snapshot of the FPGA-based reconfigurable system state when task hiding attack is launched in $rr2$. (a) expected responses of registered uses for a given challenge sent by SAM; (b) response of the hardware task configured in $rr2$ does not match the response of any registered user; (c) task hiding alert is asserted; (d) SAM determines that the attack occurs in $rr2$.

References

[1] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[2] "Amazon EC2 F1 Instance," https://amzn.to/2MupiTf.

[3] E. Nurvitadhi *et al.*, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. IEEE FPGA*, 2017, pp. 5–14.

[4] "Dynamic Hardware Patching," https://bit.ly/2gy2a8K.

[5] D. Koch, *Partial Reconfiguration on FPGAs - Architectures, Tools and Applications*. Springer, 2013, vol. 153.

[6] K. Vipin and S. A. Fahmy, "Dyract: A partial reconfiguration enabled accelerator and test platform," in *Proc. IEEE FPL*, 2014, pp. 1–7.

[7] J.-P. Delahaye *et al.*, "Partial reconfiguration of FPGAs for dynamical reconfiguration of a software radio platform," in *Proc. IEEE ISTMWC*, 2007, pp. 1–5.

[8] D. ang *et al.*, "ATRA: address translation redirection attack against hardware-based external monitors," in *Proc. ACM CCS*, 2014.

[9] F. Tramèr *et al.*, "Stealing machine learning models via prediction APIs," in *Proc. IEEE Usenix Security Symposium*, 2016, pp. 601–618.

[10] D. R. Gnad *et al.*, "Voltage drop-based fault attacks on fpgas using valid bitstreams," in *Proc. FPL*, 2017.

[11] R. Hund *et al.*, "Return-oriented rootkits: bypassing kernel code integrity protection mechanisms," in *Proc. IEEE USENIX Security Symposium*, 2009, pp. 383–398.

[12] "Using encryption and authentication to secure an ultrascale/ultrascale+ fpga bitstream," https://tinyurl.com/yaeua2c5, accessed: 2018-07-01.

[13] "Vivado design suite user guide (partial reconfiguration)," https://bit.ly/2MEviZZ, accessed: 2018-06-16.

[14] T. Xia *et al.*, "Mini-NOVA: A lightweight ARM-based virtualization microkernel supporting dynamic partial reconfiguration," in *Proc. IEEE IPDPS*, 2015, pp. 71–80.

[15] R. Chaves *et al.*, "On-the-fly attestation of reconfigurable hardware," in *Proc. FPL*, 2008, pp. 71–76.

[16] T. Feller *et al.*, "TinyTPM: a lightweight module aimed to IP protection and trusted embedded platforms," in *Proc. IEEE HOST*, 2011, pp. 6–11.

[17] A. S. Zeineddini and K. Gaj, "Secure partial reconfiguration of FPGAs," in *Proc. IEEE FPT*, 2005, pp. 155–162.

[18] M. Zhao and G. Suh, "Fpga-based remote power side-channel attacks," in *Proc. SP*, 2018.

[19] C. Beckhoff *et al.*, "Short-circuits on fpgas caused by partial runtime reconfiguration," in *Proc. IEEE FPL*, 2010, pp. 596–601.

[20] M. Bellare *et al.*, "Keying hash functions for message authentication," in *Proc. Springer CRYPTO*, 1996, pp. 1–15.

[21] J. M. Turner, "The keyed-hash message authentication code (hmac)," 2008. [Online]. Available: https://bit.ly/2Oq5Zw8

[22] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[23] "Partial reconfiguration demo," https://bit.ly/2KEIksZ.

[24] B. Wu and R. Nevatia, "Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors," in *Proc. IEEE ICCV*, vol. 1, 2005, pp. 90–97 Vol. 1.

[25] J.-P. Aumasson and D. J. Bernstein, "SipHash: a fast short-input PRF," in *Proc. Springer Int. Conf. on Cryptology in India*, 2012, pp. 489–508.

[26] "Demo files," https://tinyurl.com/yczx5occ, accessed: 2018-07-1.

[27] "Siphash core implementation on the FPGA," https://github.com/secworks/siphash, accessed: 2018-06-24.

[28] "AES core implementation," https://github.com/secworks/aes.

[29] "Xtea implementation," https://bit.ly/2NduK1Y.

[30] "Trivium-cipher implementation," https://bit.ly/2MALPO3.

[31] L. Gong and O. Diessel, "ReSim: A reusable library for RTL simulation of dynamic partial reconfiguration," in *Proc. IEEE FPT*, 2011, pp. 1–8.