HISA: Hardware Isolation-based Secure Architecture for CPU-FPGA Embedded Systems

Mengmei Ye, Xianglong Feng, Sheng Wei
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE, USA
{mye,xfeng,swei}@cse.unl.edu

ABSTRACT

Heterogeneous CPU-FPGA systems have been shown to achieve significant performance gains in domain-specific computing. However, contrary to the huge efforts invested on the performance acceleration, the community has not yet investigated the security consequences due to incorporating FPGA into the traditional CPU-based architecture. In fact, the interplay between CPU and FPGA in such a heterogeneous system may introduce brand new attack surfaces if not well controlled. We propose a hardware isolation-based secure architecture, namely HISA, to mitigate the identified new threats. HISA extends the CPU-based hardware isolation primitive to the heterogeneous FPGA components and achieves security guarantees by enforcing two types of security policies in the isolated secure environment, namely the access control policy and the output verification policy. We evaluate HISA using four reference FPGA IP cores together with a variety of reference security policies targeting representative CPU-FPGA attacks. Our implementation and experiments on real hardware prove that HISA is an effective security complement to the existing CPU-only and FPGA-only secure architectures.

1 INTRODUCTION

Reconfigurable computing architecture, which combines both the general purpose application processor (i.e., CPU) and high performance programmable logic (i.e., field-programmable gate array, or FPGA), have gained rapid advancements recently and demonstrated superior performance and power gains [16][17][19][22]. In particular, the CPU-FPGA architecture has been recently deployed in two types of computing systems: (1) CPU-FPGA based system on chips (SoCs), such as Xilinx Zynq [33], leverage the low power, high performance, and high flexibility brought by FPGAs to speed up the embedded computing tasks; and (2) CPU-FPGA based cluster and cloud systems, such as the recent commercial deployments in Amazon AWS [5] and Microsoft Azure [7], leverage FPGAs to accelerate computation-intensive and domain specific tasks. Without loss of generality, we primarily focus on the CPU-FPGA embedded SoC in this work, given the similarity between the hardware architectures and the security properties of the two systems.

Despite its strong potential in performance acceleration and the commercial deployment, the adoption of the CPU-FPGA architecture still faces significant challenges. One obvious obstacle is the burden of developing hardware oriented computation modules on FPGAs. Even though the problem is partially solved by the emerging high level synthesis tools, the community still heavily relies on third-party FPGA IP cores that are delivered as black box designs. Under this context, the security of the IP cores becomes the top concern due to the following reasons. (1) The hardware IP cores are developed by third parties that are not fully trusted; (2) The service providers typically have limited resources to deploy separate hardware for the sensitive applications; and (3) Despite the traditional belief that hardware is more secure, the reconfigurable nature of the CPU-FPGA system may introduce new vulnerabilities exposed to potential remote attacks on hardware without the need of physical access. For example, the CPU-FPGA system could enable live deployments of hardware Trojans (on the FPGA part) and malware (on the CPU part), which complicate the security measures and elevate the system security requirement to a new level.

The community has mostly focused on the performance acceleration and optimization of CPU-FPGA systems, where security has not been fully studied [16][22]. In the security community, the prior works have intensively studied the security around CPU and FPGA systems, but they have been targeting the CPU and FPGA systems separately, e.g., the former in the software security community [15][18][32] and the latter in the hardware security community [21][23][25][37]. In this paper, we focus on the following unsolved threat models *between* the CPU and FPGA systems, as outlined in Figure 1.

- *CPU to FPGA attack*, in which the software applications on the CPU side attempt to compromise the FPGA system, such as leaking confidential information; and
- *FPGA to CPU attack*, in which the third party IP cores on the FPGA side attempt to compromise the software applications on the CPU side, such as interrupting the software execution or falsifying the data.

Our key insight is to address the root cause of the vulnerabilities that the heterogeneous system components, namely CPU and FPGA, share the same system domain with direct access to each other. Our solution is two-fold. **First**, we adopt a CPU-driven *hardware isolation* mechanism to physically separate the system components, including the processes and data in the CPU domain and the third party IP cores in the FPGA domain, into a secure world and a normal world. The *hardware isolation* is accomplished at the physical

ICCAD '18, November 5–8, 2018, San Diego, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240814

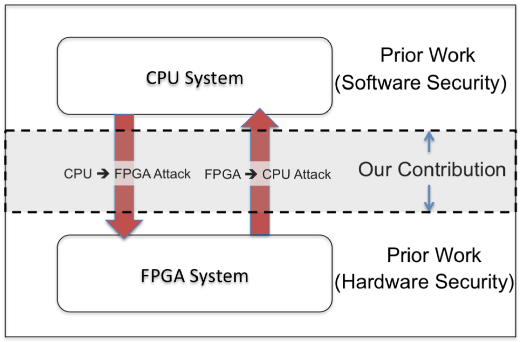


Figure 1: Scope of the work compared to the prior work.

bus level of the system, which ensures that normal world cannot directly access secure world, so that the sensitive components can be deployed in the secure world and become immune to the attack flows from the normal world. **Second**, we deploy a *secure agent* in the secure world, which enforces a set of security policies to block the illegal access from the attack flow (i.e., *access control* policies) and prevent sensitive information from being leaked (i.e., *output verification* policies). The security policies ensure that the secure world resources and services are accessible only by legitimate processes or IP cores, while the attack flows in both directions between CPU and FPGA are blocked. We deploy the hardware isolation-based security mechanism as a new architectural framework, namely HISA.

2 CPU-FPGA THREAT MODELS

2.1 Overview of CPU-FPGA Threats

Given that both the CPU and the third party FPGA IP cores often process critical data, it is not surprising that the attackers have the incentive to break into either system for the purpose of stealing certain confidential data or compromising the security sensitive operations. There are two types of attack flows that are of interest to the security community:

***CPU to FPGA* Attack.** In a *CPU to FPGA* attack, the attacker attempts to access the memory blocks (e.g., BRAMs) on the FPGA side, which store secret data, from a compromised software application on the CPU side. Typically the attacker would issue a two-step threat flow: (1) infer the memory address of the secret data in BRAM using data scan pattern analysis; and (2) access the inferred memory blocks by directly conducting memory access (DMA) or monitoring the data transfer on the bus line.

***FPGA to CPU* Attack.** In an *FPGA to CPU* attack, the attacker first compromises the FPGA design flow by embedding a maliciously modified IP core, namely hardware Trojan [31], into the FPGA. Once triggered, such a hardware Trojan could proactively access the secret data on the CPU side to either leak or falsify the data, both of which cause security consequences concerning secret data or security sensitive operations. In order to hide the Trojan from being detected, attackers typically employ extremely rare triggering conditions that are only known by themselves and difficult to be covered by regular functional test cases.

2.2 Case Study

To better illustrate the two CPU-FPGA threat models, we implement a prototype system of a security sensitive surveillance camera, which has an on-device motion detection module to detect the objects in motion in the captured video in real time, as demonstrated in Figure 2 (video sample courtesy of [10]). Figure 2(a) shows the hardware system setup using Xilinx Zynq-7000 ZC702 SoC. The SoC has a CPU component that contains two ARM cores and an FPGA component that contains a Xilinx FPGA board. We employ the CPU part on the board to provide basic interface to receive the video frames from the HDMI card, and we deploy a motion detection IP core based on the Gaussian Mixture Model (GMM) [9][28] in the FPGA part to conduct real time motion detection based on the received video frames. As shown in Figures 2(a) and 2(b), there is a remarkable white area to indicate the moving objects on the road.

We further implement a threat model based on the prototype system. The outcome caused by the threat model is shown in Figure 2(c), where the moving object is hidden in the background if there is no motion detected. The attack scenario is a “replay attack" triggered by either of the two CPU-FPGA attack flows shown in Figure 3: (1) *CPU to FPGA* attack, where a malware in the CPU system accesses BRAM via DMA, scans the memory blocks for the output video frame location, and replaces the target frame with one of the pre-recorded motion-free frames; and (2) *FPGA to CPU* attack, where a hardware Trojan embedded in the motion detection IP maliciously replaces the video output frames. Both of the attacks compromise the shared data holder (i.e., the BRAM) between the CPU and FPGA, which poses significant security challenges. Existing countermeasures based on encrypting the data in the memory [3][32] or isolating different software applications, information flow, or IP cores [15][18][23][25] do not suffice to prevent such attacks, as the former poses overwhelming overhead for this real-time video processing application, and the latter does not prevent attacks across CPU and FPGA boundaries.

3 HISA FRAMEWORK

The CPU-FPGA hardware isolation primitive is inspired by the most natural way of protecting valuable assets, which is to physically isolate them from the potential malicious attacks. As shown in Figure 4(a), with hardware isolation we split the runtime environment of a CPU core into two isolated “worlds", namely the secure world and the normal world. Under the context of the CPU-FPGA heterogeneous system, each world contains both the CPU component and the FPGA component. The hardware isolation primitive ensures that, while isolated from the secure world, the normal world components do not have access to the secure world resources. However, the secure world applications can still access normal world resources, leading to a secure one-way communication between the two isolated environments. Figure 4(a) demonstrates the four communication flows between the CPU and FPGA components in the secure and normal worlds, where only the two communications originated from the secure world will pass through, and those from the normal world will be blocked by the hardware isolation primitive.

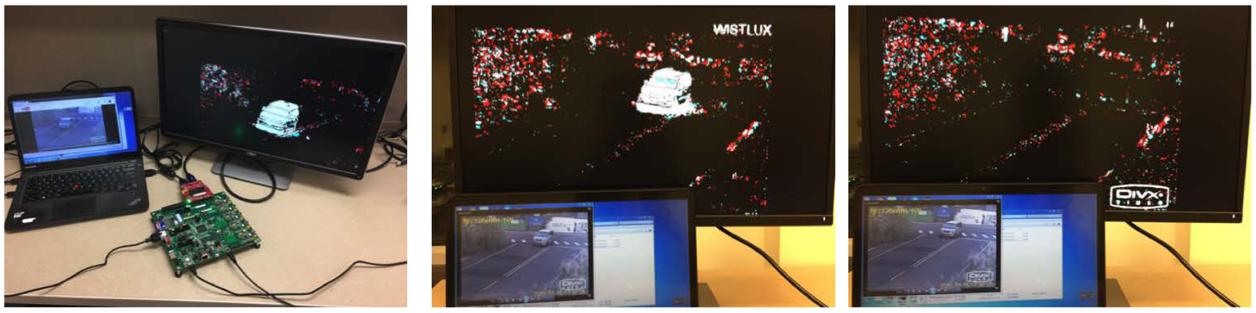

(a) Motion Detection Hardware System Setup (b) Normal Scenario (c) Attack Scenario

Figure 2: Case study of CPU-FPGA threat models in a smart camera IoT system conducting motion detection in real time surveillance video (video sample courtesy of [10]): (a) Hardware system setup using a Xilinx SoC with both CPU and FPGA; (b) Normal motion detection scenario without attacks; and (c) Motion detection results under *CPU to FPGA* or *FPGA to CPU* attacks, in which the moving objects are hidden in the background instead of being detected.

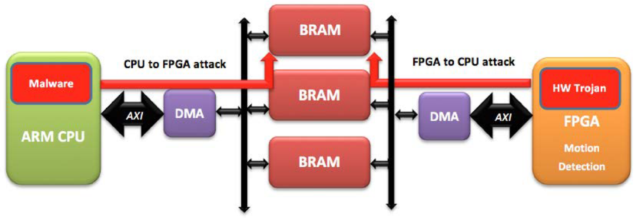


Figure 3: CPU-FPGA threat model design for the motion detection system.

With the recent advancement in CPU security features, such as ARM TrustZone [1], the hardware isolation can be realized at the physical bus level and enforced via strictly controlled CPU context switch, which is secure against software-level attacks [15]. However, the prior research and practice on CPU-enabled hardware isolation technologies involve only the protection of software data/code [18] or OS kernels [15], while the isolation of hardware (e.g., FPGA) components in a CPU-FPGA heterogeneous system has not been fully studied.

Figure 4(b) illustrates our realization of extending the CPU-based hardware isolation to the FPGA system using ARM TrustZone [1]. The ARM-based CPU-FPGA system, such as that in Xilinx Zynq SoC [33], involves a CPU system and an FPGA system connected via an AXI Interconnect. We enable the security checking feature at the AMBA bus port on the AXI interconnect using the Xilinx Vivado tool, which ensures that the AMBA bus port will enforce the security property set by its non-secure (NS) bit while granting accesses. Since the NS bit of the AMBA bus port (i.e., AWPROT[1] for writing and ARPROT[1] for reading) is set to 0 by default (i.e., the property is "secure"), it will only grant accesses to the secure world applications (with NS bit being 0), which technically achieves the secure physical isolation.

3.1 Secure Data Flow

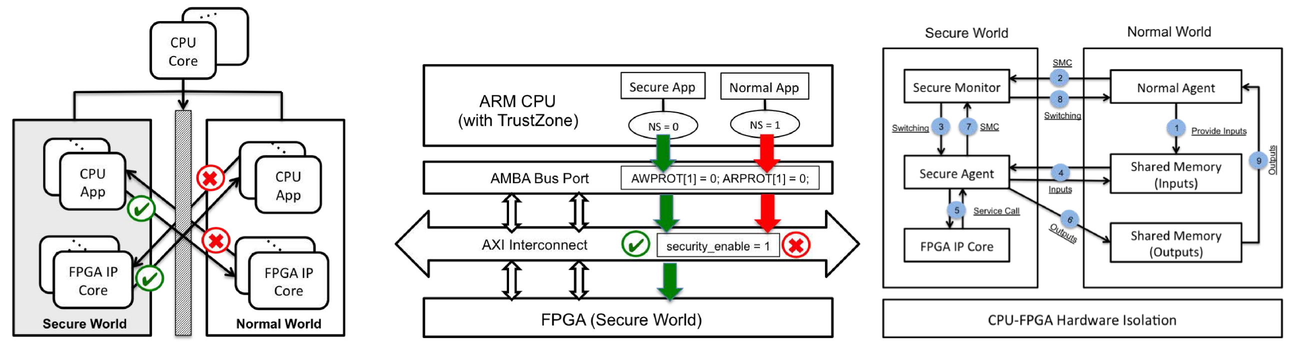
Based on the CPU-FPGA hardware isolation primitive, we deploy the FPGA IP cores into the secure world, as shown in Figure 4(c), so that it is isolated from the potentially malicious normal world applications.[1] The separation of the two worlds causes the system to execute in two independent modes with completely isolated logical and physical components and resources. At runtime, the CPU determines the schedule of switching between the two execution modes, and it employs a *secure monitor*, running in the secure world, to execute the switching upon receiving a secure monitor call (SMC) from either world. To enable the communication between the two worlds, we deploy two blocks of *shared memory* in the normal world for storing the inputs and outputs to and from the secure world. This is required as the system must support the regular functionality of computation using the IP core under protection, as long as the service request is legitimate and the output does not leak confidential information. To support the security verification, we deploy a *secure agent* in the secure world, which is in charge of invoking the IP core services on behalf of the normal world application.

With the completion of the data flow as shown in Figure 4(c), the normal agent successfully invokes the IP service and obtains the response. However, during this process, both the IP core and the data it processes are never exposed to potential malicious access from the normal world, and all the interactions related to the protected hardware and data are conducted by the secure agent that is trustworthy. In addition, the secure agent serves as a security gateway to examine and filter the data requests from the normal world applications, to further improve the security of the IP core.

3.1.1 Secure Monitor. The secure monitor plays an essential role in the entire secure data flow, as it controls the context switching between secure and normal worlds, where completely different and isolated memory blocks and registers are used. During each world switch, the monitor detects the source and destination worlds, saves the states of the source world, and restores the states of the destination world.

3.1.2 Secure Agent. The secure agent resides in the secure world, and it is the only entity that has direct access to the IP core under

[1]For the ease of discussion, here we describe the communication flow using the *CPU to FPGA* attack as an example (i.e., FPGA IP under protection). The same framework and flow can be applied to the *FPGA to CPU* attack as well.



(a) Architecture of CPU-FPGA Hardware Isolation (b) Realization of CPU-FPGA Hardware Isolation on ARM (c) Secure/Normal Worlds Communication Flow

Figure 4: Overall framework and data flow of the HISA framework.

protection, which is activated immediately after the secure monitor completes the world switching operation, as shown in Figure 4(c). The secure agent is in charge of the following three tasks: (1) receiving the original IP service request with inputs and passing it over to the target IP core; (2) delivering the output of the IP service call to the shared memory in the normal world; and (3) issuing SMC to request the switching from the secure world to the normal world.

3.1.3 Normal Agent. The normal agent serves as the access point of the hardware isolation framework to the end users. It coordinates the service call and data communications between the normal world and the secure world. Due to the fact that the normal agent resides in the normal world, it does not have direct access to the secure world data/services. Instead, the normal agent invokes the service call indirectly via an SMC to the secure monitor that in turn switches the CPU mode to the secure world.

3.1.4 Shared Memory. The shared memory is the medium that enables the secure communications between the normal world and the secure world. The hardware isolation ensures that there is no direct communication channel between the two worlds. However, since both the secure world and normal world have access to the normal world resources, it is possible to convert a memory block in the normal world as the shared memory and thus the communication channel.

3.2 CPU-FPGA Security Policies

The CPU-FPGA hardware isolation framework only provides the fundamental support for creating the isolated environment without the enforcement of security. Obviously, it is not practical to disable all accesses to the secure world, as the IP core must allow normal accesses issued by non-malicious users and applications. Therefore, we deploy *access control* policies at the secure agent to check the legitimacy of the service request, for both the *CPU to FPGA* and *FPGA to CPU* flows, to block only the malicious or suspicious requests. Furthermore, for attacks that leak or falsify security/privacy sensitive data or operations and that are not possible to identify at the incoming access control phase, we conduct *output verification* at the secure agent prior to delivering the results from the secure world to the normal world.

3.2.1 Access Control Policy. Upon receiving a request from the normal world, the secure agent first examines whether the requesting agent has the privilege for the service request using a pre-defined, application-specific access control policy. The access control serves two purposes: (1) It eliminates unauthorized service requests to the IP core, possibly issued by a malicious attacker attempting to gain access to the secret data/service hosted by the IP core; (2) For legitimate service accesses, it keeps track of the frequency of such requests and schedules them according to the resource and computation capability of the secure service, which not only ensures efficient resource usage but also prevents potential denial of service attacks. In summary, the access control policy ensures the security and integrity of the protected IP core at the entry point of the secure world.

3.2.2 Output Verification Policy. Upon completing the service request, the secure agent examines the correctness of the results before delivering them to the shared memory. There are two reasons why we enforce this policy. First, the hardware isolation technology only ensures isolation but lacks security and integrity verification for the data and service deployed in the secure world, which may cause security concerns under the events that malicious software and hardware services have been deployed in the secure world, such as a malware or hardware Trojan. Second, for data and services that are subject to reliability issues due to random or environmental effects, the additional verification ensures the correctness of the results.

4 CASE STUDY: SECURING IOT SMART CAMERA

In this section, we close the loop in the case study presented in Section 2 by leveraging HISA to counter the replay attack in the IoT security camera targeting the *FPGA to CPU* attack flow. To employ HISA, we first place the motion detection IP and the corresponding frame buffers into the secure world of HISA. Then, our focus is on the design of application-specific output verification policies for the secure agent to enforce. In particular, we develop both passive and proactive output verification policies to prevent the replay attack. The former monitors representative signals from the system in a non-intrusive manner and captures suspicious behavior; The latter employs a "design for trust" paradigm to facilitate the security verification.

4.1 Passive Output Verification

Given the difficulty in detecting the *FPGA to CPU* hardware Trojan via normal tests, our solution is to adopt a timing side channel-based approach to differentiate the attack and normal scenarios. Our intuition is that there is a clear difference in execution time between the normal motion detection operation and the one infected with hardware Trojans. The normal motion detection module involves loading video frames from the memory, executing the motion detection algorithm, and delivering the results to the output video buffer. Suppose Tc_i represents the time duration of copying the i^{th} frame, and Tp_i represents the time duration of processing the i^{th} frame, the normal motion detection time Tn_i for the i^{th} frame can be calculated as Equation (1).

$$Tn_i = Tc_i + Tp_i \tag{1}$$

In the *FPGA to CPU* attack, once the output frame without motion has been recorded, the hardware Trojan just needs to copy the recorded frame to the output frame buffer and, therefore, the total time duration under attack, Ta_i, can be represented in Equation (2).

$$Ta_i = Tc_i \tag{2}$$

We assume that Tp_i is significantly high as it is consumed by sophisticated computer vision algorithms for motion detection [28] and, therefore, the *FPGA to CPU* attack can be captured by comparing the Tn_i and Ta_i values. In our system design, we place a timer in the surveillance video system to measure the system execution time for the normal case (i.e., motion detection) and the attack case (i.e., frame copy) as shown in Figure 5. We start/stop the timer once the motion detection module begins/ends with the processing of the corresponding video frames. By monitoring the measured timing values, we are able to observe the difference in time when the hardware Trojan is activated.

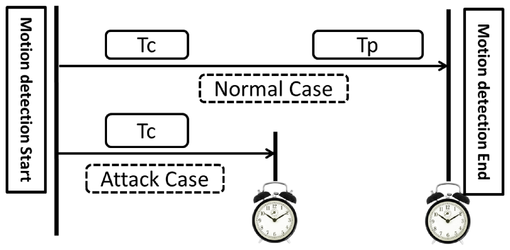


Figure 5: Passive verification via side channel analysis.

4.2 Proactive Output Verification

In proactive output verification, we leverage the fact that the replay attack outputs the same pre-recorded video frames for all the different input frames. As shown in Figure 6, our proactive countermeasure is to insert artificial motion patterns in a sampled set of input frames and check the corresponding output frames periodically for the inserted motion. In our video pipeline, the secure agent generates motion patterns and adds them into the input frames every k seconds. The inserted patterns will be captured by the motion detection module and labeled with white spots in the output frame, which can be as small as invisible to human. Then, the output verification in HISA can check the pixel values in the specific motion-inserted region to determine if they have been labeled in white. Under a replay attack, the output frame will not contain any motion including the intentionally inserted ones, which serves as an indicator for the presence of *FPGA to CPU* attack.

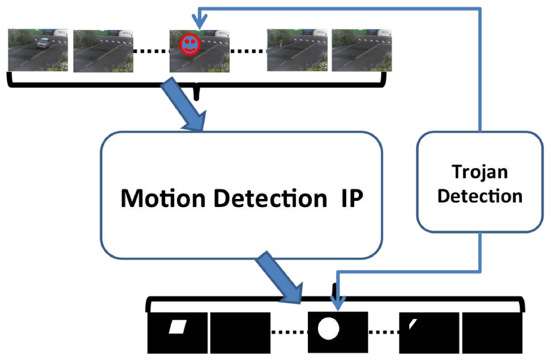


Figure 6: Proactive verification via motion frame insertion.

4.3 Effectiveness of Security Policies

Figure 7 shows the distribution of timing results with time measurements of 1245 frames in the attack case and 1000 frames in the normal case. We observe a large gap between the timing in the two cases (a more than 100% difference). The results indicate that the side channel-based output verification can achieve zero false positives/negatives in detecting the replay attack. Furthermore, we demonstrate the effectiveness of the proactive verification in Figure 8, which shows that, in the non-attack scenario, the output frame contains additional motion objects that are inserted by the defense mechanism. Note that we intentionally design large-size motion objects only for the demonstration purpose, which can be made as small as invisible in a real setting.

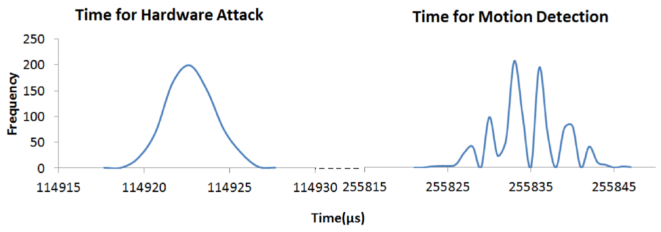


Figure 7: Demonstration of timing side channel analysis.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

5.1.1 HISA Implementation. We implement HISA on a Xilinx Zynq-7000 all programmable SoC [33], namely the ZC702 board. It is

Table 1: Description of reference IP cores, threat models, and security policy designs for security analysis.

Custom IP	Description	Threat Model	Access Control	Output Verification
Multiplier	16-bit multiplier following [4]	Denial of Service* Information Leakage†	Caller Identification Caller Access Control	Sampled Redundancy Check
RSA	32-bit RSA IP following [8]	Denial of Service* Information Leakage†	Caller Identification Caller Access Control	Information Flow
XADC	XADC sensor IP following [14]	Denial of Service* Information Leakage†	Caller Identification Caller Access Control	N/A
Motion Detection	Motion detection for surveillance video following [9]	Data Falsifying Attack*† Information Leakage*†	Secure Isolation	Side Channel Inspection

* *CPU to FPGA* attack; † *FPGA to CPU* attack.

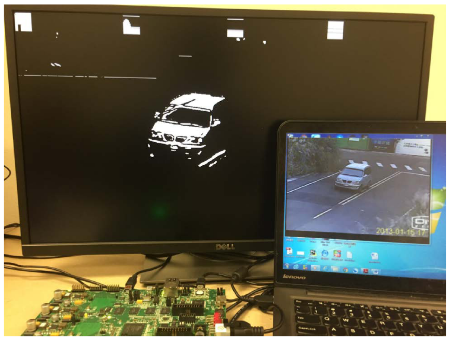


Figure 8: Demonstration of proactive output verification for the motion detection IP (video sample courtesy of [10]).

equipped with an ARMv7-A CPU in the Zynq processing system (PS) and a programmable logic (PL) subsystem (i.e., FPGA), which are connected by the AMBA Interconnect. The ARMv7-A processor has dual 667MHz ARM Cortex-A9 cores, which supports the ARM TrustZone feature for hardware isolation. Since HISA is a CPU-based security framework, it does not introduce additional hardware area and resource overhead to the SoC. Also, the security of the secure agent and the secure monitor is ensured by the secure boot process supported by ARM TrustZone [1].

5.1.2 Reference IP Implementation. For a comprehensive security and performance evaluation, we adopt four different custom IP cores, as shown in Table 1, which cover a variety of application domains, such as computing (i.e., Multiplier), cryptography (i.e., RSA), signal processing (i.e., XADC), and video processing (i.e., Motion Detection). For each IP core, we develop a set of threat models that involves both *CPU to FPGA* and *FPGA to CPU* attacks. Among them, Denial of Service (DoS) is a representative *CPU to FPGA* attack, in which a malicious CPU application attempts to issue an excessive amount of requests to the IP core to block its service to the legitimate users. The information leakage attack is a representative *FPGA to CPU* attack, in which an information leakage hardware Trojan [21] is embedded in the IP core that may leak sensitive data via a covert channel without being identified by the software applications. In addition, we consider several special threat models targeting on certain IP cores, such as replay attack for the motion detection IP discussed in Section 2.

5.1.3 Reference Security Policy Implementation. For our evaluation purpose, we implement two access control policies:

- *Caller Identification*, where the secure agent examines the identity of the caller that is requesting the IP core service and filters out unauthorized accesses via either a "blacklist" (i.e., blocks pre-recorded malicious applications) or a "whitelist" (i.e., only allows for the pre-recorded trusted applications).
- *Caller Access Control*, where the secure agent records the caller access history (e.g., number of access times and access patterns) and blocks the access upon capturing suspicious patterns (e.g., excessive number of accesses) [35].

Furthermore, we implement the following output verification policies:

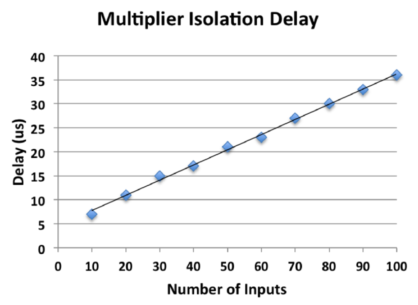
- *Sampled Redundancy Check*, in which we partially repeat the computation by the IP core at the software level. For example, for a multiplier IP, we sample and re-compute certain bits of the multiplication and check for correctness.
- *Information Flow Verification*, in which the hardware isolation framework tracks the information flow by inserting certain check points in the IP core. Then, it checks the correctness of the information flow instead of that of the end results to detect information leakage.

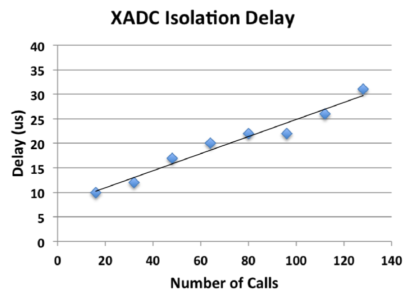
Then, we deploy the access control and/or the output verification policies for each custom IP. For example, the XADC IP adopts the caller identification and access control policies. The multiplier IP adopts sampled redundancy check for the last digit of the multiplication results. The RSA IP employs information flow verification to check whether it has been indeed invoked to differentiate the normal request from the malicious information leakage attack.

5.2 Performance Overhead Evaluation

We first measure the world switching delay for three of the IP cores, as shown in Figure 9. For each IP, we collect a group of delay values while increasing the number of times to invoke the IP service. In these experiments, HISA conducts one world switch from the normal world to the secure world, executes the computation tasks, and switches back to the normal with output to the shared memory. We observe that the hardware isolation delays per 10 IP service calls are below 10 μs, which indicates a very low overhead.

Furthermore, to evaluate the timing overhead introduced by the output verification policies in the motion detection IP, we measure





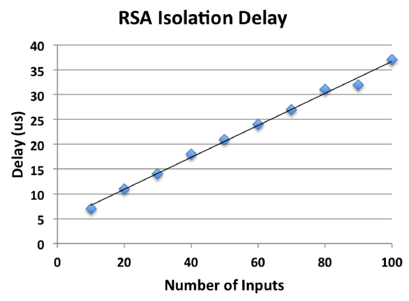


Figure 9: Evaluation of HISA isolation delay on the reference IP cores with access control and output verification.

the execution time of the passive verification for each frame with a total number of 871 frames. The distribution of timing results is shown in the first column of Table 2. We observe that the side channel analysis takes sub-micro second, which is negligible compared to the motion detection time that is hundreds of milliseconds. Also, Table 2 shows the timing overhead of the proactive verification method, which is at hundreds of milliseconds, similar to motion detection time for each frame. Considering that this process will be carried out every k seconds (e.g., k=2), the overhead is in an acceptable range to meet the real time requirement.

Table 2: Timing evaluation of output verifications.

	Passive Verification (μs)	Proactive Verification (μs)
Minimum	0.20	207,785
First Quartile	0.396	207,798
Median	0.411	207,802
Third Quartile	0.438	207,806
Maximum	0.816	207,822

5.3 Power and Energy Overhead Evaluation

We further evaluate the power/energy consumption of HISA considering that power/energy is a limited resource in embedded systems. We run the TI fusion power measuring tool [11][27] while invoking the reference IP cores for power measurements on both the PS (CPU) and PL (FPGA) components on the SoC. Figure 10 shows the detailed power/energy evaluation results of the XADC IP. Similar to the performance evaluation reported in [34], we compare three cases: (1) Regular XADC, which is the baseline IP core without any security protection; (2) Isolated XADC, which is the HISA protected XADC IP; and (3) Encrypted XADC, which protects the IP by encrypting the sensor data using AES-CBC.

In the power results for both PS and PL (i.e., the left two figures), we observe that HISA reaches the peak power with a higher frequency than the baseline (i.e., regular XADC). However, the increased frequency is noticeably lower than the encrypted XADC. Also, for the same workload assigned, HISA finishes the execution earlier than the encrypted XADC, which further saves the energy consumption. These observations are reflected into the energy results (i.e., the right two figures).

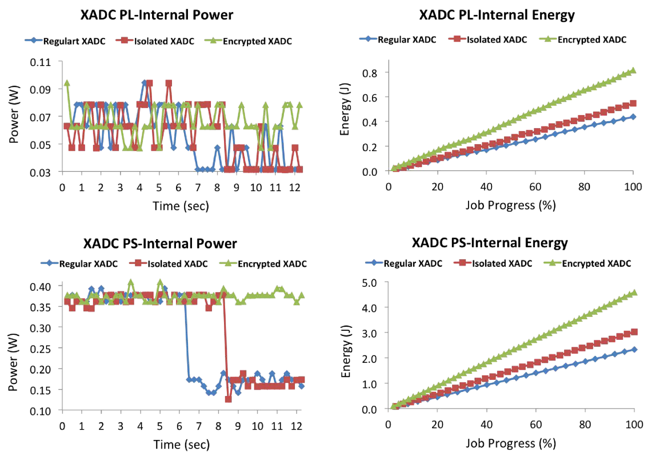


Figure 10: Power and energy evaluation for the XADC IP.

6 RELATED WORK

Intra-CPU and Intra-FPGA Security. The software security community focuses on protecting sensitive data and services from potential attacks within the CPU system, using a variety of security mechanisms, such as encryption-based [3][32] and isolation-based approaches [15][18]. The hardware security community effectively addressed the trust and integrity issues within the FPGA system, such as hardware Trojan detection [31], information flow tracking/isolation [25][37], and isolation between multiple IP cores [23]. However, neither threads of research have considered the interplay between CPU and FPGA and thus the new security challenges brought by the emerging heterogeneous system.

Inter-CPU/FPGA Security. More recently, there have been several research efforts related to the Inter-CPU/FPGA security issues. Jacob et al. employed a malicious IP core to break the secure boot process of a CPU-FPGA SoC [24]. Olson et al. developed "Border Control" [26], an OS-level solution to ensure that the accelerators are respecting the access permissions enforced by the page table. Similarly, MMU or IO-MMU based techniques [13] and the CAPI interface [29] require a trusted OS to manage and enforce the security. HISA is orthogonal to these related works in that (1) it targets the security of the CPU-FPGA systems at runtime instead of the boot-up stage; and (2) it provides a hardware-level solution other than at the OS level, which does not require the OS or upper-level software to be in the trusted computing base (TCB) and thus provides an additional and strong layer of security to the CPU-FPGA systems.

Hardware Isolation Primitives. Isolation or sandboxing has been an important security principle adopted in many security sensitive systems, such as hypervisor-based virtual machines [12] for secure cloud computing. Recently, hardware-based isolation mechanisms, such as Intel SGX [20] and ARM TrustZone [1], have been developed to provide lower level, more resilient isolation primitives. Intel SGX is based on memory encryption and does not support peripheral devices like FPGAs. ARM TrustZone functions at the physical bus level, which could be adopted to protect on-chip peripheral devices. Recently, there have been related works that employ ARM TrustZone to secure software applications [15][18][30][36] and hardware components [21][34][35]. The existing ARM Trusted Firmware [2] implements the generic functionality of TrustZone without customized security policies. HISA differentiates from these existing research efforts by providing a universal CPU-FPGA security framework with well-defined security policies, which support a wide range of applications.

7 CONCLUSION

We have developed HISA, a hardware isolation-based secure architectural extension to mitigate the new threats in emerging CPU-FPGA heterogeneous systems. HISA provides an isolated secure execution environment for both the CPU and FPGA cores in the system, which enforces access control and output verification policies to ensure the security and integrity of the heterogeneous system. We have implemented HISA on a Xilinx Zynq-7000 SoC and evaluated its security and performance using four reference IP cores. Our experiments demonstrate that HISA is able to fill the security gap in the CPU-FPGA systems and introduce minimum performance and power/energy overhead. For the readers' reference, we released the source code of HISA, the reference IPs, and the code and demo of the case study via a Github repository [6].

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Award CNS-1750867 and in part by the gift donation from Adobe Research. Also, we would like to thank Nianhang Hu who contributed to the implementation of the TrustZone framework.

REFERENCES

[1] ARM Security Technology: Building a Secure System Using TrustZone Technology. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html.
[2] ARM Trusted Firmware. https://github.com/ARM-software/arm-trusted-firmware.
[3] BitLocker Drive Encryption. https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/bitlocker-drive-encryption.
[4] Custom IP Project for the MicroZed. https://github.com/fpgadeveloper/microzed-custom-ip.
[5] Developer Preview EC2 Instances (F1) with Programmable Hardware. https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/.
[6] HISA Github Repository. https://github.com/hwsel/hisa.
[7] Microsoft Goes All in for FPGAs to Build out AI Cloud. https://www.top500.org/news/microsoft-goes-all-in-for-fpgas-to-build-out-cloud-based-ai/.
[8] OpenCores. http://opencores.org/project,basicrsa.
[9] Real-time Motion Detection in Video on Zynq FPGA. http://www.jsykora.info/2013/11/real-time-motion-detection-in-video-on-zynq-fpga/.
[10] Surveillance Video Sample. https://www.youtube.com/watch?v=Bnk6dMQFNa8&t=3s.
[11] USB Interface Adapter EVM. http://www.ti.com/tool/usb-to-gpio.
[12] The Xen Project. http://www.xenproject.org/.
[13] 2011. AMD I/O Virtualization Technology (IOMMU) Specification.
[14] 2016. 7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-bit 1 MSPS Analog-to-digital Converter User Guide, UG480 (v1.8). https://github.com/fpgadeveloper/microzed-custom-ip.
[15] A. M. Azab, K. Swidowski, R. Bhutkar, J. Ma, W. Shen, R. Wang, and P. Ning. 2016. SKEE: A Lightweight Secure Kernel-level Execution Environment for ARM. In *NDSS*.
[16] Y. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei. 2016. When Apache Spark Meets FPGAs: A Case Study for Next-generation DNA Sequencing Acceleration. In *HotCloud*. 64–70.
[17] Y. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei. 2016. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *DAC*. 6.
[18] V. Costan, I. Lebedev, and S. Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security*.
[19] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell. 2008. The Promise of High-Performance Reconfigurable Computing. *Computer* 41, 2 (2008), 69–76.
[20] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. Intel Whitepaper, https://eprint.iacr.org/2016/204.pdf.
[21] N. Hu, M. Ye, and S. Wei. 2017. Surviving Information Leakage Hardware Trojan Attacks Using Hardware Isolation. *IEEE TETC* (2017).
[22] M. Huang, D. Wu, C. H. Yu, Z. Fang, M. Interlandi, T. Condie, and J. Cong. 2016. Programming and Runtime Support to Blaze FPGA Accelerator Deployment at Datacenter Scale. In *SOCC*. 456–469.
[23] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. 2007. Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems. In *S&P*. 281–295.
[24] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, and S. Georg. 2017. How to Break Secure Boot on FPGA SoCs through Malicious Hardware. *CHES* (2017).
[25] V. Jyothi, M. Thoonoli, R. Stern, and R. Karri. 2016. FPGA Trust Zone: Incorporating Trust and Reliability into FPGA Designs. In *ICCD*. 600–605.
[26] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood. 2015. Border Control: Sandboxing Accelerators. In *MICRO*. 470–481.
[27] E. Srikanth. 2014. Zynq-7000 AP SoC Low Power Techniques Part 2 - Measuring ZC702 Power Using TI Fusion Power Designer Tech Tip. *Xilinx* (2014).
[28] C. Stauffer and W. E. L. Grimson. 1999. Adaptive Background Mixture Models for Real-time Tracking. In *CVPR*, Vol. 2. 246–252.
[29] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel. 2015. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development* 59, 1 (2015), 7–1.
[30] H. Sun, K. Sun, Y. Wang, and J. Jing. 2015. TrustOTP: Transforming Smartphones into Secure One-time Password Tokens. In *CCS*. 976–988.
[31] M. Tehranipoor and F. Koushanfar. 2010. A Survey of Hardware Trojan Taxonomy and Detection. In *IEEE Design & Test of Computers*. 10–25.
[32] R. Wang, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. 2013. Steal this Movie: Automatically Bypassing DRM Protection in Streaming Media Services. In *USENIX Security*.
[33] Xilinx Inc. 2016. Zynq-7000 All Programmable SoC Overview. In *DS190*.
[34] M. Ye, N. Hu, and S. Wei. 2016. Lightweight Secure Sensing Using Hardware Isolation. *IEEE SENSORS* (2016).
[35] M. Ye, M. Z. Shahrak, and S. Wei. 2017. PUFSec: Protecting Physical Unclonable Functions Using Hardware Isolation-based System Security Techniques. In *AsianHOST*. 7–12.
[36] Y. Zhai and L. Yin. 2016. CQSTR: Securing Cross-tenant Applications with Cloud Containers. In *SoCC*. 223–236.
[37] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers. 2015. A Hardware Design Language for Timing-sensitive Information-flow Security. In *ASPLOS*. 503–516.