Compromising FPGA SoCs using Malicious Hardware Blocks

Nisha Jacob[*], Carsten Rolfes[*], Andreas Zankl[*], Johann Heyszl[*], Georg Sigl[*][†]

[*]*Fraunhofer Institute for Applied and Integrated Security (AISEC)*, Munich, Germany,
Email: {nisha.jacob, carsten.rolfes, andreas.zankl, johann.heyszl, georg.sigl}@aisec.fraunhofer.de
[†]*Technische Universität München*, Munich, Germany,
Email: sigl@tum.de

***Abstract*—Modern FPGA System-on-Chips (SoCs) combine high performance application processors with reconfigurable hardware. This allows to enhance complex software systems with reconfigurable hardware accelerators. Unfortunately, even when state-of-the-art software security mechanisms are implemented, this combination creates new security threats. Attacks on the software are now possible through the reconfigurable hardware as these cores share resources with the processor and may contain unwanted functionality. In this paper, we discuss software protection mechanisms offered in conventional SoCs and how they can be circumvented by malicious hardware blocks. As a concrete example, we show how the malicious functionality within an IP core accesses and replaces critical memory sections. We refer to this type of attacks as hardware-assisted attacks against running software systems. We carry-out a proof-of-concept on the Xilinx Zynq device which runs a Linux OS and a software application that verifies system updates. The malicious IP core replaces the public key used to verify system updates, thus, allowing an attacker to maliciously update the FPGA SoC. Additionally, we propose a countermeasure that can be applied against such threats in the form of a security wrapper for hardware modules.**

***Index Terms*—FPGA SoCs, Zynq, Third party IP, Hardware-assisted attacks**

I. Introduction

Today's embedded systems often use powerful System-on-Chips (SoCs) executing a complex software system including a rich Operating System (OS). They are used in various sectors like automation, defense, aerospace, automotive, and medical appliances due to their high performance at low costs and low power. In such an embedded system, the software is under the threat of different attacks like code corruption or control flow hijacking which exploit weaknesses in the software implementation e.g., pointers going out of bounds or dangling pointers [1]. It is common to deploy protection techniques like secure boot, run-time integrity checks, isolated execution environments, and memory separation mechanisms in the OS, or through a hypervisor which is then supported by hardware features like a Memory Management Unit (MMU) and Input-Output MMU (IOMMU). These countermeasures are designed to enable isolation of software processes and, thus, prevent a process from maliciously interfering with other processes or data. Essentially, software processes are isolated so that a successful attack on one process cannot propagate and compromise the entire system. Through the years, a lot of effort has been spent to develop these robust protection mechanisms.

With the rapid development in the context of the internet-of-things, the need for secure, flexible and performance-driven embedded systems is growing. Since recently, new and interesting SoCs platform known as FPGA SoCs have been developed. FPGA SoCs integrate a hard-core processor system and an FPGA fabric on the same die. The hard-core processor can be used to run complex software, while the reconfigurable logic can be used to replace slow software functions e.g. cryptographic functions with efficient and high performance custom hardware accelerators. Thus, giving system architects more control over the design and performance. Further, with FPGA SoCs, the designers have the advantage of update capabilities for the software and also for reconfiguring the hardware on demand. With FPGA SoCs, designers are more flexible and can directly add new hardware accelerators, which they can download from an Intellectual Property (IP) store. As such systems are getting more versatile, they will likely become a popular choice for embedded systems in the future.

Unfortunately, the flexibility and reconfigurability offered by FPGA SoCs can be exploited by hidden functionality in third party-sourced IP cores (also known as hardware accelerators). Hidden functionalities in IP cores open up new vulnerabilities that were previously not possible with standard SoC-based embedded systems whose hardware accelerators and components are hard-wired. A common practice for a system designer is to buy hardware IP cores from third party IP vendors, as hardware design is complex and requires different engineering background when compared to software system designers. By integrating external IP cores, the design process can be accelerated and thus the time-to-market can be shortened.

These IP cores may be subject to threats, like malicious modification or inclusion of hidden functionality [2], [3]. The effects of such malicious IP cores generally range from denial-of-service [4] to severe attacks like leaking sensitive information from the device. The information can be leaked through covert channels like, power [5] or wireless [6] or manipulation of data being used by other parts of the system [7], [8]. In some cases, the IP cores may be delivered as a post-synthesis

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

netlist that designers can directly plug into their design. For instance, Xilinx offers such a soft IP core [9] that can be used for run-time integrity monitoring to detect tampering on the FPGA. For such cores, designers do not have the opportunity to review the source code and have to rely on the functional descriptions provided by the vendor. In another case, where the source code for the IP core is provided, there could still be hidden functionality as the IP cores may not always be from trusted sources. Extensive tests must be conducted to ensure the integrity of IP cores [10]. However, thorough testing of IP cores is a time consuming and difficult process. Typically, the IP vendors provide a testbench along with the IP core which can be used to test the core. System designers integrating the IP core would use the provided testbenches to verify the functionality of the IP core. These testbenches may not cover all the possible test cases. Thus, finding a small stealthy malicious function can be extremely difficult.

The IP cores on an FPGA SoC are connected through a shared bus system. A malicious IP core connected on this bus may be used to attack other resources connected on the same bus. In this work, we focus on hidden functionality in third party IP cores that can be used to circumvent memory protection and corrupt the software running on the processing system. We refer to this as hardware-assisted attacks against running software systems. For demonstration purposes, a malicious IP core is designed for the Xilinx Zynq platform which is capable of finding and replacing the public key in memory which is used for the authentication of system updates. The core scans the memory at run-time for this public key. Once the key has been located, it is replaced by a pre-defined key known to the attacker. This allows an attacker to later maliciously update and alter the system. Thus, a malicious hardware IP core is used to access unauthorised regions of memory and alter its contents. As an effective countermeasure, we propose a security wrapper for IP cores that can be easily integrated by the system designers to protect the system against such threats.

The remainder of the paper is organised as follows: Section II provides an overview of known threats to embedded systems and a summary of our contributions. In Section III we describe our proof-of-concept of our findings. Section IV provides the implementation details of the attack. Finally, we conclude this work in Section V.

II. Vulnerabilities of FPGA System-on-Chips

In this section, we introduce FPGA SoCs and the security in conventional embedded systems. Furthermore, this section also provides an introduction into our findings.

A. *FPGA System-on-Chips*

FPGA SoCs are a new kind of platform for embedded systems that bring together the best of two worlds, namely embedded processing systems and reconfigurable logic systems, into a single platform. At present, there are three vendors manufacturing FPGA SoCs, namely Xilinx (Zynq-7000 and Zynq UltraScale+) [11], [12], Altera (Cyclone V, Arria V, Arria 10 and Stratix 10) [13] and Microsemi (SmartFusion, SmartFusion2) [14]. Figure 1 depicts a general overview of FPGA SoCs. They consist of two main building blocks a hard-core processing system with dedicated peripherals and, an FPGA fabric for integrating custom hardware accelerators. In addition to these two building blocks, FPGA SoCs usually include a small on-chip memory for storing sensitive data, memory controllers to access external memory, a DMA controller for high speed data transfers, and multiple communication interfaces. Furthermore, the processing system may contain caches to enhance the performance of the system. The processing system may also include hardware-based protection mechanisms like an MMU, an IOMMU and an isolated execution environment, e.g. ARM's TrustZone, to enable run-time security for software applications running on the processor. In addition, a FPGA SoC may also contain cryptographic cores to decrypt and authenticate the FPGA bitstreams and software applications running on the processor. These cryptographic cores are often not accessible by the user design. Hence, designers may include additional cryptographic accelerators as third party IP blocks. The different IP cores and peripherals are connected using either proprietary or standard bus systems like the ARM's AMBA [15], IBM's CoreConnect [16] and Opencore's Wishbone [17].

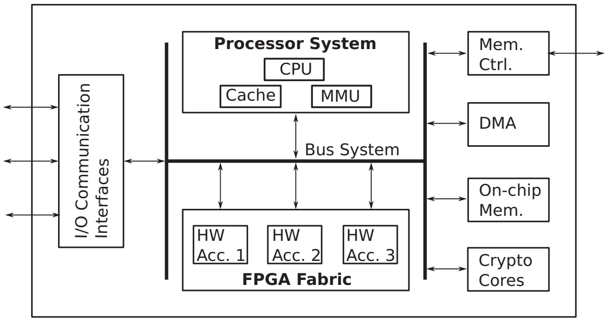

Fig. 1. General Architecture of FPGA SoCs

B. *Security in Conventional SoC-based Embedded Systems*

Isolation of software parts on embedded systems is an important topic in IT security as it can prevent successful attacks in one part of the software from corrupting other parts. In this section, we describe the different protection mechanisms currently available for embedded systems.

A Memory Management Unit (MMU) is a hardware unit that handles memory separation for the OS. The MMU is responsible for translating the virtual memory addresses to the physical memory addresses. Further, it partitions the memory into different pages and assigns different attributes, such as cacheable, read-write or read-only, for each page of the memory. This way, the MMU restricts the access of applications running on the processor to the physical memory shared between different processes.

Newer SoCs also come with an Input-Output MMU (IOMMU) [18] (also called System-MMU), which works similar to an MMU but for peripherals on the SoC that have direct access to memory. An IOMMU performs both, memory translation as well as memory protection for the peripherals on the SoC. The IOMMU also handles the remapping of memory and I/O in the case of virtualised systems.

Another solution developed for the isolation of parts of the software and data is an isolated execution environment such as ARM TrustZone [19]. The TrustZone is a hardware-based protection technique that divides the SoC into two worlds, a secure and a normal world. The TrustZone divides the memory, bus transactions, interrupts and peripherals into secure and normal transactions. Peripherals and memory associated with the secure world are completely isolated from the normal world. The memory and peripherals associated with the normal world do not have direct access to the secure world. Thus, even if the normal execution environment is corrupted, the sensitive information in the secure world of the TrustZone will not be compromised. A security monitor manages the switching between the two worlds. Typically the secure world will contain very minimal applications and interfaces, like cryptographic accelerators and cryptographic keys, while everything else is placed in the normal world. A standard operating system is generally placed in the normal world.

All these solutions have evolved over several years and shown to be useful for SoC-based embedded systems. However, with the addition of reconfigurable logic on the same die, new vulnerabilities rise.

C. Attack Vectors for FPGA SoCs with Malicious HW

In this section, we summarise the goals and contributions of this work. FPGA SoCs share several resources between the FPGA and processor, e.g. the on-chip memory, the external memory and the bus system. A novel attack vector for FPGA SoCs is the inclusion of IP cores in the reconfigurable logic that maliciously exploit resources shared between the processor and the FPGA fabric. For instance, an IP core in the FPGA fabric may exploit the shared bus system by sniffing or corrupting the data being sent on the bus. Another way is to exploit the shared memory.

In this work, we show how the shared memory can be used to retrieve or manipulate sensitive data, while they are loaded or stored in the RAM before being used by a software application. Shared RAM memory is chosen as our target to demonstrate our proof-of-concept as both SoC and FPGA fabric can access and modify its contents. For instance, the memory can be scanned for sensitive data like keys that are used during secure boot or secure update. Furthermore, the memory can be scanned for flags or jump addresses, which could be manipulated to modify the control flow. Once the memory location of interest is identified, the next step is to manipulate it while not causing a partial or complete system failure. The attacker can overwrite the memory with own code/data and thus modify the functionality of the system, e.g. modifying data in order to bypass an authentication phase. Alternatively, an attacker could also leak some sensitive information like keys used for data encryption.

III. Proof-of-Concept Attack on an FPGA SoC Running a Secure Update

In this section, we describe a proof-of-concept of a malicious IP core on a FPGA SoC that compromises an embedded system running an operating system and an application to verify secure remote updates. An update may be delivered to the FPGA SoC through a remote server or may be downloaded from an update server and transferred to the FPGA SoC [20]. In both cases, the update will first have to be verified before it is accepted. Authentication and encryption can be used to verify the authenticity and maintain confidentiality.

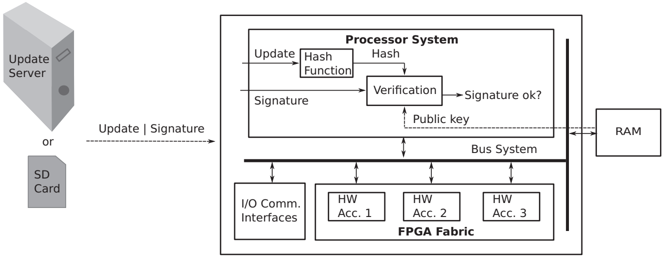

Fig. 2. Secure Update on FPGA SoCs

Figure 2 illustrates the process for verification of such secure updates. It consists of a server from which the update can be downloaded and an FPGA SoC performing the signature verification process. On the server side, the hash of the update is computed and signed using the server's private key. The update along with the signature is delivered to the FPGA SoC. On the embedded system, an application with the help of a hardware accelerator (if available) or a software library, computes the hash of the update. Next, the signature of the update, the computed hash and the public key stored in the FPGA SoC are passed to the verification process. If the signature verification process is passed, the update is accepted by the system.

The goal of the attacker is to manipulate the signature verification process using hardware assistance. We built a custom hardware IP core that will be included in the FPGA fabric of the FPGA SoC. In addition to its regular and documented functions, the IP core also includes hidden functionality. The core maliciously scans the memory for the public key used for authentication and then replaces this key with a predefined key known to the attacker at run time. By replacing this key with another key known to an attacker, the attacker is able to maliciously update the FPGA SoC.

In this work, we identify the public key by scanning for the header of *pem* formatted keys [21]. Alternatively, the mathematical properties of asymmetric keys can be exploited to identify keys in the RAM [22]. In embedded systems, with limited resources, this key may be declared statically within the program binary to avoid the overhead of file handling. Alternatively, a keyfile may be accessed using a file handler and then, its contents are copied to a buffer allocated in the stack or heap depending on the implementation. In both cases, the key will be loaded into the RAM, although at different locations.

Gonzalvo et al. [23] and Li et al. [24] carry out similar attacks. Gonzalvo et al. use the JTAG interface to scan the SoC memory for a system function call and then overwrite it to grant the user application root privileges. For such an attack, the attacker requires physical access to the SoC. Additionally, the attacker also needs an open and unprotected JTAG port on the SoC. Li et al. developed a hardware module to trace the memory accesses of software applications running on the Zynq. All the memory traffic for processing system is routed to the memory tracing IP core located in the FPGA fabric for analysis. In addition to the memory tracing, the hardware module can also be used to maliciously inject code. As an example, the authors show how this core can be used to circumvent log-in password check of Linux systems. The attack requires such a hardware debug core to be present in the final design.

IV. Practical Demonstration on Xilinx Zynq

We implementation a proof-of-concept on the Xilinx Zynq-7000 SoC [12] on a Zedboard Rev. C, which is a development board for the Zynq. The Zynq consists of a Xilinx 7 series FPGA and a dual core ARM cortex A9 processor. The processing system and FPGA fabric have access to the shared resources via the AXI interface. The processing system has two levels of cache and an MMU. The Zynq has several communication interfaces like UART, ethernet, CAN and USB. Additionally, it also includes a small on-chip memory and a DDR memory controller to access a larger external memory. The Zynq supports the trusted execution environment TrustZone from ARM for both, the processing system and the IP cores on the FPGA fabric. Hence, the IP cores on the FPGA fabric can be placed either in the secure or in the normal world depending on the criticality of its function.

A. Software

We implement a realistic embedded system which runs a Linux operating system on the application processor. We use *Linux-xlnx* [25], which a BusyBox Linux distribution for the Zynq devices from Xilinx. We also developed device drivers for the IP core on the FPGA fabric, which the applications running on the OS can use to communicate with the IP cores on demand. Currently, there is one core in the FPGA fabric that can be configured and used by applications running on the OS. The OS runs a software application that authenticates firmware updates using the Elliptic Curve Digital Signature Algorithm (ECDSA). We assume a signed update is delivered to the FPGA SoC using an SD-Card or USB device. The FPGA SoC needs to verify the authenticity of the update. We ported the mbed TLS library, which is an open source TLS library for embedded devices [26], to our application processor to verify the signature and guarantee the authenticity of an incoming update.

B. Hardware Block Including Malicious Functionality

We implemented an AXI-Lite IP core as a place holder for a cryptographic accelerator. This core depicts the characteristics of a cryptographic hash core which would be used to calculate the hash of the update and would also be used as an accelerator for run-time integrity monitoring. For simplicity, we implement a 32-bit XOR operation as a place holder function.

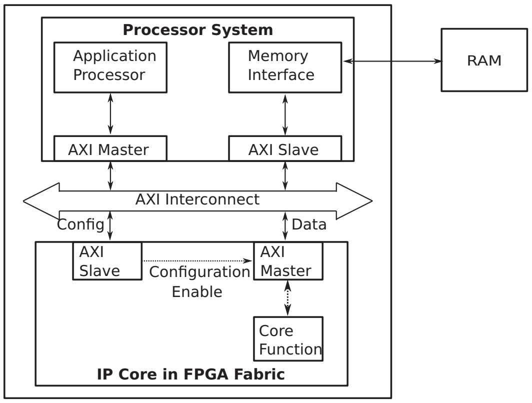

Fig. 3. Design of IP Core

If an IP core performs a hash computation, large amounts of data need to be hashed. Thus, the core would likely be designed to receive configuration information from the processor and then operate autonomously. This interface design is to our knowledge representative and commonly used in practice [27], [28], [29]. This DMA-like interface reduces the load on the processor, as it is not involved in moving the data to and from the memory. The processor passes the source address, destination address and the length of message as configuration information to the IP core followed by an enable signal. Upon the completion of the computation, a done flag is set. A similar setup is described in a white paper from Freescale [30]. In accordance to common practise, we designed our IP core to have two ports as shown in Figure 3. An AXI-lite slave interface through which the enable and configuration parameters for the IP core are passed from the processor. An AXI-lite master interface through which the core reads data from the memory, performs its core function based on the configuration parameters passed from the processor and writes the results back to the memory.

Besides the desired and documented functionalities, the IP core performs the following hidden functions:

1) Scan the RAM memory in the background.
2) Identify the header of the `pem` formatted public key (`− − − − −BEGIN PUBLIC KEY− − − − −`).
3) Overwrite the stored public key with a key known to the attacker.

These hidden functions are performed in succession to regular and documented operations of the core. The key scanning operation is executed after every regular read operation of the core. Starting from the base address of the user memory, the memory is incrementally scanned with every regular read operation. The scanning resumes from the last memory address during subsequent read operations. As the keys will normally be loaded to the user space memory, it would be sufficient to only scan this section of the memory. If a standard Linux kernel is used the user space address mapping can be found publicly. In the cases, where this is not public knowledge, the whole memory would need to be scanned for the key. This is feasible, if the IP core is performing run-time integrity checks or remote attestation, the whole memory needs to be verified to show that the system is running unaltered software. As a result, the key scan operation will also scan the whole memory and the key header can be located in the memory. Once the key header pattern is identified, the key is overwritten with a new key hard coded in the design file. The overwriting of the key is performed before the core performs its documented write operation. By performing the hidden functionality in between the regular functions, the hidden functions can be disguised as legitimate operations of the IP core. Thus, making their detection harder.

C. Results

The described implementation has been tested and as an important result we were able to successfully modify the public key at run-time using a malicious IP core in the FPGA SoC.

Our malicious functionality requires 38 additional flip-flops and 138 additional LUTs for scanning and storing the new public key. A SHA-384/512 IP core for the same FPGA family without the AXI interfaces requires 2526 LUTs (the interfaces require additional resources) [31]. While a high throughput AES-GCM for the same FPGA family with an interface similar to our set-up requires 22729 LUTs [27]. From the above we learn that our malicious functionality may have a relatively small area overhead. Currently, the malicious scanning is only performed when the IP core is being used by the processor. Further, the malicious function scans the same length of memory as the regular read operation. Each read operation reads 32-bit words. In the worst-case scenario the whole 512MB of memory needs to be scanned. As already discussed the region of memory to be scanned can be reduced based on publicly available data of the kernel memory map.

The goal of this work is to highlight the attack vector in FPGA SoCs, thus the area and performance of the hidden functionality is not the focus. This attack vector can be further extended to scan and modify flags, system function calls or symmetric keys. These attacks could be carried out with lesser hardware resources as we do not need to store large payloads in the hardware.

D. Discussion

Through this work, we can see that with the currently available security mechanisms on the Zynq such attacks cannot be prevented. Hence designers need to consider additional mechanisms to protect the systems against such threats.

The different memory protection mechanisms offered in the Zynq are discussed below. The Zynq incorporates the TrustZone for both the application processor as well as for the IP cores in the FPGA fabric. In our scenario, as the IP core also has a master interface, it is capable of setting the security bit for its memory accesses and is not set by the processor. Similarly, the MMU cannot control peripheral memory accesses as they work independent of the processor.

The AXI interconnect on the Zynq can be programmed at design time to restrict the access of peripherals to the memory [12]. By default, no restrictions on the AXI bus are implemented. In our opinion, the default setting is preferable, as it might be impractical to define the restrictions of the peripherals during the hardware design stage, as the hardware designer might not be completely aware of the software system.

Additionally, we observed that due to the incoherency between the shared RAM and the cache, the contents of shared RAM were not always correctly updated. However, in a real-world scenario, where the system runs multiple applications simultaneously, the cache would be flushed and reloaded with other data. Thus, the cache may not influence the attack.

E. Prevention

One general way to protect these systems is to conduct a thorough code review of the IP cores and check the coverage of the testbenches supplied with IP cores before integrating them into a system. Only thoroughly tested cores can be integrated into the system and thus prevent the whole system from being compromised by untrusted IP cores. However, lack of time and competence can lead to small and stealthy malicious functionality being overlooked. Furthermore, in many cases, the IP core is supplied as a netlist which contains no RTL description. Thus, further increasing the difficulty for a review.

Another way to prevent such attacks is to integrate an IOMMU which is called as System MMU (SMMU) by ARM [32]. Using the IOMMU, the access rights of the peripherals can be dynamically managed. An IOMMU is available in the newest high-end FPGA SoCs like the Xilinx UltraScale+ [11] and Stratix 10 [33]. These new FPGA SoCs are significantly more powerful and expensive when compared to the FPGA SoCs like the Xilinx's Zynq and Altera's Arria 10. As our FPGA SoC does not have an IOMMU, an IP core can be developed, which emulates the behaviour of an IOMMU. Brunel et al. [34] develop such an IP core for SoCs. However, this core is resource intensive. Furthermore, it also includes features like encryption and decryption of data written and read from the external memory, which might not always be necessary for typical embedded systems applications. Instead, a lightweight AXI-wrapper can be integrated to every IP core with direct access to the memory. An AXI-wrapper is a module that encapsulates the hardware core. The hardware core is then connected to the bus via the wrapper. As described in Section IV-B, a software application which wants to use such an IP core will send configuration information to the slave interface. Based on this configuration information, the master interface performs its core function. The configuration information typically consists of a start address, destination address and the length of the transaction. Using this configuration information, the AXI-wrapper can be developed to monitor the transactions of the master interface. The main functionality of such a AXI-wrapper would be to allow the IP core access only the memory range specified by the software application. If the IP core tries to access any memory outside of this region an alarm is raised. Further, this wrapper can also be used to set the TrustZone security bit of the master interface to prevent the IP core from maliciously accessing secure memory regions.

V. Conclusion

We demonstrated how the software in FPGA SoCs can be corrupted using malicious hardware cores. As a proof-of-concept, we show how the secure update process can be manipulated using a malicious core in the FPGA. A general way to protect against this is to thoroughly test all third party IP cores before integration. However, this is a complex and time consuming process. In newer FPGA SoCs, these threats can be handled through the use of an IOMMU. However, the lower-end devices which are already in use, are still at risk as they do not have an IOMMU. For such systems, a security enhanced AXI-wrapper can be developed that prevents malicious cores from accessing unauthorised sections of the memory.

References

[1] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal War in Memory," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 48–62. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.13

[2] M. Tehranipoor, H. Salmani, X. Zhang, M. Wang, R. Karri, J. Rajendran, and K. Rosenfeld, "Trustworthy Hardware: Trojan Detection and Design-for-Trust Challenges," *IEEE Computer*, vol. 44, no. 7, pp. 66–74, 2011.

[3] X. Zhang and M. Tehranipoor, "Case Study: Detecting Hardware Trojans in Third-Party Digital IP Cores," in *HOST*, 2011, pp. 67–70.

[4] R. Chakraborty, I. Saha, A. Palchaudhuri, and G. Naik, "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream," *Design Test, IEEE*, vol. 30, no. 2, pp. 45–54, 2013.

[5] Y. Jin and Y. Makris, "Hardware Trojans in Wireless Cryptographic ICs," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.

[6] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering," in *CHES*, 2009, pp. 382–395.

[7] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and Implementing Malicious Hardware," in *Networked Systems Design and Implementation*, 2008.

[8] P. Swierczynski, M. Fyrbiak, P. Koppe, A. Moradi, and C. Paar, "Interdiction in Practice - Hardware Trojan Against a High-Security USB Flash Drive," *IACR Cryptology ePrint Archive*, vol. 2015, p. 768, 2015. [Online]. Available: http://eprint.iacr.org/2015/768

[9] Xilinx Inc, "Security Monitor IP," http://www.xilinx.com/support/documentation/product-briefs/security-monitor-ip-core-product-brief.pdf, 2015.

[10] M. Banga and M. S. Hsiao, "Trusted RTL: Trojan Detection Methodology in Pre-silicon Designs," in *HOST*, 2010, pp. 56–59.

[11] Xilinx Inc, "UltraScale Architecture Configuration: User Guide ," 2015.

[12] Xilinx Inc, "Zynq-7000, All Programmable SoC, Technical Reference Manual." [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[13] Altera Corporation, "Altera SoC Portfolio," https://www.altera.com/products/soc/portfolio.html.

[14] Microsemi Corporation, "SmartFusion2 SoC FPGAs," http://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion2.

[15] ARM, "AMBA Specifications," http://www.arm.com/products/system-ip/amba-specifications.php.

[16] IBM, "CoreConnect," http://www2.informatik.hu-berlin.de/ fwinkler/psvfpga/amirix/CoreConnect.pdf.

[17] OpenCores, "Wishbone-SoC Interconnect," http://opencores.org/opencores,wishbone.

[18] AMD, "I/O Memory Management Unit," http://developer.amd.com/wordpress/media/2012/10/48882.pdf, 2011.

[19] T. Alves and D. Felton, "TrustZone: Integrated Hardware and Software Security," in *ARM white paper*, 2004.

[20] Fiat Chrysler Automobiles, "UCONNECT Software Update," https://www.driveuconnect.com/software-update/.

[21] All your private keys are belong to us , "Tobias Klein," http://www.trapkit.de/research/sslkeyfinder/keyfinder_v1.0_20060205.pdf, 2006.

[22] A. Shamir and N. Someren, "Playing 'Hide and Seek' with Stored Keys," in *Financial Cryptography: Third International Conference, FC'99 Anguilla, British West Indies, February 22–25, 1999 Proceedings*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 118–124. [Online]. Available: http://dx.doi.org/10.1007/3-540-48390-X_9

[23] B. Gonzalvo, E. Bourbao, F. Majéric, and L. Bossue, "JTAG Combined Attacks," in *TRUDEVICE 2015- 4th Workshop on Secure Hardware and Security Evaluation*, Saint-Malo, France, 2015, pp. 56–69.

[24] L. W. Li, G. Duc, and R. Pacalet, "Hardware-assisted Memory Tracing on New SoCs Embedding FPGA Fabrics," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015. New York, NY, USA: ACM, 2015, pp. 461–470. [Online]. Available: http://doi.acm.org/10.1145/2818000.2818030

[25] Xilinx Inc, "linux-xlnx," https://github.com/Xilinx/linux-xlnx.

[26] ARM mbed, "mbed TLS," https://tls.mbed.org/.

[27] BarcoSilex, "BA415-AES-GCM 10 to 100 Gbps IP Core," http://www.xilinx.com/products/intellectual-property/1-4sw1c9.html, 2015.

[28] Ensilica, "Ensilica eSi - SHA-256," http://www.ensilica.com/wp-content/uploads/eSi-SHA-256.pdf, 2013.

[29] BarcoSilex, "BA413-SHA1, SHA2 and HMAC IP Core," http://www.barco-silex.com/ip-cores/encryption-engine/BA413, 2016.

[30] Freescale Semiconductor, "Understanding Cryptographic Performance," http://www.embeddeddeveloper.com/news_letter/files/CRYPTOWP_Rev2.pdf, 2008.

[31] Helion, "HTSHA-FAST64: Fast SHA-384/512 Hashing," http://www.xilinx.com/products/intellectual-property/1-8dyf-612.html, 2016.

[32] ARM, "System Memory Management Unit," http://infocenter.arm.com/help/topic/com.arm.doc.ihi0062d.b/IHI0062D_b_system_mmu_architecture_specification.pdf.

[33] Altera Corporation, "Stratix 10 Secure Device Manager Provides Best-in-Class FPGA and SoC Security," 2015. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01252-secure-device-manager-for-fpga-soc-security.pdf

[34] J. Brunel, R. Pacalet, S. Ouaarab, and G. Duc, "SecBus, a Software/Hardware Architecture for Securing External Memories," in *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2014, Oxford, United Kingdom, April 8-11, 2014*, 2014, pp. 277–282. [Online]. Available: http://dx.doi.org/10.1109/MobileCloud.2014.49