Securing Hardware Accelerators: a New Challenge for High-Level Synthesis

(Perspective Paper)

Christian Pilato, *Member, IEEE,* Siddharth Garg, Kaijie Wu,
Ramesh Karri, *Senior Member, IEEE* and Francesco Regazzoni, *Member, IEEE*

***Abstract*—High-level synthesis (HLS) tools have made significant progress in the past few years, improving the design productivity for hardware accelerators and becoming mainstream in industry to create specialized System-on-Chip (SoC) architectures. Increasing the level of security of these heterogeneous architectures is becoming critical. However, state-of-the-art security countermeasures are still applied only to the code executing on the processor cores or manually implemented into the generated components, leading to suboptimal and sometimes even insecure designs. This paper discusses extensions to HLS tools for creating secure heterogeneous architectures.**

***Index Terms*—High-Level Synthesis, Hardware Security.**

I. Introduction

WE are entering the era of the *Internet of Things* (IoT), where about 200 billion "things" will be connected by 2020 [1]. Each of these systems is becoming increasingly heterogeneous, combining processor cores and specialized accelerators into the same chip to implement System-on-Chip (SoC) architectures. Since the SoC complexity is growing, these systems will be designed by re-using components (more than 90% by 2020 [2], see Fig. 1). Designers will increasingly use *high-level synthesis* (HLS) to raise the abstraction level so as to improve the design productivity [3].

These heterogeneous systems are also used in critical systems such as aircraft, automobiles, banks, and medical devices, where security is a major concern [4]. These systems should not leak secrets to unclassified outputs and untrusted execution should never access or affect critical information. These and related security concerns are doubling spending on cybersecurity in the last five years (see Fig. 1), and we expect this trend to continue [5]. Securing heterogeneous SoCs requires careful analysis and a complete understanding of hardware vulnerabilities. For example, the designer must understand how the information is elaborated to avoid leaking sensitive information or tampering by injecting malicious data [6]. Furthermore, the components should be protected from side-channel attacks [7]. Several countermeasures have been proposed to thwart such attacks, but they have not been considered *during* the generation of SoC components in a systematic and comprehensive way. When combined and applied *after* the design process to avoid leaving the SoC unprotected, it potentially leads to suboptimal solutions.

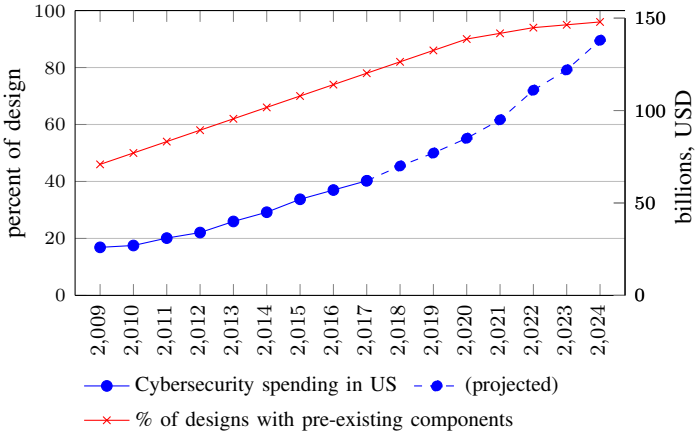

Fig. 1. There is an increase in the percentage of reused components in SoCs (source: ITRS [2]). There is also an increase in spending in the U.S. on cybersecurity (Source: TIA [5]).

This paper highlights emerging challenges that HLS tools will have to tackle to ensure security of the generated hardware accelerators. Hardware security must be considered as a primary objective side-by-side performance and power in the optimization process during HLS. In our vision, this line of research will be important to enable the design of *secure hardware accelerators*, a fundamental block towards secure-by-construction systems. We first describe hardware accelerators and identify security vulnerabilities (Section II). Then, we present state-of-the-art solutions for such challenges (Section III) and discuss how these security solutions can be integrated into future HLS tools (Section IV).

II. Vulnerabilities in Hardware Accelerators

A hardware accelerator is a component tailored to execute a specific functionality. These specialized components are able to improve performance (10-100×) and lower energy consumption (100-1,000×) when compared to implementing them in software running on processors [8]. Fig. 2(a) shows the

Manuscript received May 24, 2017; revised October 10, 2017; accepted November 3, 2017. This work has received funding from the EU Commission's H2020 Programme under grant agreement N. 732105, the CERBERO project. This work was also supported by the National Science Foundation (A#: 1526405). This manuscript was recommended for publication by C. Gebotys. (*Corresponding author: Christian Pilato.*)

C. Pilato and F. Regazzoni are with the Advanced Learning and Research Institute (ALaRI), Faculty of Informatics, Università della Svizzera italiana (USI), Lugano, Switzerland (e-mail: christian.pilato@usi.ch).

S. Garg, K. Wu, and R. Karri with the NYU center for cybersecurity (cyber.nyu.edu), New York University (NYU), New York, NY, USA. Karri is also supported in part by CCS-AD, NYU-AD. Karri and Garg are supported in part by Boeing.

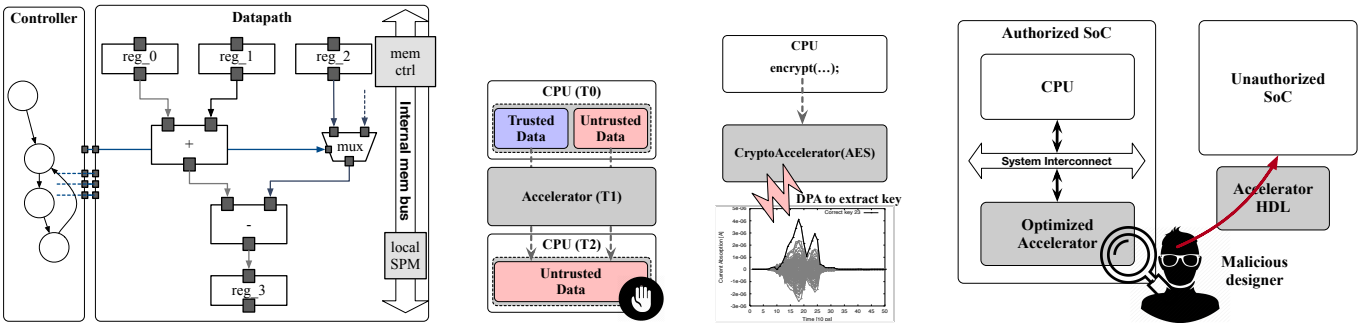

Fig. 2. (a) accelerator microarchitecture, along with examples of vulnerabilities: (b) data produced by the accelerator at time T1 cannot be considered secure by the following CPU execution at time T2, (c) side-channel attacks for key extraction on an AES implementation, and (d) reverse engineering to get an unauthorized copy of the functionality.

microarchitecture of an accelerator, which is composed of the *controller* and the *datapath*. The execution of the function is controlled by a finite-state machine (controller) that, based on a set of conditions, determines which operations are executed by the arithmetic resources (datapath) in any given clock cycle. These resources elaborate input data, provided through parameters or stored in memories – either in local directly-accessible scratchpads or external memory accessed through memory controllers – with the possibility of computing on memory addresses (e.g., pointer arithmetic) [9]. This is achieved by daisy chaining all memory components (i.e., local scratchpads and the controller for the external memory). In this way, accelerators can automatically identify the memory location accessed by a memory operation based on the dynamic value of the address, broadening the range of applications that can leverage such heterogeneous building blocks.

Since heterogeneous architectures are leveraging hardware accelerators to provide energy-efficient high-performance computation, such components are an attractive target for attacks. Current protection mechanisms target software execution on processors [10], [11], are manually implemented [12], and introduce significant overheads [13]. The approach is not efficient and scalable when applied to accelerators, requiring revisiting the design process.

We discuss hardware vulnerabilities listed in the CWE list[1], focusing on how to exploit *design errors* and alter the accelerator behavior. First, vulnerabilities in hardware accelerators can be exploited to launch software-based attacks. Even if it is not possible to implement a different functionality as is done by exploiting *buffer overflow* (CWE-121) and *code injection* (CWE-94), one can manipulate input values (either configuration parameters or memory values) to exploit *design errors*. For example, attackers may exploit vulnerabilities in the accelerator controller to launch a wide range of attacks (*CWE-691: Insufficient Control Flow Management*) [14]. Attackers can also exploit vulnerabilities in the SoC architecture. For example, the attacker may tamper with the system bus to insert malicious operations to trigger unauthorized execution of the accelerators (*CWE-284: Improper Access Control*) [15].

If the system is not adequately protected, the resulting execution may be compromised. One can access internal and sensitive data through the output port or via the memory space shared with the attacker (*CWE-485: Insufficient Encapsulation* and *CWE-922: Insecure Storage of Sensitive Information*). Hence, the execution and the outcome of an accelerator are not secure if not adequately verified and protected (see Fig. 2(b)).

Even when the specification of the accelerator is secure, its implementation can be compromised through *physical attacks*, where the adversary exploits the weaknesses of the implementation (*CWE-693: Protection Mechanism Failure*). Side-channel attacks can be used to extract secret data from embedded devices and high-end cloud servers. A paradigmatic example is the Advanced Encryption Standard (AES). The algorithm is mathematically secure, but its physical implementations have been attacked using power and timing attacks (*CWE-326: Inadequate Encryption Strength*). Accelerators can help mitigate side-channel attacks, for instance ensuring constant execution time and thus making timing attacks infeasible. For example, Intel recently added the AES-NI instructions [11]. Accelerators must be protected from a variety of other attacks, including fault-based and side-channels [16]. If not adequately protected, a circuit separated from the rest of the processor can be localized easily, becoming the target of precise power side-channel, ultimately leading to easy key recovery (see Fig. 2(c)).

Besides these hardware vulnerabilities for the end user, secure accelerators should be protected from reverse engineering, insertion of hardware Trojans (*CWE-912: Hidden Functionality*) and unauthorized copy. Otherwise, the technological advantage of the IP provider can be undermined, creating billions of dollars of economical damage [17]. The hardware description of the accelerator depends not only on the initial high-level specification but also on the optimizations selected by the designer and performed by the design tools. Reverse engineering would make all these assets available to unauthorized parties (see Fig. 2(d)).

In a nutshell, since designers are integrating hardware accelerators into their designs, we expect securing these components to become relevant in the coming years.

III. Securing Hardware Accelerators

The proliferation of third-party applications for embedded systems (e.g., in Apple App Store or Google Play) is becoming

[1]The Common Weakness Enumeration List (CWE), http://cwe.mitre.org

a serious threat to the user's privacy since such systems can leak personal information without authorization [10]. Moreover, the complexity of such applications is increasing exponentially. So, part of these applications are accelerated in hardware. This section describes the existing software-based solutions to prevent or detect malicious attacks, highlighting how they could be effectively and efficiently integrated into the design process of hardware accelerators to achieve secure heterogenous computation.

Dynamic Information Flow Tracking. Systems can be compromised by injecting malicious data to execute unauthorized code (e.g., *buffer overflow*). Dynamic Information Flow Tracking (DIFT) prevents such attacks by marking insecure data and tracking their use during the execution [6]. Many DIFT solutions have been proposed for software. The software-based solutions leverage instrumentations of the source code [6], modifications to the virtualization environment [10], or hardware extensions, like external co-processors that process the execution trace in parallel [18]. However, similar solutions do not exist for hardware accelerators, making it impossible to accurately and efficiently track information flows in heterogeneous SoCs. The dedicated microarchitecture of a hardware accelerator cannot execute additional functions. Although the attacker cannot inject code limiting the type of accelerator's misbehaviors, the designer cannot integrate DIFT without changing the microarchitecture. Given the increasing complexity of these components, manual modification of the hardware is becoming impracticable, while modifications to the underlying Boolean gate library is expensive [13]. Conversely, hardware logic for DIFT can be generated during the automatic synthesis of the accelerator and it can be optimized by exploiting high-level information (e.g., constant values, control flows, etc.)

Side-Channel Countermeasures. Malicious attackers undermine mathematically secure algorithms using side-channel attacks on their physical implementations [12]. Security primitives must be protected from side-channel attacks using countermeasures such as enforcing constant time computation to defeat timing attacks or adding randomness to prevent the attacker to extract secret keys using power analysis [7]. Timing attacks can be prevented by ensuring a computation time which is independent of the input data. Similarly, power attacks can be prevented by reducing the amount of computation that is data-dependent or by masking the sensitive data with random values, reducing thus the correlation between computed data and actual data of the algorithm. All these techniques require modifications of the hardware in specific parts of the design, i.e., the ones elaborating the information to be protected. Security primitives are manually identified by the designer, who is then in charge of implementing the protections. This is performed as an additional step once all other design goals have been met. However, this approach limits the optimizations that can be performed for performance and cost. Sometimes, these solutions are not even implemented in a proper way, yielding insecure design. In addition, such attacks depend on the target technology and there is no one-size-fits-all solution. So, a security-aware design flow requires

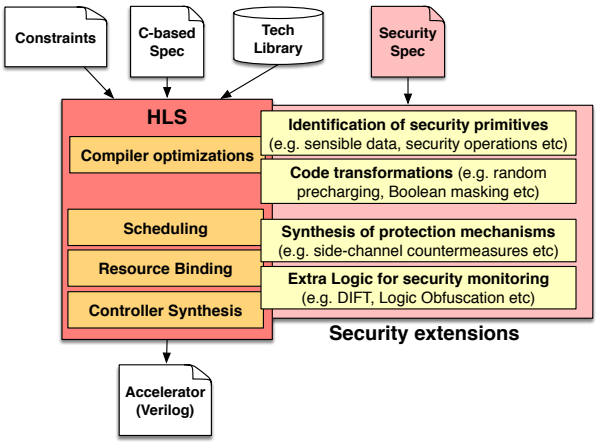

Fig. 3. HLS-based design flow for hardware accelerators with security-aware extensions.

to include an automatic identification of the security primitives to be protected, followed by an automated implementation of side-channel countermeasures that are tailored for the target platform.

Protection Againt Reverse Engineering. Several approaches can be used to protect the generated code against reverse engineering, which is a first step for the insertion of hardware Trojans or for counterfeiting. The designer can obfuscate the functionality of the accelerator by adding extra logic and connections during datapath and controller synthesis [19], [20]. This complicates the reverse-engineering of the design, dramatically increasing the costs for an adversary and reducing the possibilities of inserting the triggers of hardware Trojans. The designer can embed a secret key that the users must know to unlock the functionality of the circuit [21]. Additional approaches watermark and fingerprint accelerators to identify unauthorized copies [22]. All these solutions require modifications to the design during the creation of the accelerator microarchitecture. While simple changes to the gate-level netlists provide low protection, applying these techniques during HLS offer many opportunities to generate more secure accelerators [23].

IV. Security Extensions to High-Level Synthesis

HLS automatically translates a high-level specification of a functionality (e.g., in C/C++/SystemC) into the corresponding RTL implementation [3], [24], as shown in Fig. 3. It receives as input the description of the function, along with synthesis constraints and the target technology library. The HLS flow starts with compiler optimizations (e.g., constant propagation, loop transformations, and bitwidth analysis) and then performs scheduling (to determine which operations are executed in each clock cycle), resource and interconnection binding (to determine which hardware resources are used and how they are interconnected), and controller synthesis (to generate the controlling logic). Finally, it produces the HDL description ready for logic synthesis. Several algorithms have been proposed to improve the performance (either latency or throughput) and the cost (either resources or power consumption) [3]. HLS tools have made significant progress in the last years and have

been successfully used to design accelerators for a variety of applications [25], [26]. However, they do not address security issues. To the best of our knowledge, no commercial HLS tools implement security countermeasures as part of their design flow. This is a limitation because guaranteeing the security of pervasively and massively connected devices is extremely complex. However, it is also an opportunity because HLS tools are a natural place to embed the security mechanisms described above.

We envision extending the traditional HLS flow to include security as shown in the right-hand side of Fig. 3. Security-aware HLS supports the analysis of the input descriptions at the compiler level to automatically identify and optimize security primitives and the sensible data to protect. Code transformations can be applied to increase the level of security of the input specification through, for example, Boolean masking. Software-based techniques can be complemented with hardware protections that can be synthesized automatically and transparently. The subsequent synthesis process starts from this optimized specification and automatically generates protection mechanisms inside the datapath or the controller. For example, efficient DIFT logic can be inserted during resource and interconnection binding to monitor the data exchanges between trusted and untrusted regions mindful of the performance and resource overheads. Information leakage via side-channels can be mitigated by automatically applying countermeasures such as masking or constant-time routines. Finally, obfuscation can be automatically applied when the modules are generated or during the third-party intellectual property integration. These techniques can also limit the insertion of hardware Trojans [27]–[29].

In conclusion, our ultimate goal is the integration of efficient security protections in accelerators. This can be only achieved by embracing HLS and extending it to comprehensively support hardware security.

V. Conclusion and Future Directions of Work

This paper discussed the security challenges when designing hardware accelerators in heterogeneous SoC architectures. We analyzed hardware vulnerabilities for which state-of-the-art countermeasures are not fully integrated and automated into current design flows for hardware accelerators.

We envision high-level synthesis to be naturally extended for the automatic synthesis of a variety of protection mechanisms. The proposed HLS-based, security-aware methodology will be able to create heterogeneous architectures that are efficient (in terms of performance, energy, and resources) and secure at the same time.

References

[1] H. Bauer, M. Patel, and J. Veira, "The Internet of Things: Sizing up the opportunity," *McKinsey&Company High Tech*, Dec. 2014. [Online]. Available: http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things-sizing-up-the-opportunity

[2] ITRS, "Roadmap (system drivers)," 2009. [Online]. Available: http://www.itrs.org/

[3] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A Survey and Evaluation of FPGA High-Level Synthesis Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[4] D. Selwood, "Heterogeneous Processing, SoCs and FPGAs," *Electronic Engineering Journal*, Aug. 2015. [Online]. Available: http://eejournal.com/archives/articles/20151029-altera/

[5] TIA, "ICT Market Review and Forecast," 2016.

[6] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure Program Execution via Dynamic Information Flow Tracking," in *Proceedings of ASPLOS*, Oct. 2004, pp. 85–96.

[7] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of CRYPTO*, 1999, pp. 388–397.

[8] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *ISSCC Digest of Technical Papers*, Feb. 2014, pp. 10–14.

[9] C. Pilato, F. Ferrandi, and D. Sciuto, "A design methodology to implement memory accesses in High-Level Synthesis," in *Proceedings of CODES+ISSS*, Oct. 2011, pp. 49–58.

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of OSDI*, Oct. 2010, pp. 393–407.

[11] G. Hofemeier and R. Chesebrough, "Introduction to Intel AES-NI and Intel Secure Key Instructions," Jul. 2012. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-intel-aes-ni-and-intel-secure-key-instructions

[12] A. Gornik, A. Moradi, J. Oehm, and C. Paar, "A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1308–1319, Aug. 2015.

[13] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner, "Theoretical fundamentals of gate level information flow tracking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1128–1140, Aug. 2011.

[14] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proceedings of DAC*, Jun. 2016, pp. 1–6.

[15] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Proceedings of ESCAR*, 2004.

[16] F. Regazzoni, T. Eisenbarth, J. Grossschadl, and L. Breveglieri, "Power Attacks Resistance of Cryptographic S-Boxes with Added Error Detection Circuits," in *Proceedings of DFT*, 2007, pp. 508–516.

[17] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[18] H. Kannan, M. Dalton, and C. Kozyrakis, "Decoupling dynamic information flow tracking with a dedicated coprocessor," in *Proceedings of DSN*, Jun. 2009, pp. 105–114.

[19] N. Veeranna and B. C. Schafer, "Efficient behavioral intellectual properties source code obfuscation for high-level synthesis," in *Proceedings of LATS*, Mar. 2017, pp. 1–6.

[20] J. Rajendran, A. Ali, O. Sinanoglu, and R. Karri, "Belling the CAD: Toward security-centric electronic system design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1756–1769, 2015.

[21] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proceedings of DATE*, Mar. 2008, pp. 1069–1074.

[22] C.-H. Chang, M. Potkonjak, and L. Zhang, *Hardware IP Watermarking and Fingerprinting*. Springer, 2016, pp. 329–368.

[23] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, "High-level synthesis for security and trust," in *Proceedings of IOLTS*, Jul. 2013, pp. 232–233.

[24] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An Introduction to High-Level Synthesis," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 8–17, Jul. 2009.

[25] C. Pilato, Q. Xu, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "On the design of scalable and reusable accelerators for big data applications," in *Proceedings of CF*, May 2016, pp. 406–411.

[26] F. D. Robinson, L. H. Crockett, W. H. Nailon, and R. W. Stewart, "High-level synthesis for medical image processing on Systems on Chip: A case study," in *Proceedings of FPL*, Aug. 2016, pp. 1–2.

[27] N. Veeranna and B. Schafer, "Hardware Trojan detection in Behavioral Intellectual Properties (IPs) using Property Checking Techniques," *IEEE Transactions on Emerging Topics in Computing*, Jun. 2016.

[28] N. Veeranna and B. C. Schafer, "Trust filter: Runtime hardware Trojan detection in behavioral MPSoCs," *Journal of Hardware and Systems Security*, pp. 1–12, 2017.

[29] J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: A high-level synthesis approach," *IEEE Trans. on VLSI Systems*, vol. 24, no. 9, pp. 2946–2959, Sep. 2016.