High-Level Synthesis of Security Properties via Software-Level Abstractions

Christian Pilato[1] and Francesco Regazzoni[2,3]
[1]Politecnico di Milano — DEIB, Milan, Italy, christian.pilato@polimi.it
[2]University of Amsterdam, Amsterdam, The Netherlands, f.regazzoni@uva.nl
[3]Università della Svizzera italiana, Lugano, Switzerland, francesco.regazzoni@usi.ch

ABSTRACT

High-level synthesis (HLS) is a key component for the hardware acceleration of applications, especially thanks to the diffusion of reconfigurable devices in many domains, from data centers to edge devices. HLS reduces development times by allowing designers to raise the abstraction level and use automated methods for hardware generation. Since security concerns are becoming more and more relevant for data-intensive applications, we investigate how to abstract security properties and use HLS for their integration with the accelerator functionality. We use the case of dynamic information flow tracking, showing how classic software-level abstractions can be efficiently used to hide implementation details to the designers.

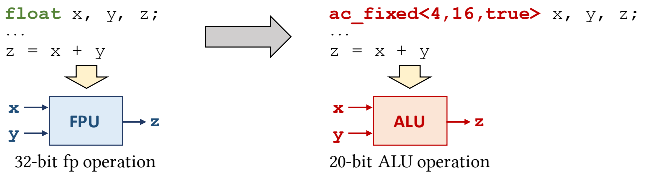
1 INTRODUCTION

The future of computing systems will be necessary data-driven. Collecting and processing large amounts of data will unleash unprecedented knowledge discovery that can improve everyday's life. However, these applications demand not only novel and heterogeneous architectures to delivery energy-efficient high performance but also effective methods to avoid unauthorized operations on the data [12]. On one side, *high-level synthesis* (HLS) is a key enabler for heterogeneous architectures. Abstracting the functionality of a component to the software level and applying automated methods for hardware generation, HLS allows non-expert designers to create more components, specialize their architectures, and reduce design costs. We expect more and more HLS-generated components to be integrated in future systems. On the other hand, dealing with valuable data attracts *malicious actors* that can steal (or alter) sensitive information or use the existing data flow to compromise the system. For example, buffer overflow is a technique to exploit software vulnerabilities to gain control of an application and potentially steal sensitive data. While hardware-assisted security protections are more efficient, their implementation requires to modify the components or the design flow. Integrating data and intellectual property (IP) protection into HLS is interesting [13] but previous attempts require extensive tool modifications [6, 11, 14, 15] or are limited to specific security properties [1].

In this line of research, we are exploring which security protections can be specified at the software level and synthesized transparently during HLS.

2 COMPILER INFRASTRUCTURE FOR HLS

Modern HLS tools leverage state-of-the-art compilers like GCC or LLVM as a frontend to software specifications [9]. Such compilers extract a language-agnostic representation with the essential semantics to synthesize in hardware. They are also used to apply code transformations (e.g., loop optimizations and constant propagation), create a more hardware-friendly description (e.g., code lowering and bit-width optimization), and extract more hardware parallelism (e.g., inlining and memory optimizations). In the following steps, the HLS engine performs temporal and spatial assignment of the operations to derive the corresponding microarchitecture. Software abstractions are widely used to create compact but flexible representations and to hide details to the programmers. Among those, **synthesizable software libraries** and **operator overloading** are common also in HLS. Software libraries can abstract common hardware functions like recurrent functions [17], memory data transfers, and communication protocols. For example, `hlslib` [2] provides libraries to support designers in common optimization steps, like interface and communication synthesis. Operator overloading can associate different implementations to the same operators based on their arguments. For example, Mentor offers `ac_types` that are bit-accurate datatypes for custom precision [8]. Figure 1 shows an example for converting floating-point to fixed-point operations.



```
float x, y, z;
...
z = x + y
```

```
ac_fixed<4,16,true> x, y, z;
...
z = x + y
```

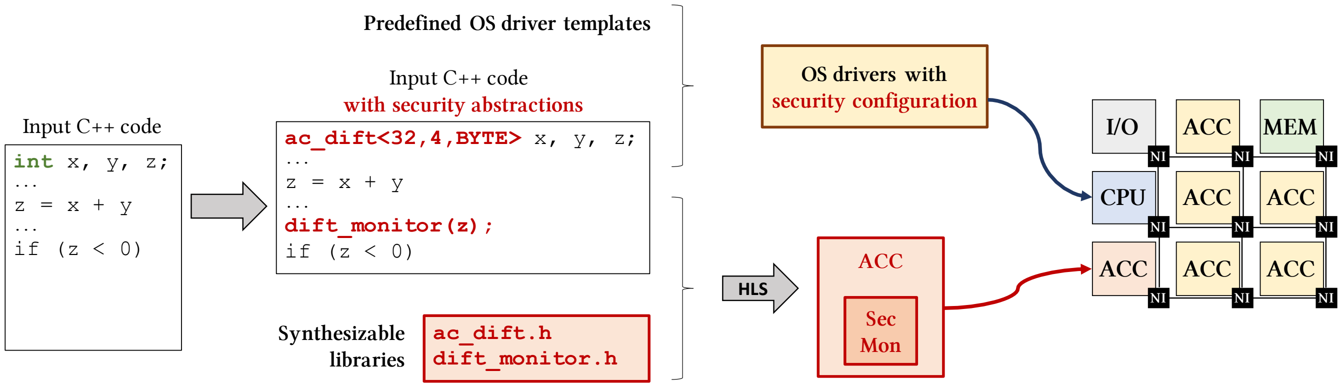
Figure 1: Example of custom datatypes for HLS.

While these solutions are commonly used for optimizing the microarchitecture of hardware accelerators, their adoption for integrating security features is still at the initial stages. Indeed, integrating specialized security components and automatically propagating security properties can be achieved with this approach. However, HLS needs to be carefully tuned to optimize the logic while, at the same time, avoid introducing hardware-level vulnerabilities, like power side-channels [19].

3 PROPAGATION OF SECURITY PROPERTIES

While HLS is good at optimizing classic non-functional requirements (e.g., area, power, and delay), the propagation of security properties and the integration of security protections need more careful investigation. We use the case of **dynamic information**

LATTE '21, April 15, 2021, Virtual, Earth



```
int x, y, z;
...
z = x + y
...
if (z < 0)
```

```
ac_dift<32,4,BYTE> x, y, z;
...
z = x + y
...
dift_monitor(z);
if (z < 0)
```

```
ac_dift.h
dift_monitor.h
```

Figure 2: Software-level abstraction to implement synthesizable dynamic information flow tracking.

flow tracking (DIFT) as a paradigmatic example. DIFT associates a *tag* to selected data of an application to monitor their influence on the program execution and, ultimately, detect security hazards [18]. The integration of specific security policies can help contain the effects of these hazards.

Implementing information flow tracking in hardware is complex and expensive. Researchers proposed several solutions to trade-off accuracy of the taint analysis and hardware cost [4]. Coarse-grained approaches apply DIFT to the "boundaries" of the components [10, 16], while fine-grained taint propagation leads to high area overhead [5] or requires tool modifications to automatically integrate the additional logic [15]. Furthermore, implementing security policies require to generate and integrate proper hardware monitors and assertion-like logic [3]. Designers can use HLS to automatically generate such additional logic, explore the design space of these solutions, and identify the best design point for the target application. HLS can also automatically handle the generation of the monitor components. However, abstracting both hardware and security details is critical for non-expert hardware designers.

4 SOFTWARE-LEVEL ABSTRACTIONS FOR ACCELERATOR-LEVEL DIFT

Since all DIFT elements (tags and propagation rules) are related to additional functionalities of the accelerators, we argue they can be embedded in the initial code with minimal changes and, leveraging a combination of synthesizable libraries and HLS, we can automatically generate DIFT-enhanced accelerators. Figure 2 shows how software-level abstractions can be used to embed DIFT in HLS-generated accelerators.

The input C++ code is modified by the designer to annotate DIFT variables. We use the custom datatype `ac_dift`, where the designer specifies the precision of the variable (like, for example, in `ac_int`), the size of the taint tag, and the type of propagation rules to be used. An additional value provides the default taint information. The HLS tool uses the **custom datatype definition** to include the additional taint variables and the **operator overloading** to synthesize the logic that combines the operator variables and their taint values. In this way, it can automatically compute the operation result along with the associated tag according to the given propagation rule.

The designer also modifies the input code to include *security checkpoints* with specific function calls. The security policies are implemented as **synthesizable libraries**. For example, the function `dift_monitor(x)` requests to check the tag of variable `x`. After applying HLS on the function augmented with the security checkpoint, the accelerator will include a **security monitor**, i.e. a submodule that receives the taint tags and produces control signals based on the tag values and the given security policy. The designer can customize the security policy by specializing the function `dift_monitor`. In case of security hazards, the monitor produces a *security exception* that is trapped and managed by the system. The exception can trigger, for example, a special *interrupt request* that activates system-level protections like component isolation. Predefined Operating System (OS) drivers are customized with the proper configuration of I/O registers to exchange tag information with software [7].

This approach has several potential advantages that are worth to be explored. First, it provides a complete infrastructure that provides implementation support for non-experts. Second, it allows the designers to check DIFT and security policies at a higher level of abstraction, along with the rest of the software code. Third, the DIFT functions inside operator overloading and the monitor libraries are synthesized (and co-optimized) along with the rest of the accelerator's logic. The HLS engine could introduce extra cycles to optimize the schedule and minimize resource utilization, without a perfect *data flow consistency* between baseline and DIFT microarchitectures [15]. However, these optimizations would not affect the DIFT results.

5 CONCLUSION

We discuss code-level extensions to specify security protections that can be later automatically synthesized with HLS. For this, we analyze the case of dynamic information flow tracking and how software-level abstractions can support the designers. This activity opens up an interesting research question: *Which security protections can be effectively abstracted and synthesized with HLS without compromising their security?*

ACKNOLEDGEMENTS

This project is partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).

REFERENCES

[1] H. Badier, C. Pilato, and G. G. J.-C. Le Lann, P. Coussy. 2021. Opportunistic IP Birthmarking using Side Effects of Code Transformations on High-Level Synthesis. In *Proceedings of the ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–4.

[2] J. de Fine Licht and T. Hoefler. 2019. hlslib: Software Engineering for Hardware Design. *arXiv preprint arXiv:1910.04436* (2019).

[3] M. B. Hammouda, P. Coussy, and L. Lagadec. 2017. A Unified Design Flow to Automatically Generate On-Chip Monitors During High-Level Synthesis of Hardware Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 3 (2017), 384–397.

[4] W. Hu, A. Ardeshiricham, and R. Kastner. 2020. Hardware Information Flow Tracking. *Submitted to ACM Computing Surveys* (2020).

[5] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. 2011. Theoretical Fundamentals of Gate Level Information Flow Tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 8 (2011), 1128–1140.

[6] Z. Jiang, S. Dai, G. E. Suh, and Z. Zhang. 2018. High-Level Synthesis with Timing-Sensitive Information Flow Enforcement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[7] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni. 2020. Agile SoC Development with Open ESP. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*.

[8] Mentor, A Siemens Business. 2020. Algorithmic C (AC) Datatypes (v4.1). https://github.com/hlslibs/ac_types. (2020).

[9] 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. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (Oct. 2016), 1591–1604.

[10] L. Piccolboni, G. Di Guglielmo, and L. P. Carloni. 2018. PAGURUS: Low-Overhead Dynamic Information Flow Tracking on Loosely Coupled Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2685–2696.

[11] C. Pilato, K. Basu, M. Shayan, F. Regazzoni, and R. Karri. 2019. High-Level Synthesis of Benevolent Trojans. In *Proceedings of the ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1124–1129.

[12] C. Pilato, S. Bohm, F. Brocheton, J. Castrillon, R. Cevasco, V. Cima, R. Cmar, D. Diamantopoulos, F. Ferrandi, J. Martinovic, G. Palermo, M. Paolino, A. Parodi, L. Pittaluga, D. Raho, F. Regazzoni, K. Slaninova, and C. Hagleitner. 2021. EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms. In *Proceedings of the ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[13] C. Pilato, S. Garg, K. Wu, R. Karri, and F. Regazzoni. 2018. Securing Hardware Accelerators: A New Challenge for High-Level Synthesis. *IEEE Embedded Systems Letters* 10 (2018), 77–80. Issue 3.

[14] C. Pilato, F. Regazzoni, R. Karri, and S. Garg. 2018. TAO: techniques for algorithm-level obfuscation during high-level synthesis. *Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1–6.

[15] C. Pilato, K. Wu, S. Garg, R. Karri, and F. Regazzoni. 2019. TaintHLS: High-Level Synthesis for Dynamic Information Flow Tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38 (2019). Issue 5.

[16] J. Porquet and S. Sethumadhavan. 2013. WHISK: An uncore architecture for Dynamic Information Flow Tracking in heterogeneous embedded SoCs. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 1–9.

[17] D. Richmond, A. Althoff, and R. Kastner. 2018. Synthesizable Higher-Order Functions for C++. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2835–2844.

[18] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. 2004. Secure Program Execution via Dynamic Information Flow Tracking. In *Proceedings of the ACM SIGOPS International Conference on Architectural support for programming languages and operating systems (ASPLOS)*. 85–96.

[19] L. Zhang, D. Mu, W. Hu, Y. Tai, J. Blackstone, and R. Kastner. 2020. Memory-Based High-Level Synthesis Optimizations Security Exploration on the Power Side-Channel. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2124–2137.