October 2017

Bridging the Gap between Software and Hardware Designers using High-Level Synthesis

Christian PILATO [1]
Università della Svizzera italiana (USI), Lugano, Switzerland

Abstract. Modern Systems-on-Chip (SoC) architectures and CPU+FPGA computing platforms are moving towards heterogeneous systems featuring an increasing number of hardware accelerators. These specialized components can deliver energy-efficient high performance, but their design from high-level specifications is usually very complex. Therefore, it is crucial to understand how to design and optimize such components to implement the desired functionality.

This paper discusses the challenges between software programmers and hardware designers, focusing on the state-of-the-art methods based on high-level synthesis (HLS). It also highlights the future research lines for simplifying the creation of complex accelerator-based architectures.

Keywords. high-level synthesis, accelerators, heterogenous systems

1. Introduction

Due to the end of Dennard's scaling, specialized hardware accelerators are increasingly used in heterogeneous System-on-Chip (SoC) architectures and CPU+FPGA platforms to execute selected computational kernels more efficiently. These components can be then turned off when unused to alleviate the power density problem [9]. However, the complexity of these components is growing exponentially, which in turn increases both design and verification costs. Conversely, the market requires such systems to be produced in a shorter time. So, designers are increasingly adopting Electronic System Level (ESL) methodologies based on High-Level Synthesis (HLS) to raise the abstraction level. With HLS, the designer can focus on the high-level description of the functionality to implement, express it in a high-level language (e.g., C, C++, SystemC, etc.), and then use automated methods and tools for the automatic creation of the corresponding specialized hardware [2]. However, this process is complex, especially because the most of the code and semantics used by software programmers is often difficult to be translated into efficiently hardware [25,27]. For instance, the synthesis of pointers requires to understand which data structures are effectively addressed at runtime [27], while different data access patterns may impact on the data layout and, in turn, on the size of the on-chip

[1]Christian Pilato, Advanced Learning and Research Institute (ALaRI), Faculty of Informatics, Università della Svizzera italiana (USI), Via G. Buffi 13, CH-6904, Lugano, Switzerland; E-mail: christian.pilato@usi.ch.

memory [29]. So, HLS developers often impose severe limitations that require extensive and error-prone code rewriting of the high-level specification before the use of HLS. In addition, the interdependence of many optimizations is often unclear and the evaluation of their impact on performance and cost (area or power) generally requires a complete synthesis and simulation. Therefore, an extensive design space exploration is very time consuming since it requires many iterations between code transformations and hardware synthesis [22,30].

This paper presents an overview of recent achievements in high-level synthesis to bridge the gap between *software programmers*, who are in charge of writing the high-level specification, and *hardware designers*, who are responsible for developing or using HLS tools to generate efficient accelerator-based systems. This discussion will cover both algorithmic solutions to synthesize efficient components and methodologies to intelligently use existing tools and improve the design of complex accelerators.

So, after presenting an overview of the high-level synthesis process (Section 2), the two main contributions of this paper are presented: (1) the presentation of current achievements in the design of complex accelerators, including a system-level methodology for the optimization of the local memory (Section 3), and (2) the discussion of the open challenges and the future research lines that we expect for high-level synthesis (Section 4 and Section 5, respectively).

2. Hardware Accelerators and High-Level Synthesis

A hardware accelerator is a component tailored to execute only the specific functionality for which it has been designed. Thanks to its specialized microarchitecture, it is usually able to achieve better performance (up to 10-100$\times$) and lower energy consumption (up to 100-1,000$\times$) than the corresponding software execution [9]. However, this comes at the cost of flexibility: the designer must carefully determine the accelerator's microarchitecture at design time to maximize the performance (by exploiting hardware parallelism), without any possibility of executing a different functionality at runtime. Conversely, adding an extra functionality requires the implementation of extra logic.

Complex accelerators are organized with submodules to reduce the design complexity. Each module is based on the classical Finite State Machine with Data (FSMD) model [39] and includes the following components:

- a **controller**, which determines the operations to execute in each clock cycle. The control flow is represented by a finite state machine that sends proper command signals to the datapath resources based on a set of conditions;
- a **datapath**, which contains the functional units to implement the functionality on the input data and the registers to contain temporary values for the computation. Multiplexers are used to drive the values based on the control flow;
- **memory elements**, which include *scratchpad memories* (SPMs) to locally store data and a *memory interface* to access the external data (e.g., in DRAM).

Each computational resource of the datapath exchange information through registers, local SPMs, or DRAM. Input data are provided through the configuration registers or stored in DRAM and accessed through the memory controller. In particular, local SPMs are heterogeneous and distributed memories, tailored on the data structures to be stored.

These memories are used to execute multiple memory operations in parallel on different data with fixed latency [20,27,29,36], enabling more hardware parallelism. However, such memory blocks requires specialized microarchitecture in case of parallel accesses [27,29]. Memory architectures have been recently proposed to support almost any input specifications, enabling computation even on memory addresses (e.g., pointer arithmetic) [27]. To this end, such memory architectures create a daisy chain with all memory components, including both local memories and controllers for the external memory. An accelerator can thus automatically identify the memory location accessed by a memory request based on the value of the address given at runtime. This allows the implementation of software code with no semantic changes, enabling the possibility of dynamically changing the execution of one task between software and hardware.

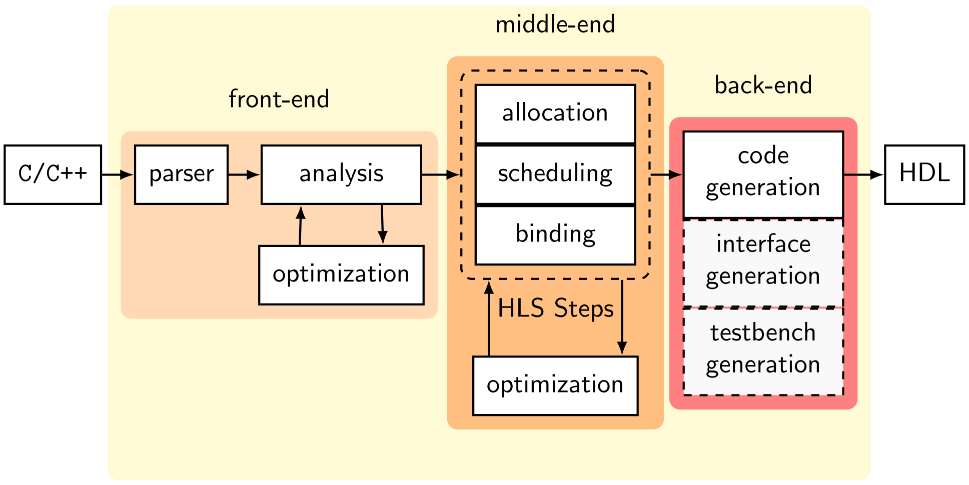

Figure 1. Typical organization of a high-level synthesis tool.

A traditional HLS flow is shown in Figure 1. It interfaces with state-of-the-art compilers (e.g., GCC or LLVM) to parse the input C code, apply compiler optimizations, and extract the resulting IR [20]. They usually leverage the Single Static Assignment (SSA) form [21], which is a machine-independent form that can be easily manipulated and translated into hardware. The SSA form generates a unique identifier for each assignment to the same variable, increasing the number of temporary values but simplifying the subsequent HLS steps. In particular, after the *allocation* phase, which includes the selection of resources (*module allocation*) and memories *memory allocation*, the scheduling is performed to determine the operations to be executed in each clock cycle. Operations scheduled in different clock cycles can potentially reuse the same resources (*module binding*), while temporary values crossing the clock boundaries must be assigned to different registers to be stored (*register binding*) [34]. The last step in the datapath is the *interconnection binding*, where the different resources are interconnected and the corresponding control signals are generated together with the controller (*controller synthesis*).

The synthesis of each accelerator is performed hierarchically, starting from the innermost C functions. In this way, when generating a module, all its submodules have been already generated and they can be properly interconnected as any other datapath resource. Testbenches and interfaces can be also automatically generated for verification and system-level integration, respectively [26].

3. Achievements in the Generation of Accelerator-Based Systems

Raising the abstraction level and using Electronic System Level (ESL) methodologies based on high-level synthesis (HLS) to automatically generate configurations for FPGAs is helping in tackling the current gap in the design complexity. HLS offers benefits to software engineers, enabling them to achieve speed and energy benefits of hardware with limited hardware expertise [20]. HLS also offers benefits to hardware engineers, by allowing them to design systems faster at a high-level abstraction and rapidly explore the design space [22]. This is crucial in the design of complex systems [2] and especially suitable for FPGA design where many alternative implementations can be easily generated, deployed, and compared. HLS has been thus recently applied to a variety of applications (e.g. medical imaging, convolutional neural networks, machine learning), with significant benefits in terms of performance and energy consumption [31,38].

Design space exploration is, indeed, an important step in the design of architectures to understand how to trade off cost (e.g., resources or power) and performance (e.g., latency or throughput). Composable DSE methods are gaining a lot of attention in the past few years [12], together with solutions to speed-up the exploration with estimation models [16]. However, this process is not simple and requires evaluating the impact of the combined transformations towards several optimization goals (e.g., maximizing performance/watt). For instance, applying aggressive loop transformations require more logic to exploit the increased hardware parallelism, but they can bring performance improvements only when supported by a proper multi-port local memory [29].

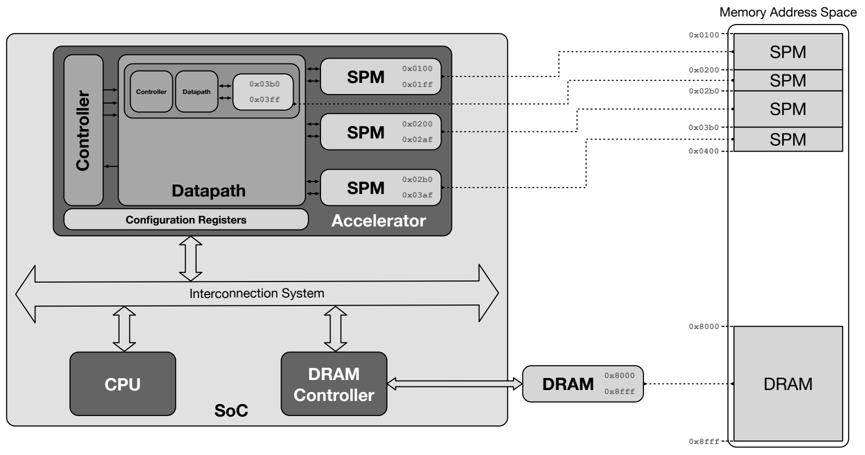

Figure 2. Example of target heterogeneous architecture: an accelerator is composed of submodules hierarchically organized. Each of them may contain one or more *scratchpad memories* (SPMs) to store local data. Rest of the data is stored in DRAM.

Accelerators are usually integrated in heterogeneous architectures in a way similar to the one shown in Fig. 2. The processor core (CPU) executes a software application to prepare the data and configure the accelerator with memory-mapped operations on the *configuration registers* through the interconnection system (e.g., a bus or a network-on-chip) [15]. The data stored in the shared memory (DRAM) are accessed through one or more memory controllers [13]. Accelerators may adopt DMA mechanisms to exchange large data blocks with DRAM [4,29] or SPMs can be used to locally store entire data structures for the entire execution of the accelerator [27]. So the designer must decide partition the data structures at design time and determine where to store each of them. This is usually a trade-off between predictable memory accesses, which are possible when the given structure is stored in a dedicated SPM, and size of the accelerator, which can be reduced by storing more data structures in DRAM. In such architectures, SPM data are accessed with a known latency (e.g., one or two cycles), which greatly simplifies the scheduling of the memory operations and the controller creation. Conversely, it is not possible to guarantee a known latency for external memory accesses. So, the accelerator must implement a memory interface with a latency-insensitive protocol to access the data without affecting the execution correctness, accordingly scheduling the corresponding memory operations [24].

In fact, memory optimization is another important aspect of accelerator design. Since accelerator-based architectures are usually characterized by a huge amount of data to elaborate, the optimization of memory accesses plays a key role in the design of efficient heterogeneous systems [13,29]. However, the memory behavior of an application is highly dependent on the characteristics of the applications, ranging from statically predictable patterns [38] to irregular memory accesses [3], depending on the application domain. The former can be easily optimized with polyhedral-based transformations [37] associated with multi-port memories [29] to satisfy the requirements in terms of parallel accesses. The latter, instead, require dedicated micro-architectures to schedule the memory accesses and hide the latency of the communication with the memory [24].

4. Open Challenges between Software and Hardware Designers

4.1. Memory Synthesis and Optimization

Memory elements are usually responsible for most of the accelerator area and power consumption [4,29] and in the era of the Internet of Things they will become more and more critical. Despite the huge progress made by HLS tools in the recent years [20], very limited attention has been devoted to the concurrent optimization of the accelerator logic and the memory accesses [22]. Pre-synthesis transformations have been proposed to optimize the loops and the memory accesses [37], and to generate resource-efficient multi-port memories [29]. However, modern accelerators must include innovative and dedicated micro-architectural solutions to better manage the memory accesses with respect to the available memory technologies in terms of energy consumption [23] and read/write latency. Future heterogeneous systems will, indeed, feature (1) hybrid memory architectures with different on-chip memories [17], (2) multiple memory controllers that can provide physical access to external memories implemented with different technologies, and (3) adaptive runtime environments to better manage and interleave the memory accesses. Combined this architecture is the only viable solution to exploit much more data parallelism with limited cost in terms of resources and energy consumption. On the other hand, the memory accesses must be analyzed on a larger scale since the compilation phase to better understand the global requirements and match them with the characteristics of the available memories.

4.2. Design and Verification

High-level synthesis provides an important support for the design of heterogeneous architectures. However, this also exacerbates verification issues for the application programmers, especially when the computation is distributed in both software and hardware tasks, and the accelerators are provided by different toolchains. Fast, efficient, and precise methods for debugging are required, featuring the possibility to backtrack the origin of a bug to the exact point of failure [1,5]. Since the amount of computation to be performed in modern heterogeneous applications before activating the bug may be considerable, simulation-based approaches have limited applicability and on-chip debugging is gaining a lot of attention [18,6]. Current methods usually allow hardware designers to add extra logic for on-line monitoring of signals, but there is limited possibility to automatically identify discrepancies with the expected behavior and restrict the area where the error is originated, especially when combining hardware and software execution of multiple IP components.

4.3. Emerging Trends for Hardware Generation and Integration

High-level synthesis tools often rely on state-of-the-art compilers for implementing aggressive code optimizations and improve the generated hardware [20]. Several transformations have been proposed, ranging from speculative code transformations [11] to aggressive loop transformations with polyhedral models and multi-port memories for increasing the hardware parallelism [29,37]. However, such approaches are often limited to specific domains (e.g., stencil computation [32]) and they never consider the use of heterogeneous memory architectures.

To provide effective abstractions exposed to the programmer, future HLS tools must leverage recent developments in heterogeneous programming models (e.g., SYCL) as well as in domain-specific languages (DSLs), which have been successfully demonstrated in different relevant areas such as image pipelines, graph analytics, linear algebra, and finite element methods. Skeleton approaches have also been successfully applied to provide high-performance computing from functional specifications, or via template meta-programming. Domain-specific language (DSL) extensions will be used to better support the semantics of memory accesses, together with specific code optimizations to match the requirements of the applications and the characteristics of the memories. Such extensions will be also used to specify hardware/software partitioning and automatically generate the corresponding code based on existing parallel programming models.

HLS tools are thus increasingly integrating a wide range of source-to-source transformations to integrate the domain-specific information provided by the application programmers, to make explicit transformations and to enable the efficient use of alternative memory technologies. These transformations will focus on the optimization of resources and the memory accesses, but the approach can be easily extended towards other design goals. In such context, co-design methods will be more and more important to simplify the integration of such components with transparent hardware services [2,15].

5. Future Research Lines for High-Level Synthesis

5.1. Emerging Technologies

Emerging technologies will play a key role in the design of efficient accelerators. First, fine-grained power management will be increasingly exploited by combining integrated voltage regulators for implementing dynamic voltage-frequency scaling [35,14], and dual-rail SRAMs to reduce the static power of memory elements [23]. Then, memory design is a key element in the creation of efficient hardware components because these components require elaborating a huge amount of data often beyond the capabilities of current systems. This easily leads to critical issues in terms of performance and energy, while the speed of DRAM memories is not increasing with the cumulated processor frequency (especially in case of parallel or heterogeneous computing), leading to the so-called *memory wall* where the overall performance becomes limited by the latency of the memory accesses. In addition, extreme-data applications require frequent memory accesses to off-chip memory, whose I/O circuitry and refresh activities is responsible for up to 30% of the system energy consumption (*power wall*). Such issues can be mitigated only with a fundamental redesign of the memory architecture with the inclusion of emerging memory technologies. However, novel current ideas include 3D packing and hybrid on-chip memory architectures [17], but they are usually limited in terms of capacity. For instance, HMC and HBM can store data only up to 4GB, whereas DDR4 is up to 128GB (*capacity wall*). Non-volatile technologies (NVM), which require no refresh and therefore reduce the power wall, can be combined with DRAM to reach capacities up to TB, potentially breaking also the capacity wall. The recent emergence of a rich alternatives for memory technologies have spawn a lot of research in memory architectures, but there is no clear winning yet and systems will combine, instead, different more and more technologies including resistance-based memories like Phase Change Memory (PCM), 3D XPoint (3DX), Spin Transfer Torque (STT), and Resistive Random Access Memory (RRAM). However, these memory technologies have different characteristics in terms of read/write latency and energy, leakage power, write endurance, resiliency [17], opening fundamental programmability issues to understand how to balance such conflicting characteristics and match them to different application requirements.

5.2. Domain-Specific Languages for Better Optimizations

Since heterogeneous systems are expected to become more and more complex, the efficient composition of different components will be critical. Domain specific languages (DSLs) will be used by application engineers to specify the algorithmic core of the computation. They will be increasingly used to provide rich information to the compiler and lower-level tools about the high-level semantics of the algorithms. However, DSLs are usually hard to be accepted by software programmers and they may create integration issues. So, they will be necessarily embedded into existing languages, e.g., via template metaprogramming in C++/SYCL to easy programming. Memory-specific transformations will be then enabled in the front-end phase. Such optimizations will include both loop transformations to improve the memory accesses with respect to the characteristics of the available technologies and code transformations to enable more optimizations during hardware generation. Rich information about the memory access patterns of the algorithm allows the compiler to automatically optimize the loop structure, thereby modifying the memory access pattern while preserving semantics. Annotations can be then provided for a variety of functional and non-functional requirements (e.g., power consumption, security, reliability, etc.) and for user-driven hardware-software partitioning. Additional annotations can be also used to specify where to perform selective profiling and debugging, reducing the cost of design and verification of such components. Finally, heterogeneous programming models (e.g., OpenCL or SYCL) will be used to express functionality and to enforce portability.

5.3. Hardware Security

Heterogeneous architectures are increasingly leveraging hardware accelerators and such components are thus becoming an easy target for malicious attacks [28]. In fact, most of the current protection mechanisms are manually implemented [7] or introduce large overheads [10], and this is clearly not efficient nor scalable and hardware vulnerabilities can be exploited to perform software-based attacks. Even if it is not possible to perform *code injection*, malicious configuration parameters or memory values can be used to exploit *design errors*. For example, attackers may exploit don't vulnerabilities in the accelerator's FSM to alter the accelerator's behavior [19]. *Physical attacks* can exploit the weaknesses of a given hardware implementation for *information leakage*. Side-channel attacks have proven to be a powerful mechanism to extract secret data from both embedded devices and high-end cloud servers. Accelerators can help avoiding side-channel attacks, for instance ensuring constant execution time and thus making timing attacks infeasible. However, accelerators must be protected from a variety of other attacks, including fault-based and side-channels ones [33]. If not adequately protected, a circuit completely separated by the rest of the processor can be localized more easily, becoming the target of more precise power measurements. Another critical issue is *IP theft*: an efficient and secure accelerator is a complex task whose outcome should be protected from reverse engineering and unauthorized copy, which can create billions of dollars of economical damages for the companies [8].

6. Conclusions

This paper describes the gap that has been created between software and hardware developers due to increasing use of heterogeneous computing platforms. This can be partially solved by the use of high-level synthesis, which allows the automatic creation of hardware components starting from high-level descriptions. However, design and verification challenges are still open for these systems, while new challenges in terms of hardware security must be investigated.

Acknowledgment

The author would like to thank Luca P. Carloni (Columbia University), Fabrizio Ferrandi (Politecnico di Milano), Francesco Regazzoni (USI Lugano), Jeronimo Castrillon (TU Dresden), Ramesh Karri (NYU), Siddharth Garg (NYU), and all his former colleagues for the useful discussions on these topics.

References

[1] K. Campbell, L. He, L. Yang, S. Gurumani, K. Rupnow, and D. Chen. Debugging and verifying SoC designs through effective cross-layer hardware-software co-simulation. In *Proceedings of the Design Automation Conference (DAC)*, June 2016.

[2] L. P. Carloni. The case for embedded scalable platforms. In *Proceedings of the Design Automation Conference (DAC)*, pages 17:1–17:6, June 2016.

[3] V. G. Castellana, M. Minutoli, A. Morari, A. Tumeo, M. Lattuada, and F. Ferrandi. High level synthesis of RDF queries for graph analytics. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 323–330, 2015.

[4] E. Cota, P. Mantovani, G. Di Guglielmo, and L. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2015.

[5] P. Fezzardi, M. Castellana, and F. Ferrandi. Trace-based automated logical debugging for high-level synthesis generated circuits. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 251–258, 2015.

[6] P. Fezzardi, M. Lattuada, and F. Ferrandi. Using efficient path profiling to optimize memory consumption of on-chip debugging for high-level synthesis. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s), Sept. 2017.

[7] A. Gornik, A. Moradi, J. Oehm, and C. Paar. A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(8):1308–1319, Aug. 2015.

[8] U. Guin et al. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proceedings of the IEEE*, 102(8):1207–1228, Aug. 2014.

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

[10] W. Hu et al. Theoretical fundamentals of gate level information flow tracking. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 30(8):1128–1140, Aug. 2011.

[11] M. Lattuada and F. Ferrandi. Code transformations based on speculative SDC scheduling. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 71–77, 2015.

[12] H.-Y. Liu, M. Petracca, and L. P. Carloni. Compositional system-level design exploration with planning of high-level synthesis. In *Proc. of DATE*, pages 641–646, Mar. 2012.

[13] P. Mantovani, E. G. Cota, C. Pilato, G. D. Guglielmo, and L. P. Carloni. Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 1–10, 2016.

[14] P. Mantovani et al. An FPGA-based Infrastructure for Fine-grained DVFS Analysis in High-performance Embedded Systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2016.

[15] P. Mantovani, G. D. Guglielmo, and L. P. Carloni. High-level synthesis of accelerators in embedded scalable platforms. In *Proceedings of the Asian and South Pacific Design Automation Conference (AS-PDAC)*, pages 204–211, Jan. 2016.

[16] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano. DeSpErate++: An enhanced design space exploration framework using predictive simulation scheduling. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(2):293–306, 2015.

[17] S. Mittal, J. S. Vetter, and D. Li. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1524–1537, 2015.

[18] J. S. Monson and B. L. Hutching. New approaches for in-system debug of behaviorally-synthesized FPGA circuits. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, Sept. 2014.

[19] A. Nahiyan et al. AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2016.

[20] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. D. Brown, F. Ferrandi, J. H. Anderson, and K. Bertels. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 35(10):1591–1604, Oct. 2016.

[21] D. Novillo. Design and implementation of tree ssa. In *GCC Developers' Summit*, pages 119–130, 2004.

[22] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni. COSMOS: Coordination of high-level synthesis and memory optimization for hardware accelerators. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–22, Sept. 2017.

[23] C. Pilato and L. P. Carloni. DarkMem: Fine-grained power management of local memories for accelerators in embedded systems. In *Proceedings of the Asian and South Pacific Design Automation Conference (ASPDAC)*, 2018.

[24] C. Pilato, V. G. Castellana, S. Lovergine, and F. Ferrandi. A runtime adaptive controller for supporting hardware components with variable latency. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 153–160, June 2011.

[25] C. Pilato, A. Cazzaniga, G. Durelli, A. Otero, D. Sciuto, and M. D. Santambrogio. On the automatic integration of hardware accelerators into FPGA-based embedded systems. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 607–610, 2012.

[26] C. Pilato and F. Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Sept. 2013.

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

[28] C. Pilato, S. Garg, K. Wu, R. Karri, and F. Regazzoni. Securing hardware accelerators: a new challenge for high-level synthesis. *IEEE Embedded Systems Letters*, pages 1–4, 2017.

[29] C. Pilato, P. Mantovani, G. D. Guglielmo, and L. P. Carloni. System-level optimization of accelerator local memory for heterogeneous systems-on-chip. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 36(3):435–448, Mar. 2017.

[30] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto. Improving evolutionary exploration to area-time optimization of FPGA designs. *Journal of Systems Architecture - Embedded Systems Design*, 54(11):1046–1057, 2008.

[31] 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 the International Conference on Computing Frontiers (CF)*, pages 406–411, May 2016.

[32] V. Rana, I. Beretta, F. Bruschi, A. A. Nacci, D. Atienza, and D. Sciuto. Efficient hardware design of iterative stencil loops. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 35(12):2018–2031, 2016.

[33] F. Regazzoni et al. Power Attacks Resistance of Cryptographic S-Boxes with Added Error Detection Circuits. In *Proc. of DFT*, pages 508–516, 2007.

[34] L. Stok. Data path synthesis. *Integration, the VLSI Journal*, 18(1):1–71, Dec. 1994.

[35] N. Sturcken, E. J. O'Sullivan, N. Wang, P. Herget, B. Webb, L. Romankiw, M. Petracca, R. Davies, R. Fontana, G. M. Decad, I. Kymissis, A. V. Peterchev, L. Carloni, W. Gallagher, and K. Shepard. A 2.5D integrated voltage regulator using coupled magnetic core inductors on silicon interposer delivering 10.8A$/mm^2$. In *ISSCC Digest of Technical Papers*, pages 400–402, Feb. 2011.

[36] Y. Wang, P. Li, and J. Cong. Theory and algorithm for generalized memory partitioning in high-level synthesis. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 199–208, Feb. 2014.

[37] Y. Wang, P. Li, and J. Cong. Theory and algorithm for generalized memory partitioning in high-level synthesis. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 199–208, 2014.

[38] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, Feb. 2017.

[39] J. Zhu and D. D. Gajski. A unified formal model of ISA and FSMD. In *Proc. of CODES*, pages 121–125, 1999.