Compiler-Driven FPGA Virtualization with SYNERGY
Extended Abstract

Joshua Landgraf[1] Tiffany Yang[1] Will Lin[1] Christopher J. Rossbach[123] Eric Schkufza[4]
[1]The University of Texas at Austin [2]VMware Research [3]Katana Graph [4]Amazon

1. Motivation

Field-Programmable Gate Arrays (FPGAs) combine the functional efficiency of hardware with the programmability of software. They can exceed general-purpose CPU performance by orders of magnitude [19, 4] and offer lower cost and time to market than ASICs. In data centers, FPGAs give infrastructure providers a means to support diverse hardware needs with a single technology [4].

Virtualization is fundamental to the success of data centers. It decouples applications from dependences on hardware and enables economies of scale through consolidation. However, a standard technique for virtualizing FPGAs has yet to emerge. There are no widely agreed upon methods for supporting key primitives such as *workload migration* (suspending and resuming the execution of a hardware program, or relocating it from one FPGA to another mid-execution) or *multi-tenancy* (multiplexing multiple hardware programs on a single FPGA). Our goal is to remove these limitations, enabling FPGAs to realize their potential as a mainstream accelerator technology.

2. Limitations of the State of the Art

FPGA virtualization is difficult because FPGAs lack a well-defined interposable *application binary interface* (ABI) and state capture mechanisms. The state of an FPGA program is distributed throughout its reprogrammable fabric in a *program-dependent* and *hardware-dependent* fashion that is inaccessible to the OS. The only way to suspend and resume execution is to inefficiently access the *entire* register state of the device. Without knowing how programs are compiled for an FPGA, there is no way to share the FPGA with other programs or relocate programs between FPGAs mid-execution.

A significant body of work has addressed the problems of sharing FPGA fabric [5, 3, 8, 24, 14, 13], spatial multiplexing [9, 22, 23, 6], context switch [15, 20], memory virtualization [7, 1, 25, 17], relocation [11], preemption [16], and interleaved hardware-software task execution [2, 22, 23, 10]. However, most of these approaches use hardware-based virtualization solutions, which partition the device into isolated regions exposed as a set of smaller fabrics. This approach enables sharing, but cannot support features like workload migration. Moreover, it suffers from fabric fragmentation and underutilization.

One current state-of-the-art solution is AmorphOS [12], an FPGA runtime which supports cross-program protection and cross-platform compatibility at very high degrees of multi-tenancy. AmorphOS allows hardware programs to adapt to changes in load and availability by dynamically scaling the amount of FPGA fabric they consume. AmorphOS can transparently change mappings between user logic and FPGA fabric to increase utilization and avoid fragmentation. However, AmorphOS delegates the problem of efficient context switch to the programmer by exposing an interface to manage application state. AmorphOS leaves over-subscription and support for multiple FPGAs completely unsolved.

Another state-of-the-art solution is Cascade [21], the first JIT compiler for Verilog. Cascade executes hardware programs in a combination of software simulation and FPGA fabric to present the illusion of zero-latency hardware compilation. Cascade also applies transformations to the user's program to produce code that can trap into the Cascade runtime at the end of the logical clock tick. These traps allow the user to run programs which contain unsynthesizable statements (code which would otherwise be restricted to software simulation) in a way that is consistent with the scheduling semantics of Verilog, even during hardware execution. However, Cascade does not support virtualization primitives such as context switch and migration, and lacks the fine granularity of control needed to execute unsynthesizable statements with side-effects which must be resolved mid-clock-cycle.

3. Key Insights

We argue that the right place to support FPGA virtualization is in a combined compiler/runtime environment. SYNERGY combines a *just-in-time* (JIT) compiler for Verilog, canonical interfaces to OS-managed resources, and an OS-level protection layer to abstract and isolate shared resources. The key insight behind SYNERGY is that a compiler can re-write Verilog code to compensate for the missing FPGA ABI and explicitly expose application-level state to the OS. The core technique used by SYNERGY is a static analysis to transform the user's code into a distributed-system-like *intermediate representation* (IR) consisting of monadic sub-programs which can be moved back and forth mid-execution between a software interpreter and native FPGA execution. This is possible because the transformations produce code that can trap to the software runtime at arbitrary points in time, even mid-clock-cycle, according to the semantics of the original program.

4. Main Artifacts

SYNERGY extends the Cascade [21] JIT compiler and composes it with the AmorphOS [12] FPGA OS. We measure SYNERGY in real-world contexts that represent the heterogeneity of the data center. We show the ability to suspend and resume programs running on a cluster of Altera SoCs and Xilinx FPGAs running on Amazon's F1 cloud instances, to transition applications between the two, and to temporally and spatially multiplex both devices efficiently with strong OS-level isolation guarantees. This is done without exposing the architectural differences between the platforms, or requiring extensions to the Verilog language or modifications to the user's program.

SYNERGY's first contribution is a set of compiler transformations to produce code that can be interrupted at *sub-clock-tick granularity* according to the semantics of the original program. Compared to Cascade, this allows SYNERGY to support a large new class of *unsynthesizable* Verilog, even while executing in hardware. Traditional Verilog uses unsynthesizable language constructs for testing and debugging in a simulator, but SYNERGY can also use them to expose interfaces to OS-managed resources and to start, stop, and save the state of a program at any point in its execution. This allows SYNERGY to perform context switch and workload migration without hardware support or modifications to Verilog.

SYNERGY's second contribution is a new technique for FPGA multi-tenancy. SYNERGY introduces a hypervisor layer into the compiler's runtime which can combine the sub-program representations from multiple applications by multiple instances of the compiler) into a single hardware program whose implementation is kept hidden from those instances. This module is responsible for interleaving asynchronous data and control requests between each of those instances and the FPGA. In contrast to hardware-based approaches, manipulating each instance's state is straightforward, as the hypervisor has access to every instance's source and knows how it is mapped onto the device.

SYNERGY's final contribution is a compiler backend targeting an OS-level protection layer for process isolation, fair scheduling, and cross-platform compatibility. Recent OS-FPGA proposals harden vendor *shells* and export interfaces for an application to assist the OS with state capture for context switch [12, 18]. A major obstacle to using these systems is the requirement that the developer implement state capture and/or quiescence interfaces. SYNERGY satisfies this requirement automatically by using static analysis to identify the set of variables that comprise a program's state and emitting code to interact with those interfaces. For applications which natively support these interfaces, SYNERGY can use that support to dramatically reduce overhead for context switch and migration.

5. Key Results and Contributions

We evaluated SYNERGY using a combination of Altera DE10 SoCs and Amazon F1 cloud instances on benchmarks representing a mixture of batch and streaming computations. Our experiments show that SYNERGY improves upon Cascade's performance. Despite targeting a $5\times$ higher frequency on F1, implementing more complex program transformations, and accounting for device frequency overheads, SYNERGY still achieves a virtual clock frequency [21] within $3-4\times$ of native unvirtualized performance and maintains a reasonable fabric cost. Moreover, we note that these figures do not represent a lower-bound, and we expect further engineering to reduce them considerably. Our evaluation demonstrates SYNERGY's support for:

- **Workload Migration.** SYNERGY executes a Verilog program on one FPGA architecture, suspends it, saves its state, and resumes its execution and state on a different FPGA architecture at a later time. SYNERGY also live migrates a program between two FPGAs.
- **Multitenancy.** SYNERGY co-schedules multiple programs on the same FPGA without contention. It also temporally multiplexes off-device IO as programs contend for it.
- **Automated Quiescence Management.** SYNERGY uses programer annotations to reduce captured state by up to 99%, freeing up fabric for use by other applications.

6. Why ASPLOS?

This paper touches on all three facets of the ASPLOS charter. It explores techniques for virtualizing FPGAs (*AS*) using techniques that rely heavily on compiler-driven transformations and language-level abstractions (*PL*) as well as *OS*-level protection, resource-management, and interfaces.

7. Citation

SYNERGY addressed a long-standing challenge in reconfigurable computing: how to virtualize FPGAs with full support for critical features like suspend/resume, workload migration, context switch, and multi-tenancy. In contrast to a multitude of previous solutions, which focused on partitioning hardware, introducing canonical interfaces, and using hardware-supported state capture primitives, SYNERGY recast the problem as a software challenge. Focusing on the compiler and runtime enabled SYNERGY to use code transformations to automatically introduce primitives and interfaces whose absence made FPGA virtualization difficult. SYNERGY transformed Verilog programs so they could yield control to software at *sub-clock-tick* granularity according to the semantics of the original program, providing efficient support for core virtualization primitives: suspend and resume, program migration, and spatial/temporal multiplexing, on hardware which was available *in 2020*.

References

[1] Michael Adler, Kermin E. Fleming, Angshuman Parashar, Michael Pellauer, and Joel Emer. Leap Scratchpads: Automatic Memory and Cache Management for Reconfigurable Logic. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 25–28, New York, NY, USA, 2011. ACM.

[2] Gordon J. Brebner. A Virtual Hardware Operating System for the Xilinx XC6200. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, FPL '96, pages 327–336, London, UK, UK, 1996. Springer-Verlag.

[3] Stuart Byma, J. Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack. In *Proceedings of the 2014 IEEE 22Nd International Symposium on Field-Programmable Custom Computing Machines*, FCCM '14, pages 109–116, Washington, DC, USA, 2014. IEEE Computer Society.

[4] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. A Cloud-Scale Acceleration Architecture. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[5] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling FPGAs in the Cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 3:1–3:10, New York, NY, USA, 2014. ACM.

[6] Liang Chen, Thomas Marconi, and Tulika Mitra. Online Scheduling for Multi-core Shared Reconfigurable Fabric. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 582–585, San Jose, CA, USA, 2012. EDA Consortium.

[7] Eric S. Chung, James C. Hoe, and Ken Mai. CoRAM: An In-fabric Memory Architecture for FPGA-based Computing. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 97–106, New York, NY, USA, 2011. ACM.

[8] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized FPGA Accelerators for Efficient Cloud Computing. In *Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, CLOUDCOM '15, pages 430–435, Washington, DC, USA, 2015. IEEE Computer Society.

[9] W. Fu and K. Compton. Scheduling intervals for reconfigurable computing. In *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pages 87–96, April 2008.

[10] Ivan Gonzalez, Sergio Lopez-Buedo, Gustavo Sutter, Diego Sanchez-Roman, Francisco J. Gomez-Arribas, and Javier Aracil. Virtualization of Reconfigurable Coprocessors in HPRC Systems with Multicore Architecture. *J. Syst. Archit.*, 58(6-7):247–256, June 2012.

[11] H. Kalte and M. Porrmann. Context saving and restoring for multitasking in reconfigurable systems. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 223–228, Aug 2005.

[12] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 107–127, 2018.

[13] O. Knodel, P. Lehmann, and R. G. Spallek. RC3E: Reconfigurable Accelerators in Data Centres and Their Provision by Adapted Service Models. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 19–26, June 2016.

[14] Oliver Knodel and Rainer G. Spallek. RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment. *CoRR*, abs/1508.06843, 2015.

[15] Trong-Yen Lee, Che-Cheng Hu, Li-Wen Lai, and Chia-Chun Tsai. Hardware context-switch methodology for dynamically partially reconfigurable systems. *J. Inf. Sci. Eng.*, 26:1289–1305, 2010.

[16] L. Levinson, R. Manner, M. Sessler, and H. Simmler. Preemptive multitasking on fpgas. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, pages 301–302, 2000.

[17] Enno Lübbers and Marco Platzner. ReconOS: Multithreaded Programming for Reconfigurable Computers. *ACM Trans. Embed. Comput. Syst.*, 9(1):8:1–8:33, October 2009.

[18] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Xiaohe Cheng, Yanqiang Liu, Abel Mulugeta Eneyew, Zhengwei Qi, and Baris Kasikci. A hypervisor for shared-memory fpga platforms. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[19] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Jim Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *41st Annual International Symposium on Computer Architecture (ISCA)*, June 2014.

[20] Kyle Rupnow, Wenyin Fu, and Katherine Compton. Block, drop or roll(back): Alternative preemption methods for RH multi-tasking. In *FCCM 2009, 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, California, USA, 5-7 April 2009, Proceedings*, pages 63–70, 2009.

[21] Eric Schkufza, Michael Wei, and Christopher J. Rossbach. Just-in-time compilation for verilog: A new technique for improving the FPGA programming experience. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, pages 271–286, 2019.

[22] C. Steiger, H. Walder, and M. Platzner. Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks. *IEEE Transactions on Computers*, 53(11):1393–1407, Nov 2004.

[23] G. Wassi, Mohamed El Amine Benkhelifa, G. Lawday, F. Verdier, and S. Garcia. Multi-shape tasks scheduling for online multitasking on FPGAs. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, pages 1–7, May 2014.

[24] Jagath Weerasinghe, François Abel, Christoph Hagleitner, and Andreas Herkersdorf. Enabling FPGAs in Hyperscale Data Centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), Beijing, China, August 10-14, 2015*, pages 1078–1086, 2015.

[25] Felix Winterstein, Kermin Fleming, Hsin-Jung Yang, Samuel Bayliss, and George Constantinides. Matchup: Memory abstractions for heap manipulating programs. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 136–145, New York, NY, USA, 2015. ACM.