Overlay Architectures For FPGA Resource Virtualization

Théotime Bollengier, Mohamad Najem, Jean-Christophe Le Lann, Loïc Lagadec

Overlay Architectures For FPGA Resource Virtualization

Théotime BOLLENGIER[‡], Mohamad NAJEM[*], Jean-Christophe LE LANN[*], and Loïc LAGADEC[*]

[‡]b<>com, Brest, France
E-mail : *theotime.bollengier@b-com.com*
[*]ENSTA Bretagne, UMR 6285, Lab-STICC, Brest, France
E-mail : {*mohamad.najem, jean-christophe.le_lann, loic.lagadec*}*@ensta-bretagne.org*

Abstract

Overlays are reconfigurable architectures synthesized on commercial of the shelf (COTS) FPGAs. Overlays brings some advantages such as portability, resource abstraction, fast configuration time, and can exhibit features independent from the host FPGA. This work presents the design of a fine-grained overlay, and the implementation of novel features easing the management of such architectures in a datacenter infrastructure.

1. Introduction

Overlays are reconfigurable architectures implemented on top of FPGAs. They are regular designs described using structural HDL, but have reconfigurable capabilities. They may be considered as "softcore FPGA IPs". Thus, an overlay can be seen from two sides: *i*) the *functional architecture* is the top view, it is the set of reconfigurable elements available to the applications targeting the overlay; *ii*) the *implementation* is the bottom view, it is the way the functional architecture is implemented and synthesized (as a regular IP) on the host FPGA . The functional architecture of an overlay is independent from its implementation and from its host FPGA, and can be one of different granularities: fine-grained overlays generally being composed of LUTs operating at the bit level, while function units of coarser architectures use mathematical operators on words. Overlays are similar to the Java Virtual Machine which enable the exact same bytecode to be executed on different processors (with different instruction sets) for which the JVM has been compiled for.

Thus, overlays come with three advantages:

- they can be used to homogenize different COTS FPGAs by implementing the same functional architecture on different hosts, bringing binary compatibility between heterogeneous FPGAs;
- the overlay fuctional architecture may be coarser, simpler and more abstract than the one of its host;
- the overlay designers can add features to the overlay implementation that may not be present on the host FPGA, such as dynamic context saving/restoring or configuration pre-loading.

However, compared with a bare metal use of FPGA resources, virtualization with overlays has a cost in terms of resource usage and operating frequency. Therefore, overlays are used in applications for which their advantages justify virtualization cost. Sekanina used coarse grained overlays [6] for evolvable hardware research, benefiting from shorter synthesis and configuration time. Lysecky et al. [5] designed a fine grained overlay with extra routing resources to ease the task of their Just-In-Time synthesizer. To lower compilation time, Coole and Stitt introduced intermediate fabrics [2], which are application specific coarse grained overlays. Dirk Koch et al. [4] integrated a fine-grained overlay in the datapath of a MIPS processor to get a portable custom instruction set softcore processor. Brant and Lemieux addressed the area overhead problem by doing target specific optimizations on the implementation of their fine-grained overlay named ZUMA [1], getting down to 40 physical LUT to implement a virtual LUT. Jain et al. [3] also decreased their overlay size down to 70 % and reached frequencies excessing 300 MHz by efficiently using the host DSP blocks in the coarse grained virtual functional units.

Our work aims to virtualize FPGA resources in the Cloud, available to the client as reconfigurable hardware accelerators. Components of a datacenter infrastructure are gradually updated and replaced over time, which results in FPGAs with different characteristics (and from different vendors) being used at the same time. In this context, overlays are relevant as they allow homogenizing these resources. Clients no longer have to be aware of the available FPGA characteristics, and their bitstreams are

no longer tied to a limited set of FPGAs from the infrastructure.

Moreover, overlay implementations can be tweaked to ease the management of such architectures in datacenters. In this work, we add novel features to allow hardware task preemptive scheduling and live migration.

2. Overlay design and use: functional architecture

In this work, we designed a LUT based overlay. Even though fined-grained overlays exhibit an important area and timing overhead, they are more general purpose than coarser architectures which are limited to data-flow applications due to the lack of control structures and are tied to application domains by the choice of their operators.

Our proposed overlay is an island-style architecture composed of Configurable Logic Blocks (CLBs) surrounded by routing channels. CLBs are clusters of LUT-FlipFlop pairs called Basic Logic Elements (BLEs) in which the flip-flop can be bypassed. Routing channels are composed of unidirectional routing tracks. These routing tracks are connected to neighboring CLBs and routing tracks from adjacent channels through Switch Blocks (SB). Each element of this architecture is conditioned by a multiplexer, which is configured on its select signals by some bits from the overlay configuration register.

To ease architectural exploration, the overlay HDL description is automatically generated from a set of parameters, such as the size of the CLB matrix, the number of BLE per CLB, the number of input per LUT and per CLB, or the number or routing tracks per routing channel. The generated HDL code is portable and can be simulated or implemented on any COTS FPGA that fit the design.

Synthesizing an application to the overlay architecture is done in different steps. First a RTL synthesizer transforms the application description into a netlist composed of latches and arbitrary logic gates. Then this netlist is transformed and optimized so that none of its logic gate has more inputs than the number of input per LUT of the overlay. Next, the netlist is placed and routed on the overlay. Finally the virtual bitstream is generated by identifying the overlay elements' state for the placed and routed netlist.

3. Implementation features

We implemented the virtual flip-flops located in BLEs as regular registers, clocked from the fixed underlay system clock, and thus benefit from the physical clock distribution tree. However, the D $\rightarrow$ Q transfer is enabled only when the virtual clock signal (which is the only clock seen by virtual applications) is active. This virtual clock is a regular FPGA signal generated from a counter. It is therefore possible to dynamically change the overlay virtual operating frequency, to pause the execution or to run it for a given number of virtual clock cycles.

Additionally, we paired each virtual flip-flop with a *snapshot register*. Two global control signals permit to save the value from a virtual flip-flop to its snapshot register, or to restore the value from the snapshot register to the virtual flip-flop. Snapshot registers are connected in a daisy chain so that the sequential execution state of an application can be serialy inserted or extracted from the overlay.

The dynamic and fine control of the virtual clock plus the ability to dynamically save or restore a snapshot of the execution state allow performing preemptive scheduling of different virtual applications on the same overlay, or to migrate a running application from one overlay to another without altering the application's execution state.

4. Conclusion

Overlay architectures are relevant in a Cloud context as they homogenize and abstract FPGA resources. Adding special features to the overlay implementations such as dynamic clock control and a state snapshot mechanism allows performing hardware task preemptive scheduling and live migration. This, in turn, allows using FPGAs as multi-tenant devices and to apply dynamic infrastructure optimization policies such as load balancing.

References

[1] A. Brant and G. G. F. Lemieux. ZUMA: an open FPGA overlay architecture. In *2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012, 29 April - 1 May 2012, Toronto, Ontario, Canada*, pages 93–96, 2012.

[2] J. Coole and G. Stitt. Intermediate fabrics: virtual architectures for circuit portability and fast placement and routing. In *Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2010, part of ESWeek '10 Sixth Embedded Systems Week, Scottsdale, AZ, USA, October 24-28, 2010*, pages 13–22, 2010.

[3] A. K. Jain, D. L. Maskell, and S. A. Fahmy. Throughput oriented fpga overlays using dsp blocks. 2016.

[4] D. Koch, C. Beckhoff, and G. G. F. Lemieux. An efficient FPGA overlay for portable custom instruction set extensions. In *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*, pages 1–8, 2013.

[5] R. L. Lysecky, K. Miller, F. Vahid, and K. A. Vissers. Firmcore virtual FPGA for just-in-time FPGA compilation. In *Proceedings of the ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays, FPGA 2005, Monterey, California, USA, February 20-22, 2005*, page 271, 2005.

[6] L. Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In *Evolvable Systems: From Biology to Hardware, 5th International Conference, ICES 2003, Trondheim, Norway, March 17-20, 2003, Proceedings*, pages 186–197, 2003.