Migration of long-running Tasks between Reconfigurable Resources using Virtualization

Oliver Knodel, Paul R. Genssler and Rainer G. Spallek
Department of Computer Science
Technische Universität Dresden
Dresden, Germany
{firstname.lastname}@tu-dresden.de

ABSTRACT

Computing performance and scalability are the essential basics in modern data centres. Field Programmable Gate Arrays (FPGAs) provide a promising opportunity to improve performance, security and energy efficiency. Especially background acceleration of computationally complex and long-running tasks is an important field of application. A flexible use of reconfigurable devices within a cloud context requires an abstraction of the actual hardware through virtualization.

In this paper we present an approach inspired by paravirtualized machines for the integration of reconfigurable hardware into cloud services. Using partial reconfiguration our hardware and software framework virtualizes a single physical FPGA to enable multiple independent user designs. Essential components are the management of those virtual user-defined accelerators (vFPGA) and their migration between physical FPGAs to achieve higher system-wide utilization. The migration requires saving and restoring the internal state or context of the vFPGA. We demonstrate the application possibilities and the resource trade-off of our approach by transferring a running design from one physical FPGA to another. Moreover, we present future perspectives for the use of FPGAs in cloud-based environments.

Keywords

Field Programmable Gate Arrays; Virtualization; Context Migration; Partial Reconfiguration; Cloud Computing.

1. MOTIVATION

Cloud computing is based on the idea of computing as a utility. The user gains access to a shared pool of computing resources or services that can rapidly be allocated and released "with minimal management effort or service provider interaction" [1]. An essential advantage, compared to traditional models in which the user has access to a fixed number of computing resources, is the elasticity of them in a cloud. Even in peak load situations there are always enough resources available, hence the user does not have to concern himself and only pays for what he actually utilizes [2].

But with the theoretically unlimited number of resources arises a major problem for data centres housing clouds: their enormous energy consumption. One possibility to enhance computation performance by simultaneously lowering energy consumption is the use of heterogeneous systems, offloading computationally intensive applications to special hardware coprocessors or dedicated accelerators. Especially reconfigurable hardware such as Field Programmable Gate Arrays (FPGAs) provide an opportunity to improve computing performance [3], security [4] and energy efficiency [5].

A profound and flexible integration of FPGAs into scalable data centre infrastructures which satisfy the cloud characteristics, is a task of growing importance in the field of energy efficient cloud computing. In order to achieve such an integration, the virtualization of FPGA resources is necessary. The provisioning of virtual FPGAs (vFPGAs) makes the reconfigurable resource available to customers of the data centre provider. Therefore we will call the service providers hereafter users. They can accelerate specific services, reduce energy consumption and service costs.

The virtualization of reconfigurable hardware devices is a recurring challenge. Decades ago the virtualization of FPGA devices started due to the limitation of logical resources [6]. Nowadays the FPGAs grow in size and full utilization of the devices cannot always be achieved in practice. One possibility to increase utilization is our virtualization approach with flexible design sizes and multiple hardware designs on the same physical FPGA. A challenge of this approach are the unsteady load situations of an elastic cloud, which processes both short tasks and long-running background acceleration services. Intelligent load balancing requires the migration of one vFPGA with a long-running job from one physical FPGA to another. Thus, fragmentation can be reduced and as a result also energy consumption by switching off a node.

In this contribution we introduce our virtualization concept for FPGAs which is inspired by traditional virtual machines (VMs). One physical FPGA can consist of multiple vFPGAs belonging to different services with different runtimes. Each of the vFPGAs can be configured using partial reconfiguration [7] and is equipped with a state management which offers the possibility to pause the vFPGA and to capture the internal register and memory context using the internal configuration access port (ICAP) [7]. In the next step the context is restored on a different vFPGA using modifications on the bitstream level. This technique allows context readback and restoration without additional modifications inside the vFPGA's design on the source code or the netlist level.

The structure of the paper is as follows. Section 2 introduces similar concepts and related research in the field of reconfigurable hardware in cloud architectures. In Section 3 we give an overview on virtual machines and in detail on our FPGA-related virtualization concept. Our computing

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2016), Hong Kong, July 25-27, 2016.

framework, which supports the concept of our FPGA virtualization, is presented in Section 4 followed by device utilization and performance results in Section 5. Section 6 concludes and gives an outlook.

2. RELATED WORK

The provisoning of reconfigurable hardware in data centres and cloud environments has gained more and more importance in the last years. Initially used mainly on the network infrastructure level, FPGAs are now also employed on the application level of data centres. Typical use cases are background acceleration of specific functions with static hardware designs. The FPGA's special feature to reconfigure hardware at runtime is still used rather rarely. Examples are the anonymization of user requests [8] and increasing security [4] by outsourcing critical parts to attack-safe hardware implementations. In most cases the FPGAs are not directly useable or configurable by the user, because the devices are hidden deeply in the data centre, due to a missing provisioning or virtualization [9].

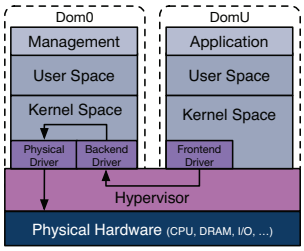
The term *virtualization* is used for a wide range of concepts. An example for abstractions on the hardware description level is VirtualRC [10], which uses a uniform hardware/software interface to realize communication on different FPGA platforms. BORPH [11] takes a similar approach, employing a homogeneous UNIX interface for hardware and software. The FPGA paravirtualization pvFPGA [12], which uses an integration of FPGA device drivers into a paravirtualized Xen virtual machine, presents a more sophisticated concept. A framework for the integration of reconfigurable hardware into cloud architecture is shown by Chen et. al [13] and Byma et. al [14]. The framework of Byma et. al allows user-specific acceleration cores on the reconfigurable hardware devices, which are accessible via an Ethernet connection.

Approaches more closely related to the *context-save-and-restore* mechanism required by our migration can be found in the field of bitstream readback, manipulation and hardware preemption. In [15], Jozwick et al. present a system to capture and restore the context of a hardware task on a Xilinx Virtex-4 FPGA with additional logic inside their reconfigurable regions. Morales-Villanueva et al. [16] use the Virtex-5 to relocate designs with their states. In ReconOS [17] hardware task preemption is used to capture and restore the states of all flip-flops and block RAMs on a Virtex-6 to allow multitasking with hardware threads. Approaches using scan-chains to extract or insert states employing additional hardware are shown in [18, 19]. Both solutions are fast, but they require modifications of the designs on the source or the netlist level.

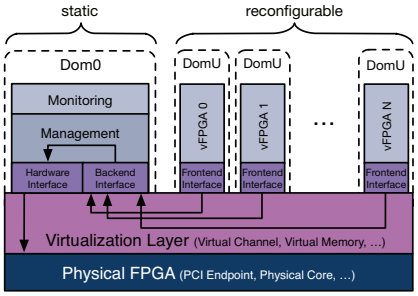
An approach which places multiple user designs on a single FPGA is introduced by Fahmy et. al [20], using tightly attached FPGAs to offload computationally intensive tasks. The FPGAs are partially reconfigurable and can hold up to four individual user designs. A cloud integration with network-attached FPGAs and also multiple user designs on one FPGA are introduced by Weerasinghe et. al [9].

3. VIRTUALIZATION CONCEPT

As mentioned before, the integration of FPGAs into a cloud context requires a profound virtualization concept. Multiple concurrent vFPGAs allocated by different users with different acceleration design will be deployed together on one physical FPGA. In the following, we first give some background information on traditional operating system VMs in Section 3.1. Based on that, we introduce our virtualization concept for reconfigurable hardware in Section 3.2.



(a) Traditional paravirtualized System.



(b) Concept for FPGA virtualization.

Figure 1: Traditional paravirtualization compared to FPGA virtualization with multiple vFPGAs with different sizes.

3.1 Basics

One of the first virtualized systems was the IBM Virtual Machine Facility/370 (VM/370) [21] in 1960 with a first abstraction and partitioning in host and guest. Nowadays a common definition is that

"*Virtualization provides a way of relaxing the forgoing constraints and increasing flexibility. When a system device (...), is virtualized, its interface and all resources visible through the interface are mapped onto the interface and resources of a real system actually implementing it.*" [22, p. 3]

The two classic approaches are either the use of a virtual machine monitor (VMM), a small operating system controlling the guest system's access to the hardware, or multiple guest systems embedded into a standard host operating system [23]:

- Type 1: Bare metal (VMM or Hypervisor)
- Type 2: Host operating system

Another distinction can be made on the level of code execution and driver access, where the relevant approaches are [22]:

- Hardware virtualization (full virtualization)
- Paravirtualization
- Hardware-assisted virtualization

An interesting starting point for FPGA virtualization is especially the VMM concept with paravirtualization in which the interfaces to the VMs are similar to those of the underlying hardware. The VM interfaces are modified to reduce the

time spent on performing operations which are substantially more difficult to run in a virtualized than in a non-virtualized environment. This kind of paravirtualized system is shown in Fig. 1(a). The unprivileged guests (DomU) run on a hypervisor which forwards calls from frontend driver to the backend driver of the management VM (Dom0).

3.2 FPGA Virtualization Approach

As demonstrated in Section 2, the term *virtualization* is used for a wide range of concepts. We decided to virtualize the FPGA like a paravirtualized VM executed by a VMM hypervisor. Fig. 1(b) shows an FPGA virtualization inspired by the paravirtualization introduced before. The virtualization is limited to the interfaces and inside the vFPGAs (DomU), every design is generated using the traditional design flow with predefined regions for partial reconfiguration. The vFPGAs can have different sizes (Fig. 3) and are completely independent from each other.

The interface to and from the vFPGAs is a so-called *frontend interface*, which is connected inside a virtualization layer to the *backend interface* in the static FPGA region. There, all frontends are mapped to the static PCIe endpoint and the on-board memory controller inside the Dom0, which also manages the states of the vFPGAs and the ICAP used for the reconfiguration. The internal state management allows for generating snapshots of each vFPGA's context, which offers the possibility to pause (context switch), stop, resume and migrate the whole user design. To offer these features, it is necessary to implement capture and restore mechanisms without additional hardware inside the vFPGA's user design. The vFPGA inside the DomU only has a predefined frontend interface and the user design itself does not have to be modified on the source or the netlist level. A profound virtualization requires context extraction and resumption only on the bitstream level. The migration we use is a so-called *warm* migration, because we stop the device without shutdown and new boot sequences.

4. VIRTUALIZATION FRAMEWORK

To implement the virtualization described in the previous section we developed a framework fully integrated into our cloud management system [24]. The components of the framework will be described in section 4.1 and the ICAP controller in section 4.2. In 4.3 we outline the novel states a virtualized design undergoes and section 4.4 combines the parts described before into the overall migration process.

4.1 Overall Hardware Design

The framework we developed in [24] is originally designed as a computing framework for reconfigurable coprocessors or acceleration cores connected to a host via PCIe. The framework provides easy access to resources like PCIe or DDR3 RAM for up to four individual users. The main part of the framework consists of a controller managing configuration and user cores as well as monitoring of status information. The controller's memory space is accessible from the host through an API and on the FPGA via dedicated control signals. In- and output-FIFOs are providing high throughput for streaming applications.

Based on our concept, we transform the accelerator designs into vFPGAs with additional state management and a static frontend interface as shown in Fig. 2. The *virtualization layer* manages the on-chip communication between backend and frontend interfaces for both, PCIe and with page tables virtualized DDR3 RAM. The virtualization layer provides the frontends to the vFPGAs. The number of frontends and their locations are defined by the provider. The *RC2F control* unit manages the ICAP controller and the *vControl* units, which maintain and monitor the vFPGAs.

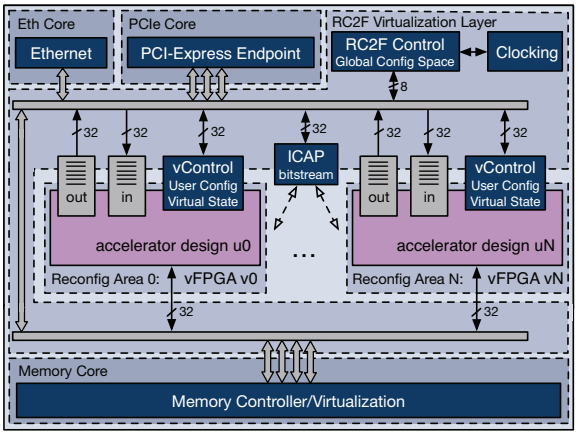


Figure 2: Computing framework with partial reconfigurable areas required for the vFPGAs.

In our examples we typically use four frontends on a Xilinx Virtex-7. Depending on the resources necessary, the utilization of up to four different-sized vFPGAs is possible with the same static part without reprogramming. If one of the vFPGAs covers all regions, only one frontend connection is used as shown in Fig. 3. Currently this feature is experimental because it is not provided directly in the manufacturer's tool flow.

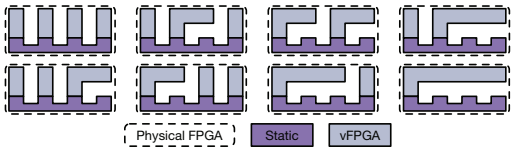


Figure 3: Abstract representation of all possible locations and sizes in a static virtualization layer in an example with four frontends.

4.2 ICAP for Configuration and Readback

Since partial reconfiguration was introduced in the year 2000 [25], many ICAP controller implementations have been proposed. However, to connect the optimal combination of speed, resource utilization and features to the framework, a new ICAP controller had to be developed. High throughput rates of 400 MB/s are important to minimize migration times and cannot be achieved with standard Xilinx components. The low resource utilization, as shown in Table 1, allows for more extensive accelerator designs. The readback feature is essential to our approach and must not have any overhead. Therefore, a FSM parses the readback commands arriving via PCIe and automatically triggers an ICAP readback without further interactions.

4.3 vFPGA States

Our FPGA virtualization includes states and transitions similar to traditional VMs. The virtualization of an FPGA requires off-chip monitoring and administration of the vFPGA bitstream database, connected to our cloud management sys-

Table 1: Resource utilization of the ICAP controller.

Component	LUT	FF	BRAM
ICAP FSM	96	106	0
Utilization*(%)	0.03	0.02	0.00
ICAP Controller	153	233	1
Utilization*(%)	0.05	0.04	0.10

* Xilinx VC707 board with a XC7VX485T

tem [24] as well as additional on-chip state transitions. Fig. 4 gives an overview of these two parts in our virtualization. A state with a control flow transition between host and FPGA is called *transition state*.

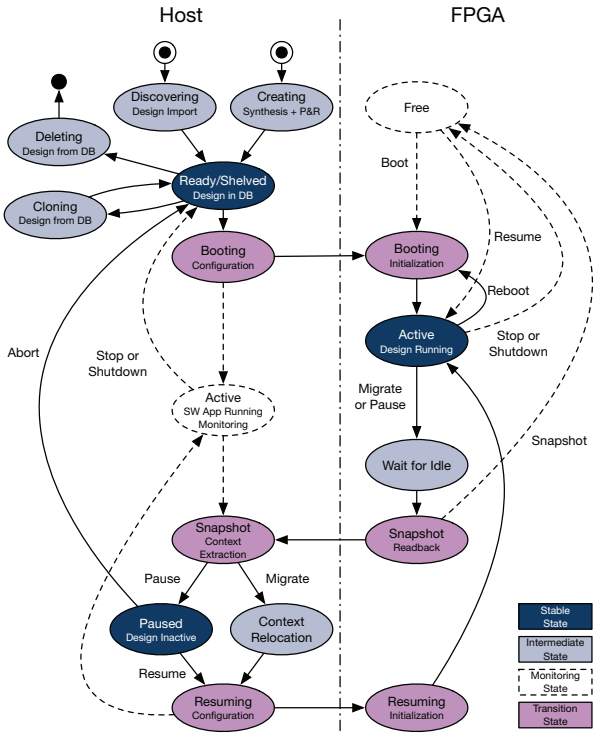


Figure 4: State transitions of a vFPGA (on the host and FPGA).

The global design database and the scheduling of the acceleration tasks (vFPGAs), the allocation to a node and a free region are performed by the cloud system, which also sends commands triggering state transitions on the host and FPGA devices. In the following, the most important states are described in detail:

Ready/Shelved: The vFPGA design is located in the global database on a management node.

Booting: First, the node containing the selected vFPGA has to verify if the actual vFPGAs is marked *free*. In a second step the boot process starts, where the partial bitstream is loaded from the database and written into it's respective vFPGA location using PCIe and ICAP.

Active: After initialization the vFPGA accelerator is *Active* and the corresponding host application can send/receive application data until a state transition occurs. In case of a *reboot* or *stop* command, the design is halted and reconfigured using the initial or an empty vFPGA design.

Wait for Idle: When a *migration* or *pause* command is received during the *active* state by the host, it forwards the command to the FPGA and both stop the computation and the transmission of further data. Host and vFPGA both wait a limited duration (timeout) until the last data packages are received and stored in the vFPGA's input FIFO and the application's memory.

Snapshot: After the timeout the context of the vFPGA is stable and the actual readback of the vFPGA design is performed by the host using the ICAP. Moreover the context extraction is performed (see Section 4.4.2) and it is stored in the *virtual register content* file (`.vrc`). At the same time the context of the host application is stored on disk.

Paused: In case of a *pause* command the software and hardware context are stored on disk. If an *abort* command follows, the vFPGA's context in the `.vrc` file becomes invalid (also the host application's context) and the vFPGA gets into the initial *Ready* state. In case of a *resume* command, the initial vFPGA's context is restored by modifying the bitstream using the `.vrc` file as shown in Section 4.4.3.

Context Relocation: If the state transition is triggered by a *migration* command, the next vFPGA region is known and the context relocation (bitstream modification) can be performed immediately with the bit positions provided by the `.vrc` file. The modified bitstream and the host application with the context from the previous run are transferred to the new vFPGA/node.

Resuming: The modified bitstream with the context from the previous run is used to boot or restore the old context on a different vFPGA.

The context of the vFPGA's DDR3 memory also needs to be saved and restored in the *snapshot* or *resuming* stage using the PCIe connection.

4.4 Virtualization and Migration Process

Our extended design flow, which generates partial bitstreams and supports vFPGA snapshots as well as the context resumption, is shown in Fig. 5. In the following, the components, all additional design flow steps and the generated metadata are described in detail.

4.4.1 vFPGAs Bitstream Generation and Boot

In the initial step, the full bitstream containing the static design is generated with the traditional Xilinx flow. The bitstream produced contains the basic components, such as PCIe endpoint, memory controller, virtualization layer including the ICAP controller and the static frontend interfaces as well as the local state management for the vFPGAs. The vFPGAs regions themselves are completely empty. The corresponding netlist (`.ngc`) of the static part and all bit positions are stored in the global database and are accessible for the production of partial bitstreams. The vFPGAs are always configured using the static design's ICAP via PCIe.

The following step includes the generation of a partial bitstream based on the static design and a netlist containing the vFPGA design. To achieve an efficient load-balancing and placement on the vFPGA, all possible bitstreams are produced in a single design flow run. For designs which require more than one vFPGA slots, additional partial bitstreams are generated in separate runs. The possible vFPGA

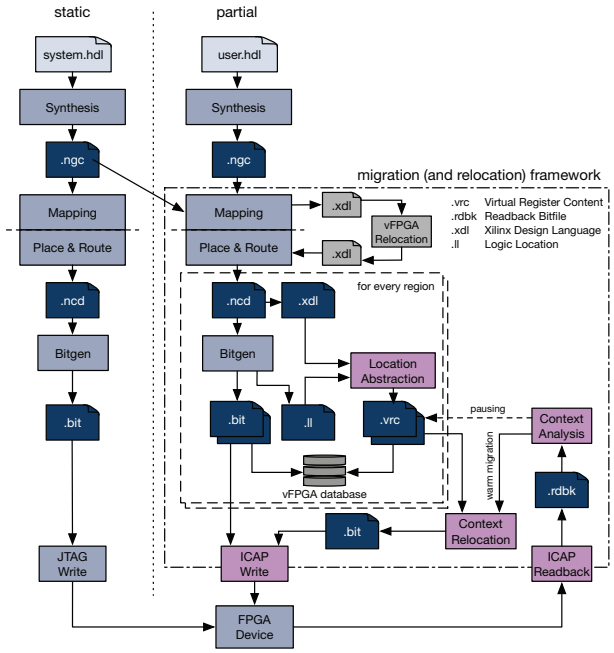


Figure 5: Extended design flow generating partial vFPGA bitstreams and it's additional metadata.

locations for an example with four vFPGAs on one physical FPGA, are shown in Fig. 3. The overall runtime will be reduced in the future by using relocation of placed vFPGAs using homogenous regions [26]. For the context resumption it is essential to set the option `RESET_AFTER_RECONFIG` for each vFPGA region.

A significant step is the generation of metadata out of these files, which is required to find the register and memory locations in all vFPGA bitstreams. We store the metadata in our *virtual register content* file (`.vrc`) as shown in Listing 3. The information required is extracted from the additional *Logic Location* files (`.ll` in Listing 2) and the *Xilinx Design Language* files (`.xdl` in Listing 1), which are generated during the design flow as shown in Fig. 5. The result of this step are partial bitstreams and the corresponding metadata for every possible region. Everything together is stored in the global vFPGA database.

Listing 1: XDL file with register location (`.xdl`)

```
inst "RC2F.cores[0].frame/reg[11]"
  "SLICEL",placed CLBLM_R_X107Y67 SLICE_X154Y67,
cfg " A5FFINIT::#OFF A5FFMUX::#OFF A5FFSR::#OFF
    ...
  D5LUT::#OFF D6LUT::#OFF DCY0::#OFF
  DFF:RC2f.cores[0].frame/reg_11:#FF
  DFFINIT::INIT0 DFFMUX::DX ... ";
```

Listing 2: Location file with bit locations (`.ll`)

```
Bit 0x403587 1122 Block=SLICE_X155Y67 ... Net=reg[2]
Bit 0x40359f 1146 Block=SLICE_X154Y67 ... Net=reg[11]
Bit 0x4035a0 1147 Block=SLICE_X155Y67 ... Net=reg[3]
```

4.4.2 Bitstream Readback and Context Extraction

In case of a *pause* or *migration* command, the FPGA is stopped as explained in Section 4.3. After the state became stable, the clock of the corresponding vFPGA is deactivated and the whole context (flip-flops and block RAMs) is frozen. At this point a readback for the `CLB/IO/CLK` and the `BRAM` block is performed and the context is extracted from the bitstream using the `.vrc` file. By the use of the location

Listing 3: Metadata with vFPGA context (`.vrc`)

```
"regFrame": [
  {
    "address":  "0x0040359f",
    "reg": [
      {
        "net":"/reg[11]",
        "offset":1146,
        "content":0
      } ]
  } ],
"ramFrame": [
  ... ]
```

metadata we only save the registers and the memory used in the design. The readback itself is performed on configuration frame level. In case of a *migration* the location of the new vFPGA is known and the context is written directly into a new bitstream. In case of *pause*, the extracted content is stored in the database as a copy of the `.vrc` file.

4.4.3 vFPGA Migration and Context Resumption

In this final step, the relocation and the context resumption are performed. The initial vFPGA bitstream and the corresponding `.vrc` file are used to generate a new bitstream by modifying certain configuration bits. The old flip-flop values are written into the positions of the register initialisation bits using the information in the `.vrc` file. To load the values into the flip-flops, the global set/reset (`GSR`) is triggered for the single vFPGA (not global). The Cyclic Redundancy Check (CRC) at the end of the readback bitstream is replaced by a `nop` command to ignore the old CRC.

5. UTILIZATION AND PERFORMANCE

In this section, we first show the additional overhead due to the virtualization with a maximum of four vFPGAs on a single physical device, followed by the time required for configuration, readback and relocation in Section 5.2.

5.1 Device Utilization

Table 2 shows the components' resource utilization for an example implementation with up to four vFPGAs frontends. On a Xilinx Virtex-7 XC7VX485T, the resource utilization for a basic design providing PCIe interface and four vFPGA interfaces is less than 3 %. The *Virtualization Layer* with frontends, *vControl* units and ICAP requires 2 % of the FPGA's resources because parts of the PCIe logic are moved to the frontend. As mentioned before, additional components for context readback and resumption are not necessary.

Table 2: Resource utilization of the individual components for an example with four static vFPGA frontends on a Virtex-7.

Component	LUT	FF	BRAM	Throughput in MByte/s
PCI Endpoint	3,268	3,592	8	≈ 800
Virtualization Layer	125	255	1	
4 vControl Units	211	121	0	
4 vFPGA Frontends	5,139	4,471	16	≈ 196-798
Total	8,743	8,439	25	
Utilization*(%)	2.9	1.5	2.3	

* Xilinx VC707 evaluation board with a XC7VX485T

Table 3: Time and Size of a Virtex-7.

Partial Bitstream	Configuration (Boot/Resume)	Snapshot Readback	Snapshot Relocate
4 MB	52 ms	953 ms	153 ms

5.2 Context Migration Overhead

The time necessary for the initial configuration using our ICAP controller, the readback and the relocate step are shown in Table 3. In this example we migrate a random number generating Monte Carlo Black-Scholes core, which utilize about 90 % of one vFPGA and about 62,000 registers spread across almost a thousand frames. The relocation, especially the readback is the most I/O intensive step and takes about 953 ms. Compared with ReconOS [17] our readback is ten times slower, because we read single configuration frames to reduce the computational overhead in the migration step. Our priority is an optimization of this bottleneck to reduce the migration time significantly. We plan to achieve this by reading larger contiguous blocks of the configuration frames.

6. CONCLUSIONS AND OUTLOOK

This paper presents an approach to integrate FPGAs as resources into a cloud environment using virtualization. Our definition of the term *virtualization* is inspired by traditional VMs whose functionality are transferred to reconfigurable hardware. We develop a paravirtualized structure on a physical FPGA device with multiple vFPGAs. The concept is integrated into a framework, which allows for interaction with the vFPGAs similar to traditional VMs. We implemented the possibility to pause and to migrate vFPGAs between regions of one physical FPGA and also among different FPGAs.

Our migration concept requires no additional logic inside the user design running on a vFPGA and therefore provides high flexibility. The virtualization overhead is less than 2 % on a Virtex-7 (XC7VX485T). Our measurements identify the bottleneck of our migration which requires up to 1 second for a user core utilising a single vFPGA.

In the future, we plan to reduce migration times and embed the virtualization into our cloud load balancer to demonstrate the functionality and especially the energy savings in a complex workload scenario. Furthermore, we plan to improve the relocation step to reduce the costs of a migration or context switch. Another interesting field of application for further research is an integration in safety-critical systems.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing", *National Institute of Standards and Technology Gaithersburg*, 2011.

[2] M. Armbrust, A. Fox, R. Griffith, *et al.*, "A view of cloud computing", *Communications of the ACM*, vol. 53, 2010.

[3] O. Knodel *et al.*, "Next-generation massively parallel short-read mapping on FPGAs", in *Application-Specific Systems, Architectures and Processors (ASAP), Int'l Conf. on*, IEEE, 2011.

[4] J.-A. Mondol, "Cloud security solutions using FPGA", in *Communications, Computers and Signal Processing*, IEEE, 2011.

[5] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services", in *Computer Architecture (ISCA), 41st Int'l Symp. on*, IEEE, 2014.

[6] W. Fornaciari and V. Piuri, "Virtual FPGAs: Some steps behind the physical barriers", in *Parallel and Distributed Processing*, Springer, 1998.

[7] Xilinx Inc., "Partial Reconfiguration – User Guide (UG702 v14.5)", April 26, 2013.

[8] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing", in *Field Programmable Logic and Applications (FPL), 2012 22nd Int'l Conf. on*, IEEE, 2012.

[9] J. Weerasinghe *et al.*, "Enabling FPGAs in Hyperscale Data Centers", in *Cloud and Big Data Computing, Int'l Conf. on*, IEEE, 2015.

[10] R. Kirchgessner *et al.*, "VirtualRC: a virtual FPGA platform for applications and tools portability", in *Proceedings of the ACM/SIGDA Int'l Symposium on Field Programmable Gate Arrays*, ACM, 2012.

[11] H. K.-H. So *et al.*, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH", *ACM Trans. on Embedded Computing Systems (TECS)*, 2008.

[12] W. Wang *et al.*, "pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment.", *Hardware/Software Codesign and System Synthesis, Int'l Conf. on*, 2013.

[13] F. Chen *et al.*, "Enabling FPGAs in the cloud", in *Proc. of the 11th Conf. on Computing Frontiers*, ACM, 2014.

[14] S. Byma *et al.*, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack", in *Field-Programmable Custom Computing Machines (FCCM), 22nd Int'l Symp. on*, IEEE, 2014.

[15] K. Jozwik *et al.*, "A novel mechanism for effective hardware task preemption in dynamically reconfigurable systems", in *Field Programmable Logic and Applications (FPL), 2010 Int'l Conf. on*, IEEE, 2010.

[16] A. Morales-Villanueva and A. Gordon-Ross, "HTR: on-chip hardware task relocation for partially reconfigurable FPGAs", in *Reconfigurable Computing: Architectures, Tools and Applications*, Springer, 2013.

[17] M. Happe, A. Traber, and A. Keller, "Preemptive Hardware Multitasking in ReconOS", in *Applied Reconfigurable Computing*, Springer, 2015.

[18] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing: concepts, overhead analysis, and implementation", in *Proc. of the ACM/SIGDA 15th Int'l Symp. on Field programmable gate arrays*, ACM, 2007.

[19] S. Jovanovic *et al.*, "A hardware preemptive multitasking mechanism based on scan-path register structure for FPGA-based reconfigurable systems", in *Adaptive Hardware and Systems (AHS). NASA/ESA Conf. on*, IEEE, 2007.

[20] S. A. Fahmy *et al.*, "Virtualized FPGA accelerators for efficient cloud computing", in *Cloud Computing Technology and Science (CloudCom), Int'l Conf. on*, IEEE, 2015.

[21] R. P. Goldberg, "Survey of virtual machine research", *Computer Journal*, vol. 7, no. 6, 1974.

[22] J. Smith and R. Nair, *Virtual machines: versatile platforms for systems and processes*. Elsevier, 2005.

[23] M. Rosenblum, "The Reincarnation of Virtual Machines.", *ACM Queue*, vol. 2, no. 5, 2004.

[24] O. Knodel *et al.*, "RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment", *FPGAs for Software Programmers (FSP), Second Int'l Workshop on*, 2015.

[25] Xilinx Inc., "Virtex-II Platform FPGA – User Guide (UG002 v1.0)", June 12, 2000.

[26] R. Backasch *et al.*, "Identifying homogenous reconfigurable regions in heterogeneous FPGAs for module relocation", in *ReConFigurable Computing and FPGAs (ReConFig), Int'l Conf. on*, IEEE, 2014.