Virtualizing Reconfigurable Hardware to Provide Scalability in Cloud Architectures

Oliver Knodel, Paul R. Genssler and Rainer G. Spallek

Department of Computer Science
Technische Universität Dresden
Dresden, Germany
Email: {firstname.lastname}@tu-dresden.de

***Abstract*—Field Programmable Gate Arrays (FPGAs) provide a promising opportunity to improve performance, security and energy efficiency of computing architectures, which are essential in modern data centers. Especially the background acceleration of complex and computationally intensive tasks is an important field of application. The flexible use of reconfigurable devices within a cloud context requires abstraction from the actual hardware through virtualization to offer these resources to service providers. In this paper, we enhance our related Reconfigurable Common Computing Frame (RC2F) approach, which is inspired by system virtual machines, for the profound virtualization of reconfigurable hardware in cloud services. Using partial reconfiguration, our hardware and software framework virtualizes physical FPGAs to provide multiple independent user designs on a single device. Essential components are the management of the virtual user-defined accelerators (vFPGAs), as well as their migration between physical FPGAs to achieve higher system-wide utilization levels. We create homogenous partitions on top of an inhomogeneous FPGA fabric to offer an abstraction from physical location, size and access to the real hardware. We demonstrate the possibilities and the resource trade-off of our approach in a basic scenario. Moreover, we present future perspectives for the use of FPGAs in cloud-based environments.**

Keywords–Cloud Computing; Virtualization; Reconfigurable Hardware; Partial Reconfiguration.

I. 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 within a cloud. Even in peak load situations, a sufficient amount of resources are available [2].

With the theoretically unlimited number of resources, their enormous energy consumption arises as a major problem for data centers housing clouds. One possibility to enhance computation performance by simultaneously lowering energy consumption is the use of heterogeneous systems, offloading computationally intensive applications to special hardware co-processors 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 center 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 provision of virtual FPGAs (vFPGAs) makes reconfigurable resources available to customers of the data center provider. Therefore, service providers will be called *users* throughout this paper. The users can accelerate specific services, reduce energy consumption and thereby 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, FPGAs have grown in size and full utilization of the devices cannot always be achieved in practice. One possibility to increase utilization is our virtualization approach which allows for flexible design sizes and multiple hardware designs on the same physical FPGA. One challenge of this approach are the unsteady load situations of elastic clouds, which process short- and long-running acceleration services.

In this paper, 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 vFPGA can be configured using partial reconfiguration [7] and the internal configuration access port (ICAP). The vFPGAs are, therefore, flexible in their physical size and location. Moreover, they are fully homogenous among each other and thereby become a wholesome virtualized cloud component, which supports even an efficient migration of a whole vFPGA context. Especially the vertical scaleability of vFPGAs from small designs up to full physical FPGAs is gaining importance by providing efficient utilization of the reconfigurable resources in modern cloud architectures.

The paper is structured as follows. Section II introduces similar concepts and related research in the field of virtualization of reconfigurable hardware, cloud architectures and bitstream relocation. In Section III, we give an overview on our virtualization concept. Our prototype, which implements our concept with homogenous and in their size flexible vFPGAs, is presented in Section IV followed by device utilization, vFPGA size and performance results in Section V. Section VI concludes and gives an outlook.

II. RELATED WORK

The provisoning of reconfigurable hardware in data centers and cloud environments has gained more and more importance in the last years as shown by the overview from Kachris et al. [8]. Initially used mainly on the network infrastructure level, FPGAs are now also employed on the application level of

 ISBN: 978-1-61208-585-2

data centers. Typical use cases in this field are background accelerations of specific functions with static hardware designs. The FPGAs' special feature to reconfigure hardware at runtime is still used rather rarely. Examples are the anonymization of user requests [9] 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 due to a missing provisioning or virtualization hidden deeply in the data center.

A comparable contribution with stronger focus on the transfer of applications into an FPGA grid for high performance computing is shown in [10]. The application focus on a single cloud service model with background acceleration of services using FPGAs. An approach which places multiple user designs on a single FPGA is introduced by Fahmy et al. [11], using tightly attached FPGAs to offload computationally intensive tasks. The FPGAs are partially reconfigurable and can hold up to four individual user designs. The approach was extended by Asiatici et al. in [12] with additional memory virtualization. A cloud integration model with network-attached FPGAs and multiple user designs on one FPGA is introduced by Weerasinghe et al. [13].

The term *virtualization* itself is used for a wide range of concepts. An example for abstractions on the hardware description level is VirtualRC [14], which uses a uniform hardware / software interface to realize communication on different FPGA platforms. BORPH [15] provides a similar approach, employing a homogeneous UNIX interface for hardware and software. The FPGA paravirtualization pvFPGA [16], which integrates 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 developed by Chen et al. [17] and Byma et al. [18].

Approaches more closely related to the *context-save-and-restore* mechanism required by our migration concept can be found in the field of bitstream readback, manipulation and hardware preemption. In ReconOS [19], 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. In combination with homogenous bitstreams for different physical vFPGA positions, methods like relocation of designs as shown in [20], provide an opportunity for an efficient context migration of virtualized FPGAs.

III. FPGA Virtualization Approach

As the cloud itself is based on virtualization, the integration of FPGAs requires a profound virtualization of the reconfigurable devices in order to provide the vFPGAs as good as other resources in the cloud. Furthermore, it is necessary to abstract from the underlying physical hardware.

A. Requirements for Virtual FPGAs in a Cloud Environemnt

As discussed in Section II, the term *virtualization* is used for a wide range of concepts. The application areas of FPGAs in clouds require a direct use of the FPGA resources to be efficient. Thus, an abstraction from the physical FPGA infrastructure is only possible in size and location. Our approach is related to traditional system virtualization with VMs that corresponds to a Type-1 bare-metal virtualization with use of a hypervisor [21]. This kind of virtualization is designed for

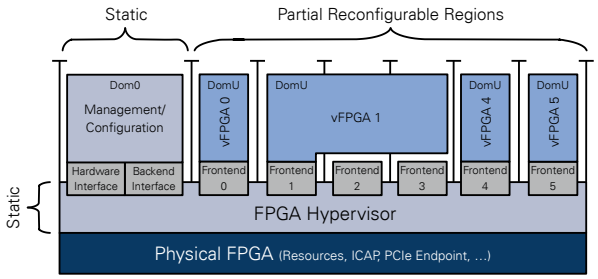


Figure 1. Paravirtualization concept used in RC2F to provide virtual FPGAs (vFPGAs) using partial reconfiguration. vFPGAs can be combined to group larger regions and thereby provide more resources.

the efficient utilization of the physical hardware with multiple users. Therefore, it is necessary to adapt the required FPGA resources closely to the requirements of the users' hardware design capsuled by vFPGAs. By this, an efficient utilization of the physical hardware with multiple concurrent vFPGAs on the same hardware can be achieved.

Furthermore, the vFPGA has to appear as a fully usable physical FPGA with separated interfaces and its own infrastructure management like clocking and resetting. For an efficient cloud architecture which requires elasticity [1], it is necessary to migrate vFPGAs with their complete context (registers and BlockRAM), which requires to enclose a complete state management of the vFPGA as described in [22]. An extraction of internal DSP registers is not supported in recent Xilinx FPGAs and must be considered in the design.

B. FPGA Virtualization Approach

We decided to virtualize the FPGA similar to a paravirtualized system VM executed by a hypervisor to provide access to the interfaces. Figure 1 shows an FPGA virtualization inspired by the paravirtualization introduced before. The virtualization is limited to the interfaces and the designs inside the reconfigurable regions, which constitute the actual vFPGAs as unprivileged Domain (DomU). Each vFPGA design is generated using the traditional design flow with predefined regions for dynamic partial reconfiguration [7] and static interfaces. The vFPGAs can have different sizes (Figure 1) and operate completely independent from each other. The infrastructure encapsulating the vFPGAs has to be located in the static region corresponding to a privileged domain (Dom0) or hypervisor.

The interface providing access to the vFPGAs is a so-called *frontend interface*, which is connected inside the hypervisor to the *backend interface* in the static FPGA region. There, all frontends are mapped to the static PCIe-Endpoint and the onboard memory controller inside the Dom0, which also manages the states of the vFPGAs.

IV. FPGA Prototype RC2F

Our prototype RC2F introduced in [23] provides multiple concurrent vFPGAs allocated by different users on a single physical FPGA. The main part of the FPGA frame(work) consists of a hypervisor 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. Input- and output-FIFOs are providing high throughput for streaming applications. The vFPGAs appear to the user as individual devices inside the System VM on the host.

 ISBN: 978-1-61208-585-2

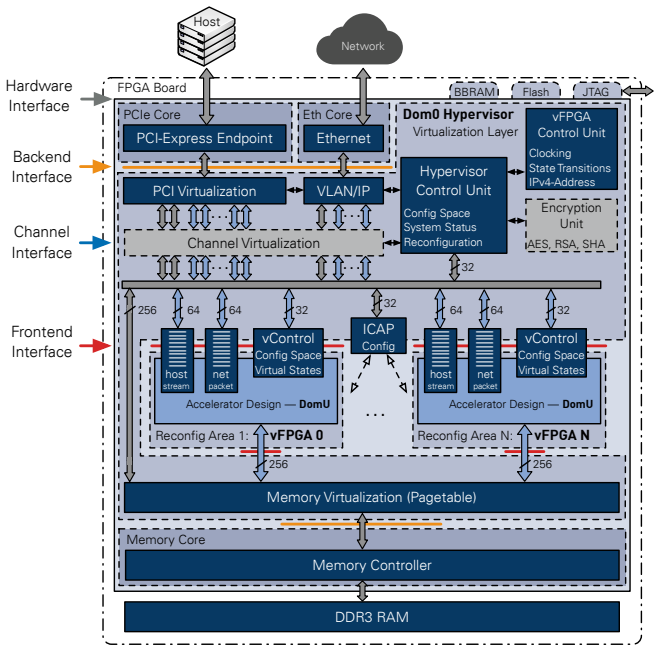


Figure 2. Virtualization frame RC2F with hypervisor, I/O components and partial reconfigurable areas housing the vFPGAs. The vFPGAs have access to the host using PCIe (FIFO interface and config space), to the Cloud network using Ethernet and the virtualized DDR3 memory.

A. System Architecture

The physical FPGAs are located inside a host system and are accessible via PCIe. On both hardware components (host and FPGA), there are hypervisors managing access, assignment and configuration of the (v)FPGAs. Based on our concept, we transform the FPGAs into vFPGAs with an additional state management and a static frontend interface as shown in Figure 1. Our architecture, designed to provide the vFPGAs, is shown in Figure 2. The hypervisors manage the on-chip communication between backend and frontend interfaces for PCIe (Our prototype uses a PCIe-Core from Xillybus for DMA access [24]), Ethernet and a DDR3 RAM. The RAM is virtualized using page tables, managed by the host hypervisor, which also manages the vFPGA states we introduced in [22]. The number of frontends and their locations are defined by the physical FPGA architecture as shown in Figure 6. The *Hypervisor Control Unit* manages the ICAP controller and the *vControl* units, which maintain and monitor the vFPGAs.

To exchange large amounts of data between the host (VM) and the vFPGAs a FIFO interface is used. To exchange state and control information the vFPGAs can be controlled by the user via a memory interface as shown in Figure 3. The memory is mainly intended for simple transfers and configuration tasks like resets, state management (pause, run, readback, migrate) and the selection of a vFPGA system clock. In addition to these static fields, there is also a user-describable memory region which can be used as virtual I/O. The communication using Ethernet is also provided but out of the scope of this paper.

B. Configuration of the FPGA Hypervisor

The tasks of the FPGA hypervisor are the management of its local vFPGAs and their encapsulation, the state management, as well as the reconfiguration using the ICAP. The interaction between host and FPGA hypervisor is based on the configuration memory shown in Figure 4, which includes

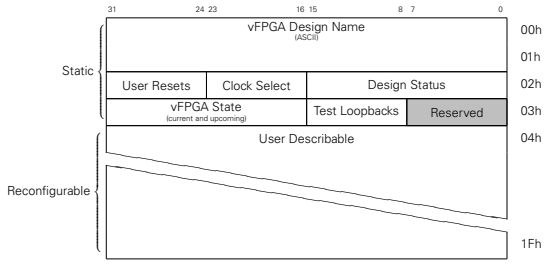


Figure 3. Register and memory interface for the management of vFPGAs accessible by the user VM (rc2f_cs).

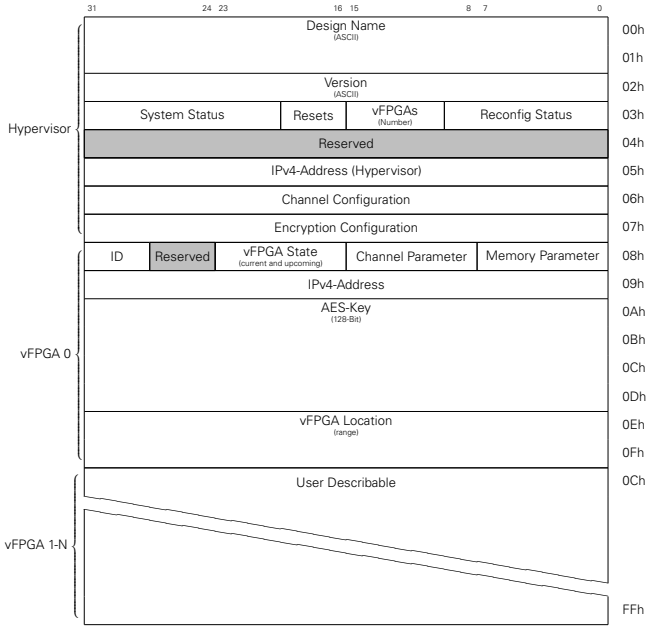


Figure 4. Register and memory interface for the management of the FPGA hypervisor accessible by the host hypervisor (rc2f_gcs).

configuration of the FPGA hypervisor (system status, reconfiguration data and status) and the administration of the vFPGAs. Other important vFPGA-related entries are an AES-key for encryption of the vFPGA-bitsteam and the allocated vFPGA region(s) for additional validation during reconfiguration. The information inside the FPGA hypervisor are only accessible and modifiable through the host hypervisor.

C. The Role of the Host-Hypervisor

Our virtualization concept on the host-system includes passing through the vFPGAs' FIFO channels and the configuration memories from the host-hypervisor to the user VMs (DomU) and the FPGA hypervisor memory to the management VM (Dom0). The overall system architecture on hypervisor level of host and FPGA is shown in Figure 5. The frontend FIFOs and the FPGA memories are mapped to device files inside the host hypervisor. There, our system forwards the user devices to the assigned VM using inter-domain communication based on vChan from Zhang et al. [25] in our Xen virtualized environment, similar to pvFPGA [16].

The management VM thereby accesses the FPGA hypervisor's configuration memory and the ICAP on the FPGA via a dedicated FIFO interface for the configuration stream (read and write). Thus, only the hypervisors can configure the vFPGA regions on the physical FPGA whereby a sufficient level of security can be guaranteed.

 ISBN: 978-1-61208-585-2

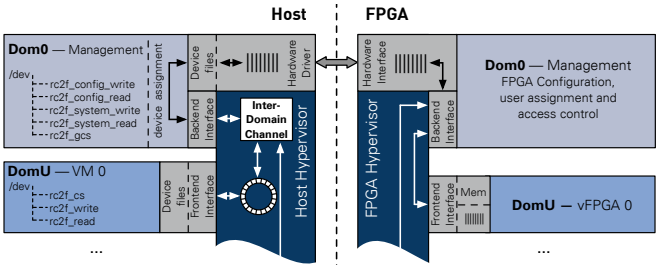


Figure 5. System architecture on the hypervisor level of the host system. FIFOs (rc2f_write, rc2f_read) and configuration memories (rc2f_cs) are displayed in the different host memories.

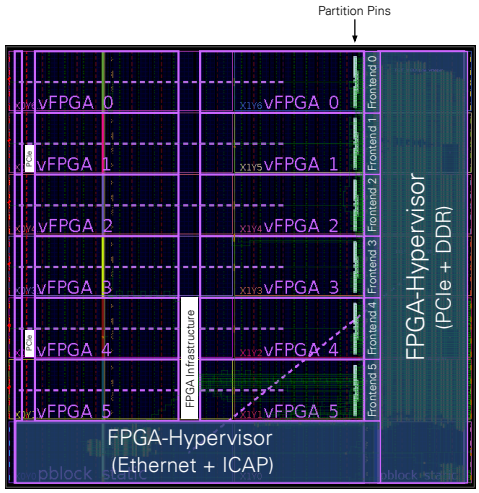


Figure 6. Layout of a Xilinx Virtex-7 XC7VX485T with six vFPGA regions configurable using dynamic partial reconfiguration. The regions and their number are determined by the height of the configuration frames, which consist of one complete column inside a clock region. Regions are homogenous to allow migration of vFPGAs.

D. Mapping vFPGAs onto physical FPGAs

In our example we use six frontends on a Xilinx Virtex-7. Depending on the resources required, the utilization of up to six different-sized vFPGAs is possible with the same static without reprogramming. If one of the vFPGAs covers more than one region, only one frontend connection is used as shown in Figure 1. Among the vFPGAs, the partition pins (PP) between the static and the reconfigurable regions are placed with identical column offset as shown in Figure 6. The regions forming the vFPGAs are not free from static routes as for example the region vFPGA 5 shows.

To reduce migration times, all components which hold the context of the current vFPGA design as registers, FIFOs or BlockRAM, are placed at the same positions inside each vFPGA. Therefore, it is necessary that all of these positions exist in each region. Hardmacros like PCIe-Endpoints or parts of the FPGA infrastructure interrupt the homogenous structures. Thus, we establish homogenous vFPGAs, which are identical among each other by excluding these areas in all vFPGAs as shown in Figure 6. The advantage of this approach is that only one mask file is necessary to extract the content of the different vFPGAs. Furthermore, it allows the provision of almost identical vFPGAs.

E. Extended Design flow

For our virtualization we extend the Xilinx Vivado design flow to generate vFPGA bitstreams from user-netlists for every possible vFPGA position. First, directly after synthesis the required region size (single, double, etc.) is chosen (see Table I for appropriate vFPGAs). Afterwards, the design is placed at a first vFPGA region. Before the routing step, the vFPGA region is expanded over the full width of the vFPGA for unlimited routing of the design inside the uninterrupted region. The placements of the same design for all the other vFPGA positions are created by setting the LOC (Location) and BEL (Basic Element Location) information accordingly to the initial placed design. Only the routing is carried out for the additional vFPGA designs to allow static routes inside the different vFPGAs, resulting in designs with identical register and BlockRAM positions for each vFPGA locations on the physical FPGA. After generation of the first bitstream, a mask for extracting the context bits is generated to allow an efficient migration in significantly less time compared to our first approach in [22]. This allows flexible placement of the vFPGA designs at various positions in a cloud system, as well as the migration between vFPGAs on the same or to other physical FPGAs. The bitstreams required for all possible vFPGA positions belonging to a single user design are stored as virtual reconfigurable accelerator images (vRAI).

F. Description of vFPGAs

The execution of a vRAI requires allocation of a vFPGA which fulfills all requirements. Therefore, it is necessary to describe the vFPGAs in a particular configuration file. Figure 7 gives an overview of such an configuration, which is evaluated by the resource management system to allocate the necessary resources. After allocation the host hypervisor chooses from the vRAI the appropriate bitstream and configures the device.

```
service = 'ba'              #Background Acceleration Service
name = 'vfpga-kmeans'       #vFPGA/User Design Name
vm = ['vm1-pvm']            #VM-Instance Name
vfpga = 1                   #Number of vFPGA
size = [3]                  #vFPGA Size
memory = [2000]             #DDR-Memory Size in MByte
vif = ['ip=10.0.0.43']      #vFPGA-IP
boot = ['running']          #Initial vFPGA-State
design = ['kmeans.vrai']#Initial Design
```

Figure 7. Configuration file for the allocation of a single vFPGA with network access and external memory of 2 GByte.

V. Implementation Results and Scenario

The resources required for the implementation described in the previous section are shown in the following with a real-world scenario based on our motivation from Section I.

A. Implementation

The resource consumption of our prototype introduced in Figure 2 is shown in Table I. Furthermore, the table introduces the size of homogenous vFPGA regions as outlined in Figure 6.

$$\vec{\rho} = \begin{pmatrix} SliceLUTs \\ SliceRegister \\ BlockRAM \\ DSP \\ \dots \end{pmatrix} \quad (1)$$

is used in the following to describe the resources. The aggregated homogenous vFPGAs $\overrightarrow{\rho_{agg}}$ can be calculated using

$$\overrightarrow{\rho_{agg}} = \overrightarrow{\rho_{single}} \cdot n_{agg} - (n_{agg} - 1) \cdot \overrightarrow{\rho_{ppr}} \quad (2)$$

where $\overrightarrow{\rho_{single}}$ are the resources of a single vFPGA region, n_{agg} is the number of aggregated vFPGAs and $\overrightarrow{\rho_{ppr}}$ represents the

partition pin region (PPR) necessary to exclude the unused frontend interfaces from the grouped vFPGAs. The open frontends are therefore treated as stubs and are securely sealed using a partial vFPGA bitstream. The cost of the provision of identical vFPGAs are in the case of our Virtex-7 XC7VX485T FPGA only 6.44% of slices registers/LUTs and 8.33% of the BlockRAM tiles compared to a compete, but inhomogeneous region. In our floor planning shown in Figure 6 there are no further DSPs affected. All regions except the largest one (Hexa), which has only one possible position, are homogenous.

The throughput between vFPGAs and host (PCIe Gen2 8x on a Xilinx VC707) with different numbers of concurrently active vFPGAs is shown in Figure 8. The throughput of a single design is limited by a user clock of 100 MHz and a 64-bit data interface. Starting from three vFPGAs, a limitation due to the concurrent users occurs. The throughput shown in Figure 8 is the minimal guaranteed throughput for each vFPGA.

The size of the vRAI packages and the number of possible locations on the physical device are shown in Table II. With 69.2 MByte, a quad vFPGA with bitstreams for three possible positions and a mask file for context migration is the largest vRAI package. Compared to our first approach, the information necessary for context migration is reduced by several orders of magnitude by using homogenous vFPGAs.

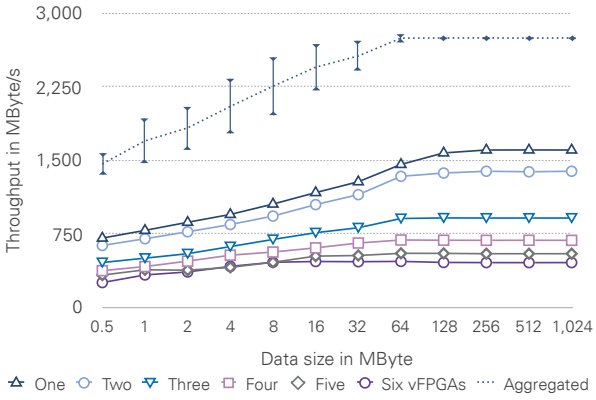


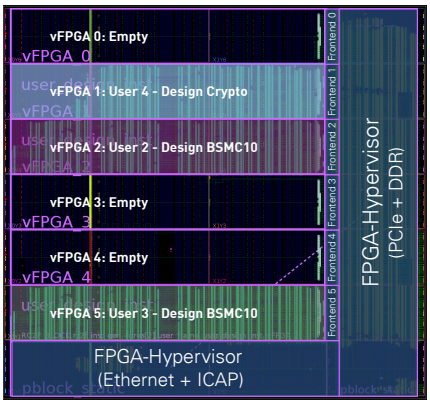
Figure 8. Throughput between host and FPGA with different numbers of concurrent vFPGAs. The diagram shows for each number of vFPGAs the average throughput of one representative vFPGA. The aggregated throughput is thereby the average throughput of all vFPGA compositions on the device.

B. Scenario

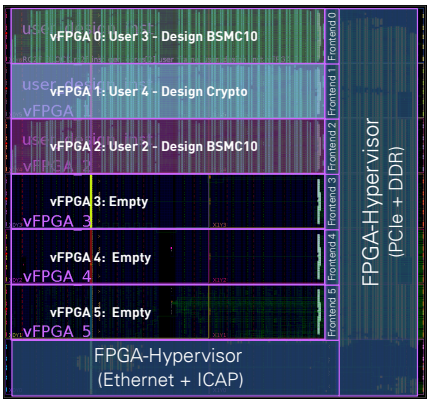
In the following, we show a scenario based on a typical real-world application for our virtualization approach. The goal is it to migrate vFPGA designs to achieve a high utilization as shown in Figure 9(c). In a system with jobs arriving and being finished at different points in time, situations as shown in Figure 9(a) can occur. The fragmentation of the physical FPGA restricts only one small vFPGA and one aggregated double sized vFPGA. By migrating the design from user 3 from vFPGA 5 to vFPGA 0 as shown in Figure 9(b), an area for a group of three vFPGAs (triple) becomes available and makes higher utilization of the physical device possible.

VI. Conclusion and Outlook

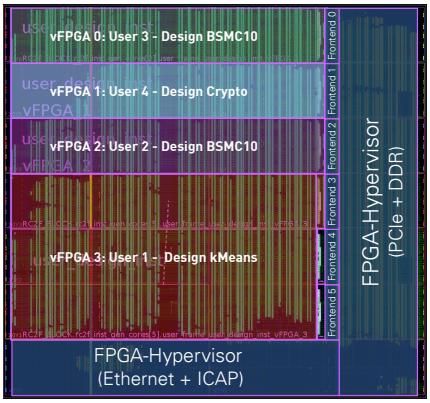
This paper presented a comprehensive virtualization concept for reconfigurable hardware and its integration into a cloud environment. Our definition of the term *virtualization* is inspired by traditional VMs whose functionalities are transferred to reconfigurable hardware. We develop a paravirtualized infrastructure 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 create homogenous regions for the vFPGAs on the physical FPGA to optimize the process of vFPGA migration between different physical FPGAs. Implementation details are described, the necessary resources and the virtualization overhead are presented.



(a) Fragmentation of the physical FPGA caused by dynamic de- and allocation.



(b) Defragmentation providing aggregated vFPGA regions for larger designs.



(c) Utilization of the free region with a design using three aggregated vFPGAs (Triple).

Figure 9. Szenario with different users and designs on a Xilinx Virtex-7 XC7VX485T with six (vertically) scaleable vFPGAs.

TABLE I. NUMBER OF AVAILABLE RESOURCES INSIDE THE STATIC AND THE AGGREGATED VFPGA REGIONS AND UTILIZATION OF STATIC CONTAINING INFRASTRUCTURE AND HYPERVISOR. THE PARTITION PIN REGION (PPR) IS NECESSARY TO EXCLUDE AND ISOLATE UNUSED PARTITION PINS (PP).

Ressource	Static region	Utilization of static region					PPR	Into aggregated vFPGA regions					
		HF[a]	P[b]	E[c]	M[d]	Total		Single	Dual	Triple	Quad	Quint	Hexa[e]
Slice LUTs	94,824	26%	3%	2%	11%	**42%**	1,200	30,800	60,400	90,000	120,800	151,600	188,400
Slice Register	189,648	11%	2%	1%	4%	**18%**	2,400	61,600	120,800	180,000	241,600	303,200	376,800
Block RAM Tile	369	23%	2%	2%	3%	**30%**	0	100	200	300	400	500	600
DSPs	726	–	–	–	–	**–**	20	340	660	980	1,320	1,660	1,940

[a] **HF**: Hypervisor and Frontends [b] **P**: PCIe-Endpoint [c] **E**: Ethernet [d] **M**: DDR3 Memory [e] Largest region without considering homogeneity

TABLE II. SIZE OF A SINGLE BITSTREAM FOR A VFPGA REGION, NUMBER OF POSSIBLE POSITIONS INSIDE THE FPGA AND SIZE OF THE VRAIS.

	Single	Dual	Triple	Quad	Quint	Hexa
Bitstream (MByte)	4.8	9.0	13.0	17.3	21.3	25.3
Locations	6	5	4	3	2	1
vRAI (MByte)	33.6	54.0	65.0	69.2	63.9	50.6

One significant result of this paper is that the provision of homogenous FPGA resources is possible with state-of-the-art FPGAs. We think that such approaches are necessary for establishing FPGAs in modern data centers housing clouds. Certainly, when cloud providers like Amazon expand their cloud architectures with high-end FPGAs, such as Xilinx Virtex-7 UltraScale devices [26] it is necessary to utilize the hardware efficiently with multiple designs in a scaleable frame inside one physical FPGA. Such kind of flexible approach allows for adaption the individual resources to the users' requirements.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing, Revised", *Computer Security Division, Information Technology Laboratory, NIST Gaithersburg*, 2011.

[2] M. Armbrust, A. Fox, R. Griffith, *et al.*, "A view of cloud computing", *Communications of the ACM*, vol. 53, pp. 50–58, 2010.

[3] T. El-Ghazawi, E. El-Araby, M. Huang, *et al.*, "The promise of high-performance reconfigurable computing", *IEEE Computer*, vol. 41, no. 2, pp. 69–76, 2008.

[4] J.-A. Mondol, "Cloud security solutions using FPGA", in *PacRim, Pacific Rim Conf. on*, IEEE, 2011, pp. 747–752.

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

[6] W. Fornaciari and V. Piuri, "Virtual FPGAs: Some steps behind the physical barriers", in *Parallel and Distributed Processing*, Springer, 1998, pp. 7–12.

[7] Xilinx Inc., *Vivado Design Suite User Guide – Partial Reconfiguration*, UG909 (v2017.1), April 5, 2017.

[8] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing", in *Field Programmable Logic and Applications (FPL), 26th Int'l Conf. on*, 2016.

[9] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing", in *Field Programmable Logic and Applications (FPL), 22nd Int'l Conf. on*, IEEE, 2012, pp. 63–70.

[10] J. Dondo Gazzano, F. Sanchez Molina, F. Rincon, and J. C. López, "Integrating reconfigurable hardware-based grid for high performance computing", *The Scientific World Journal*, 2015.

[11] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing", in *Cloud Computing Technology (CloudCom), Int'l Conf. on*, IEEE, 2015.

[12] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, "Designing a virtual runtime for FPGA accelerators in the cloud", in *Field Programmable Logic and Applications, Int'l Conf. on*, 2016.

[13] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in Hyperscale Data Centers", in *Cloud and Big Data Computing (CBDCom), Int'l Conf. on*, IEEE, 2015.

[14] R. Kirchgessner, G. Stitt, A. George, and H. Lam, "VirtualRC: a virtual FPGA platform for applications and tools portability", in *FPGAs, Proc. of the ACM/SIGDA Int'l Symp. on*, 2012.

[15] H. K.-H. So and R. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 2, p. 14, 2008.

[16] W. Wang, M. Bolic, and J. Parri, "pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment.", *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 Int'l Conf. on*, pp. 1–9, 2013.

[17] F. Chen, Y. Shan, Y. Zhang, *et al.*, "Enabling FPGAs in the cloud", in *Computing Frontiers, Proc. of the 11th ACM Conf. on*, ACM, 2014, p. 3.

[18] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack", in *Field-Programmable Custom Computing Machines (FCCM), 22nd Annual Int'l Symp. on*, IEEE, 2014, pp. 109–116.

[19] M. Happe, A. Traber, and A. Keller, "Preemptive Hardware Multitasking in ReconOS", in *Applied Reconfigurable Computing*, Springer, 2015, pp. 79–90.

[20] J. Rettkowski, K. Friesen, and D. Göhringer, "RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs", in *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*, IEEE, 2016, pp. 1–8.

[21] J. E. Smith and R. Nair, "The architecture of virtual machines", *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[22] O. Knodel, P. Genßler, and R. Spallek, "Migration of long-running tasks between reconfigurable resources using virtualization", in *ACM SIGARCH Computer Architecture News Volume 44, HEART 2016*, ACM, 2016.

[23] O. Knodel and R. G. Spallek, "Computing framework for dynamic integration of reconfigurable resources in a cloud", in *2015 Euromicro Conference on Digital System Design, DSD 2015*, IEEE, 2015, pp. 337–344.

[24] Xillybus Ltd., Haifa, Israel, *An FPGA IP core for easy DMA over PCIe*, Website, Online: http://xillybus.com, 2017.

[25] X. Zhang, S. McIntosh, P. Rohatgi, and J. L. Griffin, "Xensocket: A high-throughput interdomain transport for virtual machines", in *Middleware 2007*, Springer, 2007, pp. 184–203.

[26] Amazon Inc., *Amazon EC2 F1 Instances – Run Custom FPGAs in the AWS Cloud*, Website, Online: https://aws.amazon.com/ec2/instance-types/f1/, 2017.