FPGAs and the Cloud – An Endless Tale of Virtualization, Elasticity and Efficiency

Oliver Knodel*‡, Paul R. Genssler†‡, Fredo Erxleben‡ and Rainer G. Spallek‡

* Department of Information Services and Computing, Helmholtz-Zentrum Dresden-Rossendorf, Dresden, Germany
† Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany
‡ Department of Computer Science, Technische Universität Dresden, Dresden, Germany
Email: *o.knodel@hzdr.de, †genssler@kit.edu, ‡{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 present our Reconfigurable Common Computing Frame (RC2F) approach – inspired by system virtual machines – for the profound virtualization of reconfigurable hardware in cloud services. Using partial reconfiguration, our framework abstracts a single physical FPGA into multiple independent virtual FPGAs (vFPGAs). A user can request vFPGAs of different size for optimal resource utilization and energy efficiency of the whole cloud system. To enable such flexibility, we create homogeneous partitions on top of an inhomogeneous FPGA fabric abstracting from physical locations and static areas. The RC2F$_{\text{SEC}}$ extension combines this virtualization with a security system to allow for processing of sensitive data. On the host side our Reconfigurable Common Cloud Computing Environment (RC3E) offers different service models and manages the allocation of the dynamic vFPGAs. 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

The idea of FPGAs as virtualized resources in Cloud environments in the projects RC3E and RC2F was temporarily completed with introducing homogeneous virtualized FPGAs in 2017 by Knodel et al. in [1]. This article henceforth describes the two parts of out project – RC3E and RC2F – beginning with first considerations related to FPGA-Clusters in [2]. First cloud approaches with service models were introduced in [3] and [4], the overall RC3E-Cloud description in [5], a hardware migration in [6] and additional security considerations by Genssler et al. in [7].

Cloud computing itself is based on the idea of computing as a utility [8]. 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" [9]. 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 [8].

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 coprocessors or dedicated accelerators. Especially reconfigurable hardware, such as FPGAs, provide an opportunity to improve computing performance [10], security [11] and energy efficiency [12].

A profound and flexible integration of FPGAs into scalable data center infrastructures, which satisfies 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. Provisioning vFPGAs makes reconfigurable resources available to customers of the data center provider. These customers are usually service providers themselves – nevertheless, they will be called *users* throughout this paper. Those users can accelerate their 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 [13]. 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 tasks.

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 [14] and the internal configuration access port (ICAP). The vFPGAs are, therefore, flexible in their physical size and location. This vertical scalability of vFPGAs from a small design up to a full physical FPGA enables an efficient utilization of the reconfigurable resources. Moreover, the vFPGAs are fully homogeneous among each other and thereby become a wholesome virtualized cloud component, which also supports an efficient migration of a whole vFPGA context.

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. The requirements for a profound provision of FPGAs in a cloud environment are discussed in Section III. Section IV introduces the prototypical cloud management system RC3E followed by definitions necessary for the virtualization of the FPGAs themselves in Section V. In Section VI, we give an overview on our FPGA related virtualization concept RC2F. Our prototype, which implements our concept with homogeneous and in their size flexible vFPGAs, is presented in detail in Section VII. The additional security extension RC2F$_{SEC}$ is introduced in Section VIII, followed by device utilization, vFPGA sizes and performance results of the simulation of our FPGA-Cloud in Section IX. Section X concludes and gives an outlook.

II. Related Work

The provisioning 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. [15]. Initially used mainly on the network infrastructure level, FPGAs are now also employed on the application level of data centers [12]. 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 [16] and increasing security [11] by outsourcing critical parts to attack-safe hardware implementations. In most cases, the FPGAs are not directly usable or configurable by the user, because the devices are, due to a missing provisioning or virtualization, hidden deeply in the data center.

The development of methods for the deployment of FPGA related projects in a cloud infrastructure is performed by Kulanov et al. in [17]. A comparable contribution with stronger focus on the transfer of applications into an FPGA grid for high performance computing is shown in [18]. 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. [19], 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 [20] with additional memory virtualization. A cloud integration model with network-attached FPGAs and multiple user designs on one FPGA was introduced by Weerasinghe et al. [21].

The term *virtualization* itself is used for a wide range of concepts as shown by Vaishnav et al. in [22]. An example for abstractions on the hardware description level is VirtualRC [23], which uses a uniform hardware / software interface to realize communication on different FPGA platforms. BORPH [24] provides a similar approach, employing a homogeneous UNIX interface for hardware and software. The FPGA paravirtualization pvFPGA [25], 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. [26] and Byma et al. [27]. The framework of Byma et al. allows user-specific acceleration cores on the reconfigurable hardware devices, which are accessible via an Ethernet connection. In [28] Chen et al. use FPGAs for processing network streams on virtualized FPGA resources similar to our approach. A virtualized execution runtime for FPGA accelerators in the cloud is shown by Asiatici et al. in [29]. They demonstrated a complete methodology and a resource management framework that allows a dynamic mapping of the FPGA resources in a simple cloud environment.

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 [30], 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 homogeneous bitstreams for different physical vFPGA positions, methods like relocation of designs as shown in [31], provide an opportunity for an efficient context migration of virtualized FPGAs. A preemption of the reconfiguration process itself is shown by Rossi et al. in [32].

The outlined systems virtualize FPGAs and makes them easily available in the cloud. But not every user can utilize such a service, because their sensitive data is at risk in a data center. Security audits are well established in traditional systems, but new cloud environments provide new challenges [33, 34]. In [35] the idea of securing FPGAs in the cloud is outlined, but no prototype realized or protocol described. A secure cloud featuring FPGAs was proposed in [16] relying on a third party, called trusted authority, to establish any trust in the hardware in the cloud. In [36] a simple public key based systems was implemented, however, their protocols fail to protect against, e.g., replay attacks. But none of these proposals virtualizes the FPGA to increase their flexibility and utilization.

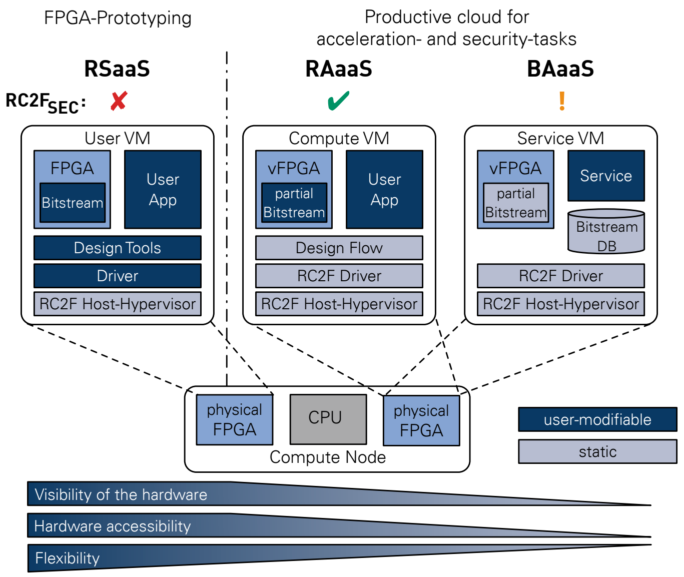


Figure 1. The three service models provided in our cloud environment. In the RSaaS model, users can allocate full physical FPGAs. The RAaaS and BAaaS model allow concurrent user designs on a single physical FPGA.

III. Possibilities and Requirements for FPGAs in the Cloud

The overall motivation is to build a system providing the FPGA for a wide range of service providers with various requirements. The particularity hereby is that we have a data center provider with physical FPGAs, a cloud provider offering a virtual infrastructure and a service provider who offers only a background acceleration, which requires a virtualized FPGA as shown in Section VII. In the following we introduce three key service perspectives as shown in Figure 1. The figure gives also an overview on modifiable and fixed components for each of the service models and shows also the different levels of visibility, accessibility, flexibility and security.

A. *Reconfigurable Silicon as a Service – RSaaS*

This model provides full access to the reconfigurable resource and is primarily intended for a *cloud provider* to develop special acceleration cores without the use of a virtualized FPGA or with a dedicated secured access to the cloud. A cloud provider can allocate a full physical FPGA from the data center operator to implement the hardware of their choice. The FPGA is forwarded and passed through to a VM by the management environment. This model allows developers to reconfigure the full physical FPGA, thus, the RC2F$_{SEC}$ extension cannot protect the users' data. It also opens new attack vectors that do not exist in current cloud environments and so this model should be limited to cloud providers. The concept can be compared to bare-metal cloud services and is related to the traditional cloud service models Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

B. *Reconfigurable Accelerators as a Service – RAaaS*

A model with less freedom for the developer (*service provider*) and typically used by service providers is the Reconfigurable Accelerators as a Service (RAaaS) model. Only vFPGAs of different sizes are visible, allocatable and usable. The model allows the development of hardware designs, which can be used for background acceleration of a specific service and the communication is performed via the framework introduced in Section VII. Such restrictions have the advantage that the RC2F$_{SEC}$ extension can be used, which significantly increases the security of the system compared to the RSaaS model. The RAaaS model can be compared to the PaaS model.

C. *Background Acceleration as a Service – BAaaS*

The third model is suitable for applications and services using background acceleration running in common data centers. The vFPGA is not visible or accessible by the service *users*. Instead, services are using vFPGAs in the background to accelerate specific tasks. The pre-build configuration files and host applications are used by the cloud service provider. Resource allocation and vFPGAs reconfiguration occurs in the background using the RC3E resource management system. Because this model provides concrete service applications to the user, it is similar to the PaaS model. Especially the BAaaS service model demands resource pooling and a rapid elasticity for typical workloads. FPGAs allow a higher flexibility than virtual machines due to faster booting times. From a security perspective, this model is similar to current cloud environments, because of the limited reconfigurability of the FPGA. The RC2F$_{SEC}$ extension cannot be used in this model. In Section IX we demonstrate the cloud's performance with a workload using our background acceleration service model.

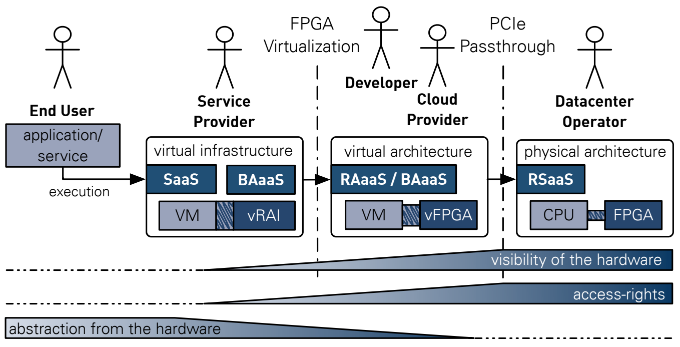


Figure 2. Involved stakeholders and the visibility of resources in a flexible environment. Background acceleration is primarily used in systems were service provider and datacenter operator are the same.

D. *Chaining it all together: RSaaS – RAaaS – BAaaS*

Figure 2 illustrates how a physical FPGA is abstracted by multiple layers into a transparent background accelerator. First, the datacenter operator makes the FPGAs available to the cloud providers (RSaaS). Their developers implement applications for the vFPGAs (RAaaS) and package them in Virtual Reconfigurable Acceleration Images (vRAIs), which are described in Section IV-C. Such a vRAI is used by the service provider as a black box in the BAaaS model (see Section IV-C and Section VII-F). At this point a virtual FPGA infrastructure is provided, which can be used to accelerate services executed by end users. Combined with the classic Software as a Service (SaaS) model, this allows for a seamless integration of vFPGAs to accelerate the service, reduce energy consumption and thus, saving operating costs. At this highest level of abstraction, the FPGA is transparent to the end user.

IV. RECONFIGURABLE COMMON CLOUD COMPUTING ENVIRONMENT – RC3E

In this section, we will present the **R**econfigurable **C**ommon **C**loud **C**omputing **E**nvironment – **RC3E** – and explain the components depicted in Figure 3 in detail. In contrast to other cloud architectures with FPGA integration presented in Section II, the RC3E environment is designed especially for an integration of virtualized FPGA resources and the service models described in Section III. The system is a proof-of-concept to study different approaches for the virtualization and the flexible integration of reconfigurable hardware into a cloud management system. The evaluation results will be used for future integrations of specific RC3E components into a cloud management system such as OpenStack [37].

A. *Overall System Architecture*

The overall system design is a distributed three tier client-server architecture to provide a high degree of scalability and flexibility. RC3E offers three access possibilities to use and administer the RC3E system. The most common way is a *login shell* either on a local computer with our RC3E client or via secure shell login to the remote login server. Additionally, it is possible to connect a web frontend (see Section IV-B) to the core system's API.

The RC3E core system running on the management node, which itself is a three tier architecture, orchestrating the connected clients and all registered compute nodes. It uses

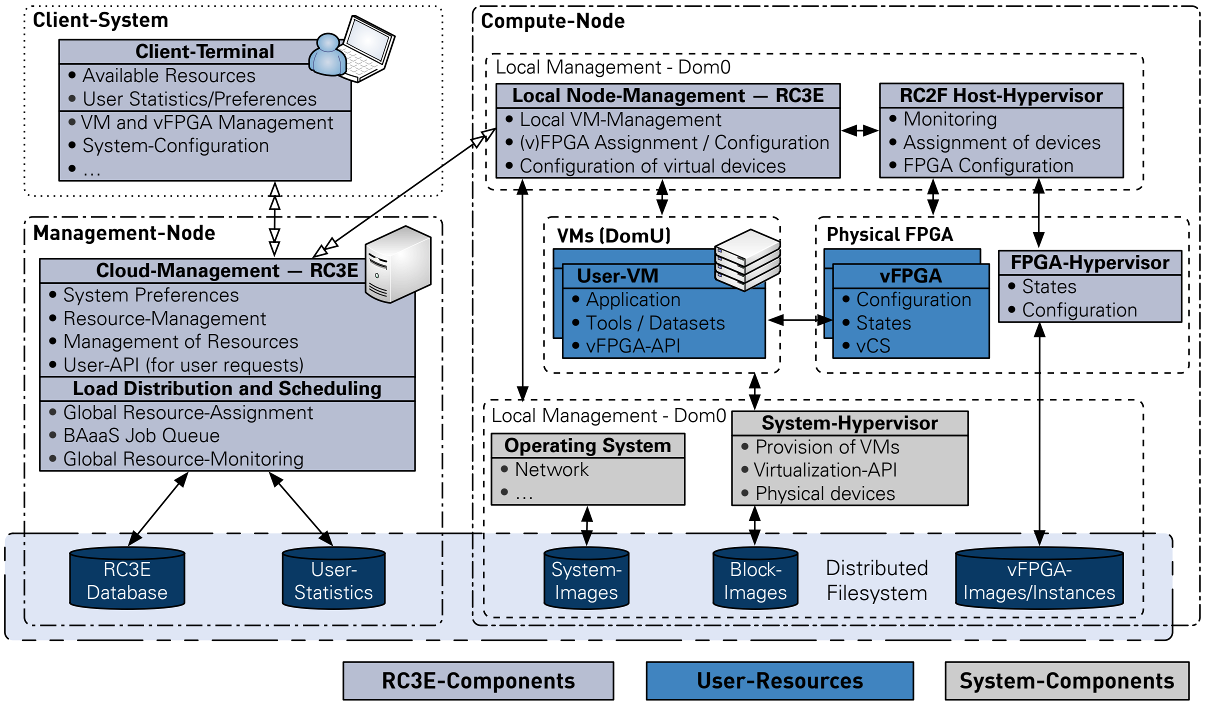


Figure 3. Architecture of the resource management and hypervisor RC3E consisting of core system (management, monitoring and job scheduling) and compute node providing VMs and vFPGAs.

a centralized database to store all required information and manages a distributed file system, which is shared between all compute nodes and the management nodes.

B. Web-based User Interaction and Database Backend

In a cloud environment it is common that the majority of users does not have administrative access to the system. A web-based frontend allows these users a fast and comfortable way to reserve FPGA slices on the server, upload and run their designs. The `Django` framework was used as a foundation of such a web frontend. All data required for the frontend is stored in a `MySQL` database, which interacts directly with `Django`'s web-server. The *RC3E* tools discussed in the other sections have to be present on the same machine as the web-UI in current implementations. Separating the cloud systems control instance from the web-server is desirable and expected to be done with a reasonable effort in the future.

In Figure 5 the modeled entities and their reference relations are shown. Data types and classes provided by `Django` itself are printed in italics for distinction. Attributes that only serve framework-internal purposes and are added by `Django` automatically have been omitted for clarity.

The models focus points and their synergies will be examined in the following. Words in bold typeface thereby correspond with the entities in the model.

To avoid ambiguity, the term *user* refers to any human interacting with the system, while *administrators* refers to users with the privileges to access and modify the system's internal state. Persons without such privileges will be called *consumers*. `Django`'s integrated user group and permission management system is used to reflect user categories and facilitate access control across the web-UI.

It was decided to use a fine-grained modeling approach to retain flexibility and changeability in an attempt to create a future-proof software base for future development iterations. This also includes the avoidance of unmanaged data redundancy by preferred usage of foreign references.

Entities are represented on the database level as separate tables with each of the entity's attributes as a table column and each instance of the entity as a table row, containing the actual attribute values within the respective cells.

The modeling is heavily influenced by the operation principles of foreign key references in SQL-based data storage systems. Thus, in situations where two entities *A* and *B* form a *[A]1:n[B]*-relationship, the foreign key has been placed on the *B* side referencing *A* to avoid creating a separate associative entity each time such a relation shows up. In *[A]m:n[B]* scenarios such a separate entity can not be avoided though. The UML-style *B contains A* symbolization should therefore be read as *B contains a reference to A*. Foreign key references are set up to execute cascading deletions and modifications, since the entity containing the reference would enter an invalid state if not deleted/modified as well.

1) Nodes, FPGAs and Regions: A **node** represents the physical cloud server in which **FPGA**s are installed. Each node is named and identified by an unique IP-address. An optional comment allows the node's administrator to easily convey additional information to the consumer aside from the installed FPGAs.

Installed FPGAs are initialized by the *RC3E* management system, which provides **PCI-addresses** for the node and the device itself. The latter are queried during the registration of the FPGA with the web-frontend and associated database entries are created automatically.

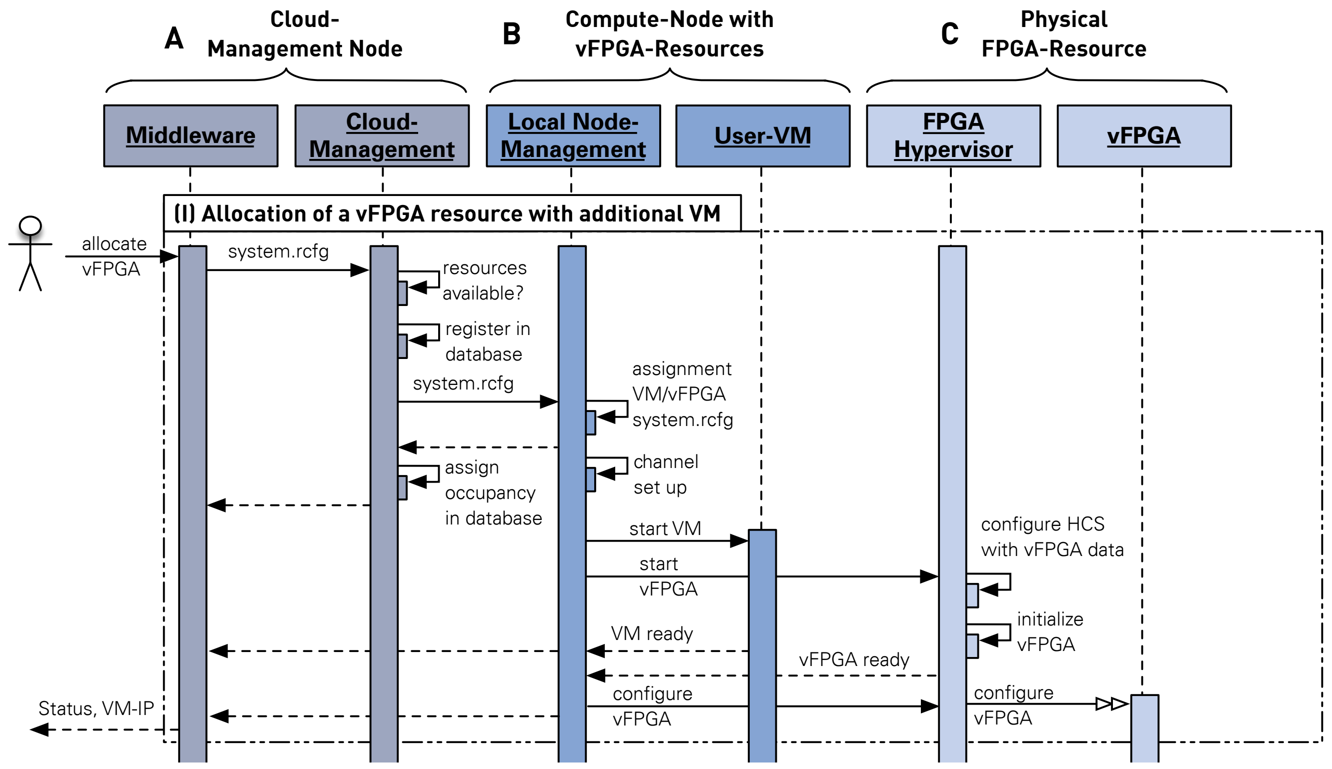


Figure 4. Sequence diagram showing interaction of levels in the RC3E system via cloud management nodes, the compute node with a (free) vFPGA resource up to the physical FPGA for three exemplary scenarios.

To offer a uniform description of an FPGA, the **FPGA model** has been introduced as an abstraction of generic and structural information. The **producer** entity has been externalized with the prospect of providing additional information about it in future implementations. An FPGA model references a **region type** and holds the amount of regions an FPGA of this model has. With this approach only homogeneous FPGA architectures can be modeled. For heterogeneous architectures the region type and -count would have to be externalized and act as associative entity between the FPGA model and the region type.

One or multiple **region** instances are created alongside with an FPGA instance, the amount depending on the FPGAs region count. Its region type is determined by the associated FPGA model and an region index is determined. These indices are unique per FPGA, 0-based and continuous, with the purpose of identifying regions on an FPGA and determine whether they are neighbors and can therefore be reserved together. For programming purposes the *RC3E*-system provides a file path to a memory device, which is also stored.

2) Reservations and Virtual FPGAs: The most common interactions of customers with the system are the reservation of vFPGAs and the programming of such. For the first step, the customer provides points in time for the start and end of the reservation period and selects a region type suitable for his use case along with the required amount of consecutive regions. The database-backend will then be queried for matching FPGA regions. Already existing reservations are taken into account when selecting a sufficient amount of consecutive FPGA regions to reserve. On success, a new vFPGA instance is created, alongside with a **region reservation** for each affected region. The latter is an association entity to facilitate the *[vFPGA]n:m[Region]*-relation. While region reservation database entries are removed after the reservation period has passed, vFPGA entries are retained for bookkeeping purposes.

3) Programming the Virtual FPGA: The administrators provide information about the installed **programmer**s and the programming **script** for the available FPGA models. Both entities are used to determine which programmer-FPGA model-combinations are supported and thus, which programmers are offered to the customer for usage with his reserved vFPGA. Upon programming, the script's template gets parsed and placeholders within it matched against the available **device variables** and **runtime variables**. If a match occurs, it is replaced by the variables' value. Device variables are bound to specific FPGAs and are set by the administrator while runtime variables may be `python` expressions or fixed values and will be evaluated at the point of replacement. Within the reservation period the user may upload a bitfile, which will then be passed on to the programmer alongside with the appropriate script and variables. In case of a reservation spanning multiple regions, the memory device path of the region with the lowest index is the one used by the programmer for the whole reserved section. Uploaded bitfiles are currently not stored in the database backend.

C. Description of vFPGAs (RCFG and vRAI)

All necessary information for the execution of a background accelerator is combined in a so called vRAI. The vRAI can be delivered as fully encapsulated accelerators to

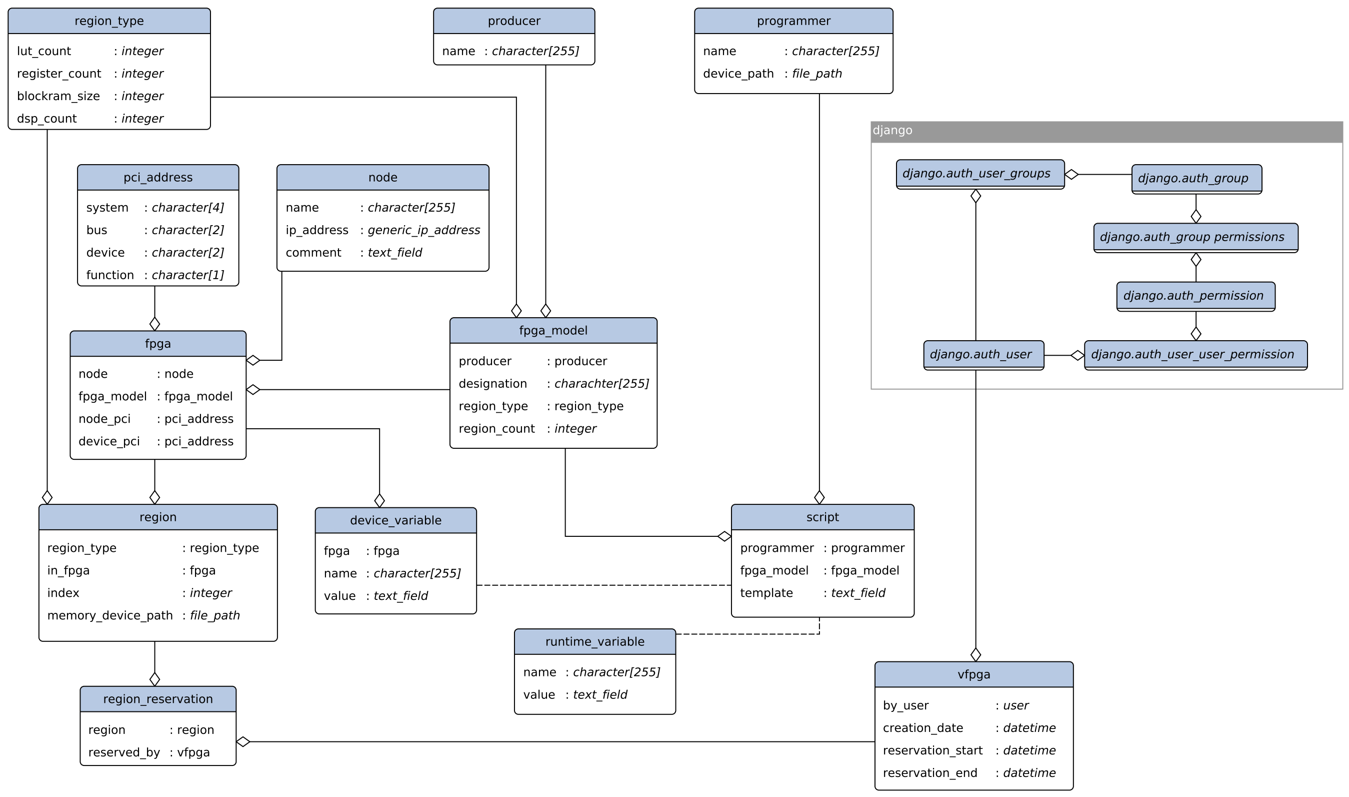


Figure 5. Overview over the entities involved in modeling a database backend. Italic text represents primitives provided by the `Django` framework.

the higher-level cloud service developers. From the point of view of the provider of a service, there is the requirement to process a request in the form of a function call as compact, safe or energy-efficient as possible, without having any knowledge of the physical hardware in the background. The execution of a vRAI requires allocation of a vFPGA, which fulfills all requirements described in the **R**econfigurable (Device) **Config**uration (RCFG). In order to allocate and execute an accelerator from a VM, several components are required within the vRAI package (see Figure 6):

- The required vFPGA-Images for all possible vFPGA-Slots (necessary for a migration of a vFPGA-Instance).
- The RCFG file describing the required vFPGA suitable for the vFPGA image.
- The host application for initialization and interaction with the vFPGA-Instance, which is embedded directly into the user's offloading service (BAaaS).
- Virtual Context Bit Mask (VCBM) to read the relevant bits within the vFPGA instances that identify the current state (see Section VII-E).

Since different RCFGs are required depending on the different service models, these are outlined below and explained accordingly. Figure 7 shows an exemplary RCFG, which describes a complete physical FPGA in the model RSaaS `service='rs'` with the name `name='fpga0'` gets and in the VM instance `vm='vm1-hvm'` via hardware virtualization and PCI passthrough at a certain address in the PCI tree `pci ='01:00.0'` is displayed. The virtual network address is sent to the system via `vif='10.0.0.43'`. The VM must have its own configuration file depending on the virtualization, and the embedding of the VM configuration in the RCFG for the FPGA is also possible. Since different FPGAs are to be provided in this model, there is a corresponding entry with the name of the FPGA board `board='vc707'`. The configuration of the FPGA is done using the JTAG interface `config= 'jtag'`, where an initial design is additionally specified: `design ='led.bit'`. Using a RCFG file, the RSaaS model can allocate only one FPGA and its associated VM, otherwise the requested system may become too complex and it may not necessarily be mapped to the physical hardware resources.

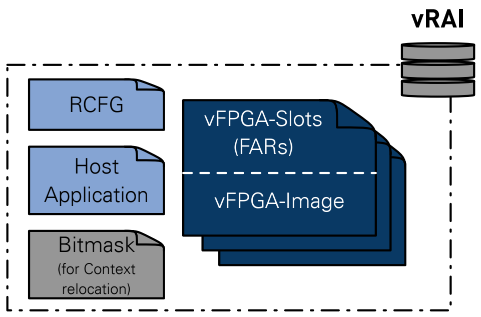


Figure 6. Virtual Reconfigurable Acceleration Image (vRAI) package with all the files required to run a vFPGA-Instance, such as vFPGA-Image (partial bitstreams), RCFG, host program and optional bitstream masks for the bitstream relocation.

```
service = 'rs'                    #Service Model RSaaS
name = 'fpga0'                    #FPGA-Instance Name
vm = 'vm1-hvm'                    #VM-Instance Name

board = 'vc707'                   #FPGA-Board
vif = 'ip=10.0.0.43'              #FPGA-IP
vpci = '01:00.0'                  #PCI Node in VM-Instance
design = 'led.bit'                #Initial Design
config = 'jtag'                   #Configuration Method
```

Figure 7. Configuration file for the allocation of a physical FPGA in the Service Model RSaaS.

More complex is the description of the vFPGAs in the model RAaaS `service = ra` as shown in Figure 8. In addition to the already known parameters, in this example two vFPGAs with different number of vFPGA slots `size = [2, 1]` are allocated via `vfpga = [2]`, where both are passed to the same VM `vm = ['vm1-pvm']`. The number of frontend interfaces is determined by `frontends=[2, 1]` for each vFPGA, where the number must be less than or equal to the number of vFPGA slots. At this point, the cloud management must try to map the desired virtual system to a physical system, where in the model RAaaS additionally the position of the vFPGA on the physical FPGA can be specified by the field `loc = [0,2]`. Via `debug = ['csp']` in the resource management model RAaaS it is communicated which debug / tracing interface is to be additionally instantiated. The capacity of the external DDR memory can also be specified (`memory = [2000,1000]`), as well as the desired state of vFPGAs `boot = ['paused']`. The location of the values within the lists, such as `size`, `loc`, `vif`, or `design`, decides how to map the entries. Furthermore, if there is only one entry in a list, such as `key`, it will be applied equally to all vFPGAs. If no clear assignment is possible or if this is not permitted, an error message is output.

```
service = 'ra'                    #Service Model RAaaS
name = ['vfpga-bsmc']             #vFPGA/User Design Name
vm = ['vm1-pvm']                  #VM-Instance Name

vfpga = [2]                       #Number of vFPGAs
size = [2, 1]                     #vFPGA-Slots
frontends = [2, 1]                #Frontend-Interfaces
loc = [0,2]                       #vFPGA-Slot on device
memory = [2000,1000]              #DDR-Memory Size in MByte
vif = ['ip=10.0.0.42', ...]       #vFPGA-IPs

boot= ['paused']                  #Initial vFPGA-State
design = ['bsmc-2.bit', ...]      #Initial Designs
```

Figure 8. Configuration file for the allocation of a vFPGA-Cluster in the Service Model RAaaS.

In the model BAaaS, the actual user has no knowledge of the vFPGA resources. The RCFG file, which is stored together with all the required vFPGA images in the vRAI, is reduced. For example, as shown in Figure 9, there are no locations of the vFPGA slots (`loc`) or information about the debug / tracing interface (`debug`) required. The RCFGs must be checked by the resource manager for the rights of the users within the service model before the global allocation of the appropriate resource is first performed and assigned to the user. The concrete processing of the content then happens within the Dom0 of the assigned node as introduced in Section IV-B1.

```
service = 'ba'                    #Service Model BAaaS
name = ['vfpga-kmeans']           #vFPGA/User Design Name
vm = ['vm1-pvm']                  #VM-Instance Name

vfpga = [1]                       #Number of vFPGAs
size = [4]                        #vFPGA-Slots
frontends = [2]                   #Frontend-Interfaces
memory = [4000]                   #DDR-Memory Size
vif = ['ip=10.0.0.151']           #vFPGA-IP

key = ['AAAABC1yc2 ... BuHNE']    #User AES-Key
boot= ['booting']                 #Initial vFPGA-State
design = ['kmeans-quad.vrai']     #Initial Design
```

Figure 9. Configuration file for the allocation of a single vFPGA in the service model BAaaS

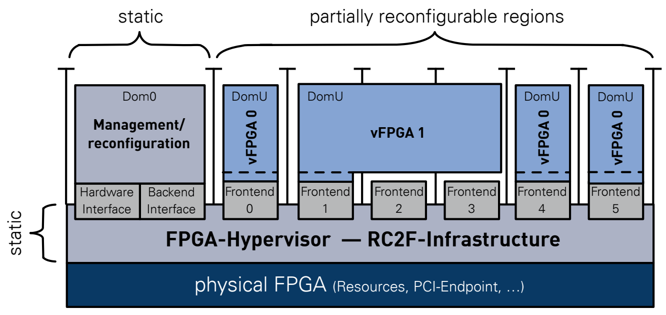


Figure 10. Paravirtualization concept used in RC2F to provide virtual FPGAs (vFPGAs) using partial reconfiguration. vFPGAs can be combined to group larger regions and thereby provide more resources.

V. Definitions for FPGA virtualization in the cloud-context

In order to establish a common name for the following chapters with regard to the virtualized FPGAs, the necessary terms are defined in order to better distinguish the vFPGAs (see Definition 1) according to their life cycle based on the requirements analysis in this chapter to be able to. The terms are based on those of system virtualization after [38].

Definition 1: vFPGA *A virtual FPGA (vFPGA) is located within a physical FPGA on one or more vFPGA slots (see Definition 2). A vFPGA is perceived by the user as a stand-alone resource with a dynamic number of hardware resources (slices, LUTs, registers, etc.).*

Definition 2: vFPGA-Slots *A vFPGA is mapped to individual physical regions with a fixed number of hardware resources, and thus fixed size within the physical FPGA, called vFPGA-Slots.*

Definition 3: vFPGA-Design *The vFPGA-Design is the hardware design / the user's hardware design, which is placed and wired from a netlist (RTL level) within a vFPGA with its frontend interfaces.*

Definition 4: vFPGA-Instance *A vFPGA-Image (see Definition 5) within a vFPGAs that is directly associated with a user and can contain user-specific data (context) is called a vFPGA-Instance and can be detached from vFPGA-Slots (see Definition 2).*

Definition 5: vFPGA-Image *A partial bitstream, which forms the basis for a vFPGA-Instance, is called vFPGA-Image, the specific vFPGA slots is assigned.*

In addition to the definitions just made, which relate to the specific vFPGAs and their life cycle, furthermore, the different hypervisors in the overall system are to be differentiated and defined. The term *hypervisor*, is defined as a system for managing and allocating guest-to-host resources, forms the basis for the following definitions.

Definition 6: System-Hypervisor *The System-Hypervisor corresponds to the classic hypervisor (VMM), which provides the VMs within the system virtualization on the host system.*

Definition 7: RC2F Host-Hypervisor *The management structure for the vFPGAs on their host system is called RC2F Host-Hypervisor, or just Host-Hypervisor.*

Definition 8: FPGA-Hypervisor *The FPGA-Hypervisor is the management structure on the FPGA that monitors the accesses of the vFPGAs within the physical FPGA.*

VI. FPGA Virtualization

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 Environment

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 [39]. This kind of virtualization is designed for 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 [9], 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 [6] and [1]. An extraction of internal DSP registers is not supported in recent Xilinx FPGAs and must be considered in the design.

One of the first virtualized systems was the IBM Virtual Machine Facility/370 (VM/370) [40] 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.*" [38, p. 3]

The two classic approaches are either the use of a 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 [41]:

- 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 [38]:

- 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 introduced in Section VII-C. The unprivileged guests (DomU) run on a hypervisor, which forwards calls from frontend driver to the backend driver of the management VM (Dom0).

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 10 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 [14] and static interfaces. The vFPGAs can have different sizes (Figure 10) 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.

VII. FPGA Prototype RC2F

Our prototype RC2F introduced in [4] 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.

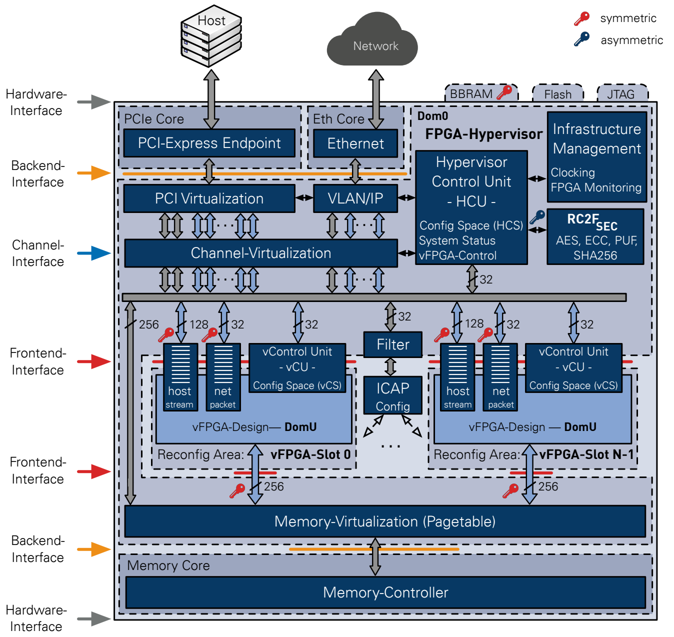


Figure 11. Virtualization frame RC2F with hypervisor, I/O components and partial reconfigurable areas housing the vFPGAs. The vFPGAs have access to the host using PCIe (FIFO interface and config space), to the Cloud network using Ethernet and the virtualized DDR3 memory.

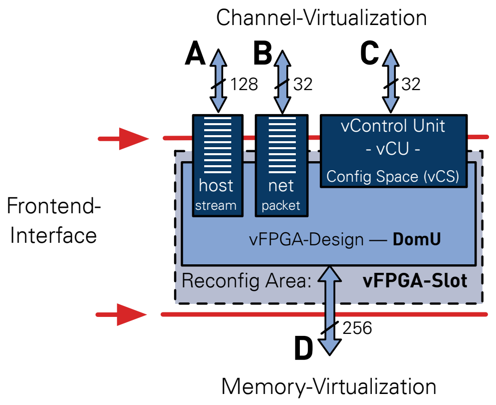


Figure 12. Architecture of a RC2F-vFPGA with (C) the local vFPGA Control Unit (vCU) containing the Virtual Control Space (vCS). The data lines for the memory interface (D) and the two FIFOs are each available for input (A) and output (B) streams.

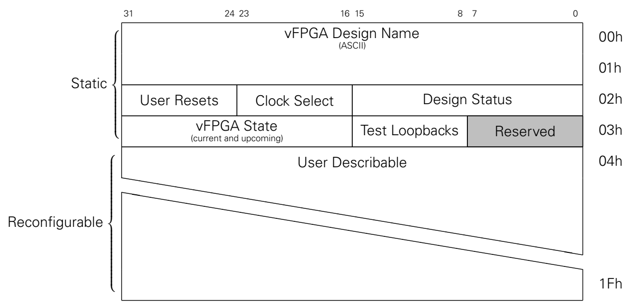


Figure 13. Register and memory interface for the management of vFPGAs accessible by the user VM (rc2f_cs).

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 10. Our architecture, designed to provide the vFPGAs, is shown in Figure 11. 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 [42]), 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 [6]. The number of frontends and their locations are defined by the physical FPGA architecture as shown in Figure 16. The *Hypervisor Control Unit* manages the ICAP controller and the *vControl* units, which maintain and monitor the vFPGAs.

1) 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 13. 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.

2) Components of the RC2F infrastructure: The RC2F infrastructure is exemplarily implemented within the static area with the components as shown in Figure 11. For the infrastructure, as shown in Figure 16, both the right side of the physical FPGA and the lower clock region are provided. The constant components of the static infrastructure within the RC2F infrastructure are:

FPGA-Hypervisor: At the heart of the implementation is the FPGA hypervisor, which provides the frontends to the vFPGAs as shown in Figure 11. Essential components are the configuration memory of the FPGA hypervisor explained in Figure 14, which transmits all control commands and signals to the FPGA, and the ICAP controller for reconfiguring the vFPGA slots and to read out a partially reconfigurable vFPGA instance for migration. The memories are built from components of the Pile of Cores (PoC) library [43] and constructed as shown in Figure 14. The internal clock rate (system clock) of the FPGA hypervisor and device virtualization is 250 MHz. To decouple the FPGA hypervisor from the internal logic of the vFPGAs as well as the I/O components, there are cross-clocking FIFOs at the interfaces between the clock domains.

PCIe-Controller: The PCIe controller is Xilinx's provided Intellectual Property Core (IP-Core) *7 Series FPGAs Integrated Block for PCI Express v3.3* [44] with a Xillybus controller [42], which provides both FIFO and memory interfaces on the FPGA, as well as a driver within the host hypervisor.

DDR3-Controller and Memory-Virtualization: The used DDR3 controller is the IP Core *Xilinx MIG V1.4* [45], which is the hardware endpoint to the backend interface as introduced in Section VI-B and illustrated in Figure 11. The resulting storage virtualization managed by the host hypervisor organizes the specific translation from the virtual to the physical addresses, thus, separating the user areas in the memory from each other.

Ethernet-Controller: The Ethernet controller used is based on the IP Core *LogiCORE IP Tri-Mode Ethernet MAC v5.2* [46], which is an interface on the Media-Access-Control (MAC) layer of Open Systems Interconnection (OSI) Reference Model [47] offers. Based on this, parts of the PoC library [43] are used to implement the interfaces to the vFPGAs.

In addition to the previously discussed components of the RC2F infrastructure, additional components are required whose hardware resources depend on the number of physical vFPGA-Slots. These components are also in the static region:

Device-Virtualization: Device-Virtualization provides the concurrent communication channels for the vFPGAs. The realization of the PCIe-Virtualization is done by means of the Xillybus-Controller [42] provided components. The provided FIFOs are passed on to the vFPGAs and decoupled (cross-clocking) to allow different clock domains for the FPGA hypervisor and the vFPGA design. Memory virtualization requires one page table per user. The prototypical implementation uses page sizes of 8 MByte.

vFPGA-Frontends: The frontends are implemented as outlined in Figure 12 and Figure 15. The configuration memories are constructed according to Figure 13 and consist on the one hand of a part located in the static area of the FPGA and on the other hand of a user area, which can be used freely. In addition to the stores, the states of each vFPGAs are managed as outlined in Section VII-E.

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 14, which includes 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.

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 is shown in Figure 15. The frontend FIFOs and the FPGA memories are mapped to device files inside the host hypervisor. There, the system forwards the user devices to the assigned VM using inter-domain communication based on vChan from Zhang et al. [48] in our Xen virtualized environment, similar to the FPGA device virtualization pvFPGA [25].

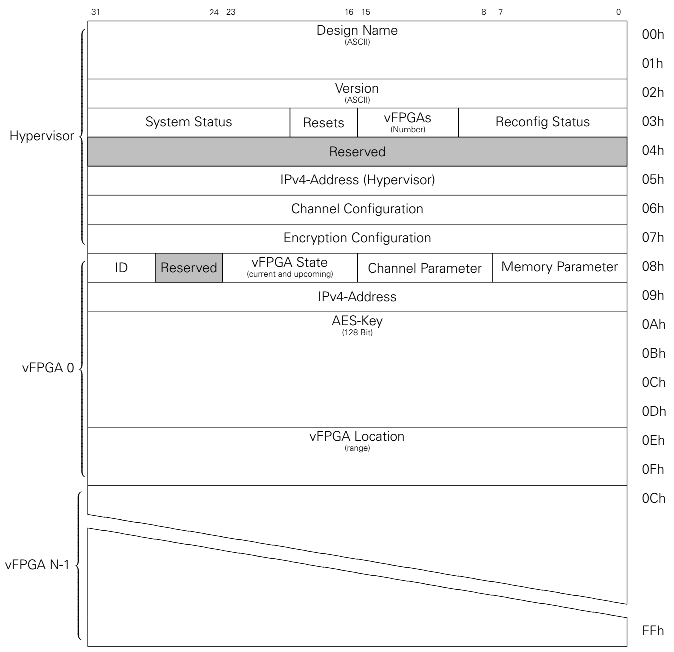


Figure 14. Register and memory interface for the management of the FPGA hypervisor accessible by the host hypervisor (rc2f_gcs).

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.

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 10. Among the vFPGAs, the partition pins (PP) between the static and the reconfigurable regions are placed with identical column offset as shown in Figure 16. 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 homogeneous structures. Thus, we establish homogeneous vFPGAs, which are identical among each other by excluding these areas in all vFPGAs as shown in Figure 16. 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. Figure 17 shows the breakdown of the FPGA resources to the three different areas: static infrastructure, partial reconfigurable vFPGAs and unusable due to the homogeneity.

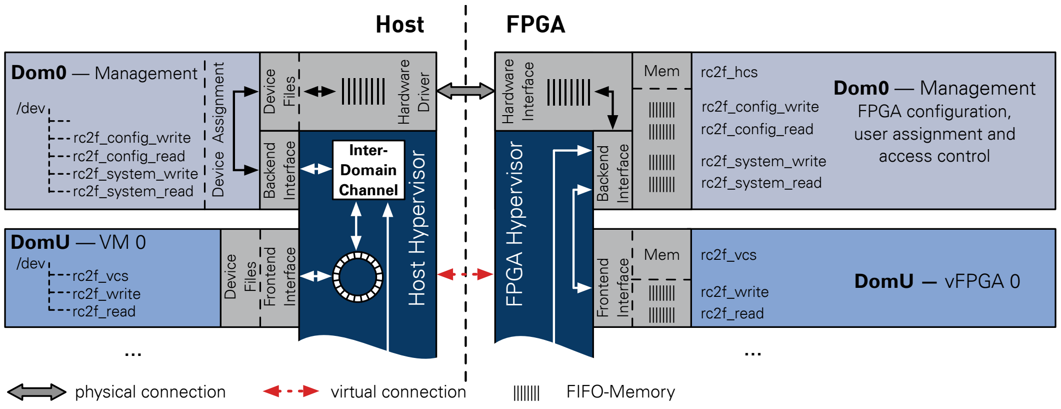


Figure 15. System architecture on the hypervisor level of the host system. FIFOs (rc2f_write, rc2f_read) and configuration memories (rc2f_cs) are displayed in the different host memories.

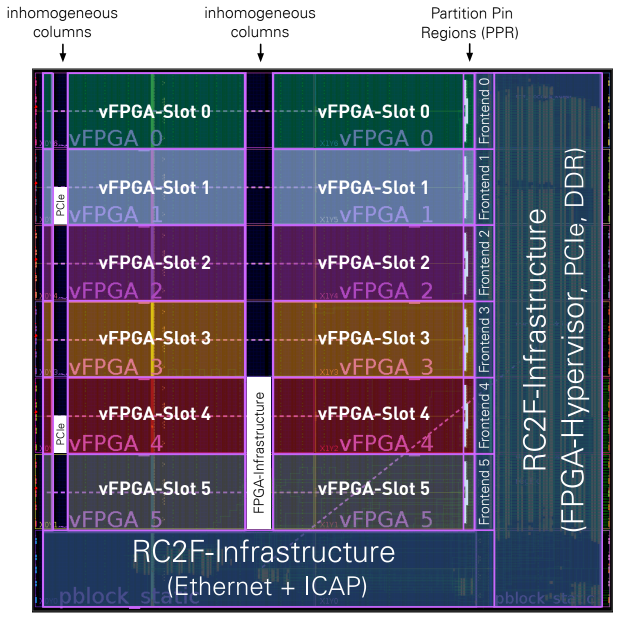


Figure 16. Layout of a Xilinx Virtex-7 XC7VX485T with six vFPGA regions configurable using dynamic partial reconfiguration. The regions and their number are determined by the height of the configuration frames, which consist of one complete column inside a clock region. Regions are homogeneous to allow migration of vFPGAs.

E. 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 system [3] as well as additional on-chip state transitions. Figure 18 gives an overview of these two parts in our FPGA virtualization. A state with a control flow transition between host and FPGA is called *transition state*.

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 hosts and FPGA devices. In the following, the most important states on host and FPGA – as introduced in Figure 18 – are 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 the 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 Figure 19) 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 VII-F2.

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 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.

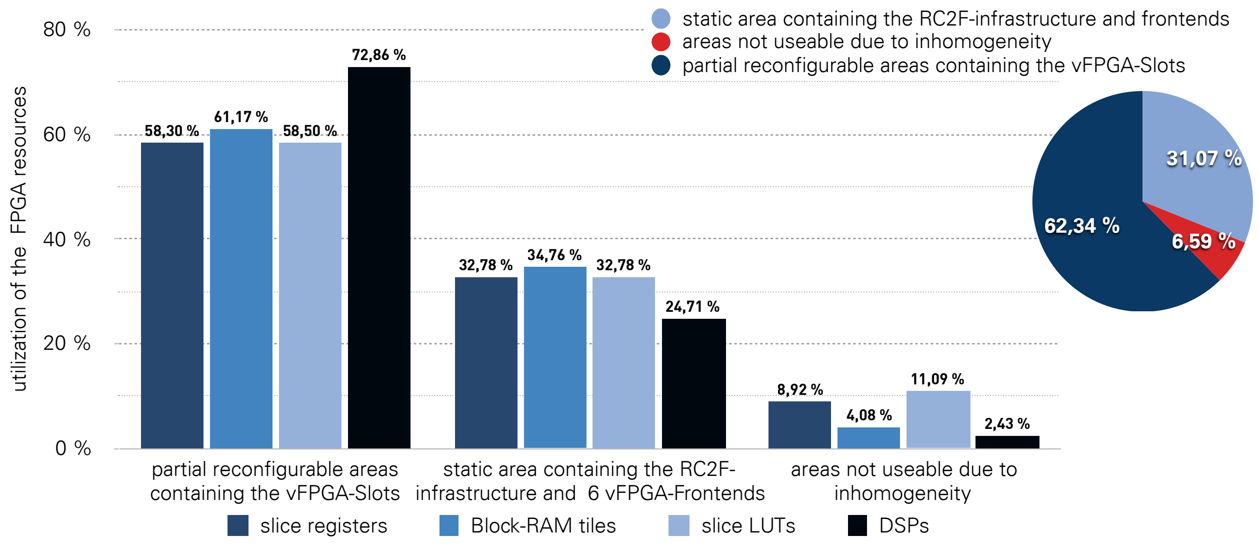


Figure 17. FPGA resources of the three different regions of a Xilinx Virtex-7 XC7VX485T with six vFPGA-Slots, the static region with the FPGA hypervisor and the unusable regions due to homogenization.

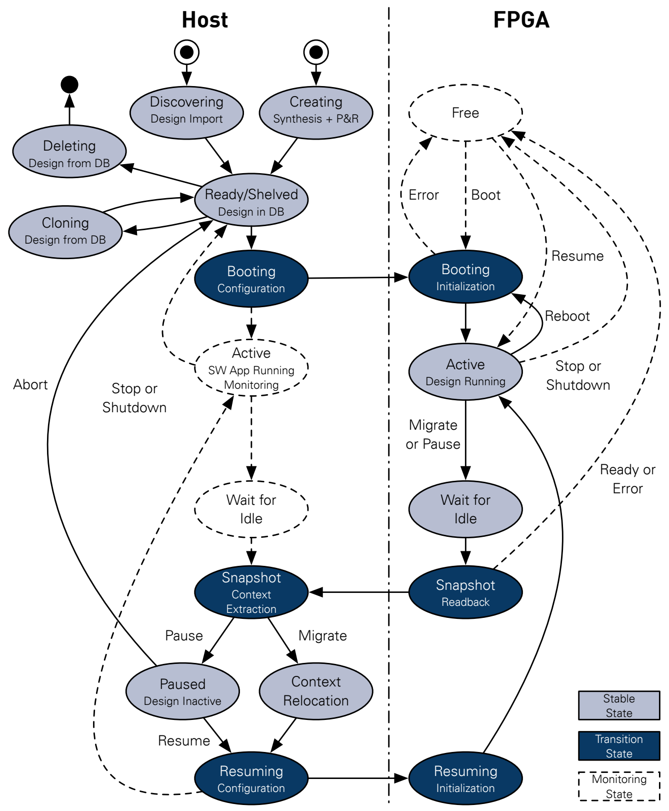


Figure 18. State transitions of a vFPGA (on the host and FPGA).

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.

F. *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 Figure 19. In the following, the components, all additional design flow steps and the generated metadata are described in detail.

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 III 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 [6]. 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 vRAI as shown in Figure 19.

1) vFPGAs Bitstream Generation and Boot: In the initial step, the full bitstream containing the static design is generated with the traditional Xilinx flow as shown in Figure 19. 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.

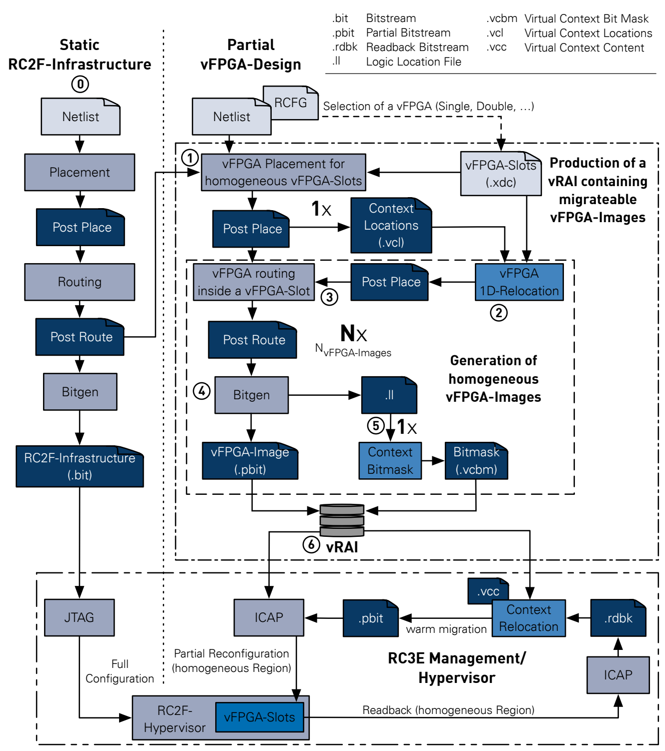

.bit Bitstream
.pbit Partial Bitstream
.rdbk Readback Bitstream
.ll Logic Location File
.vcbm Virtual Context Bit Mask
.vcl Virtual Context Locations
.vcc Virtual Context Content
Static RC2F-Infrastructure
⓪
Partial vFPGA-Design
Netlist
Netlist
RCFG
Selection of a vFPGA (Single, Double, ...)
Placement
①
vFPGA Placement for homogeneous vFPGA-Slots
vFPGA-Slots (.xdc)
Production of a vRAI containing migrateable vFPGA-Images
Post Place
Post Place
1x
Context Locations (.vcl)
Routing
vFPGA routing inside a vFPGA-Slot
③
Post Place
vFPGA 1D-Relocation
②
Post Route
Post Route
Nx
N vFPGA-Images
Generation of homogeneous vFPGA-Images
Bitgen
④
Bitgen
.ll
⑤
1x
RC2F-Infrastructure (.bit)
vFPGA-Image (.pbit)
Context Bitmask
Bitmask (.vcbm)
⑥ vRAI
JTAG
ICAP
.pbit
warm migration
.vcc
Context Relocation
.rdbk
Full Configuration
Partial Reconfiguration (homogeneous Region)
RC3E Management/ Hypervisor
ICAP
RC2F-Hypervisor
vFPGA-Slots
Readback (homogeneous Region)

Figure 19. Extended design flow generating partial vFPGA bitstreams and the additional metadata (Beginning with the vFPGA Placement ①).

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 overall runtime will be reduced in the future by using relocation of placed vFPGAs using homogeneous regions [49]. 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 context content* file (`.vcc`) as shown in Figure 19. The information required is extracted from the additional *Logic Location* files (`.ll`) and the *Xilinx Design Language* files (`.xdl`), which are generated during the design flow. 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.

In case of a *pause* or *migration* command, the FPGA is stopped as explained in Section VII-E. 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 `.vcc` file. By the use of the location 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.

2) 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 initialization 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.

VIII. RC2F$_{\text{SEC}}$ EXTENSION

Security is now more important than ever. Therefore our RC2F$_{\text{SEC}}$ extension provides a high level of security for client's data and algorithms. To achieve this, we propose a novel combination of existing security features, a subset of the Transport Layer Security (TLS) protocol and a filter for the partial bitstreams. However, due to various degrees of flexibility show in Figure 1, the extension is only fully available to the production-ready service model RAaaS. But with changes to current FPGA architectures, which will be described later, it would also be available in the BAaaS service model.

A. Security Model

Various adversaries challenge the system's security through multiple vectors. But before these challenges are formalized as requirements for the design, a few assumptions have to be made.

Assumption §A1: *The selected cryptographic algorithms cannot be computationally broken by state-of-the-art attackers. Encrypted data cannot be decrypted or messages signed without access to the keys.*

Assumption §A2: *Naive implementations of cryptographic algorithms are susceptible to side channel attacks, but hardened implementations can withstand better and protect the keys, both shown in [50]. Providing such implementations is not within the scope of this paper. Hence, it is assumed that any cryptographic keys and sensitive intermediate values are secure inside the chip.*

Assumption §A3: *The client's workplace can be trusted and is inaccessible to an attacker.*

Assumption §A4: *The FPGA vendor can be trusted and tries to detect backdoors introduced by manufacturers, tools suppliers or IP vendors, e.g., through analyzing the hardware to find unwanted modifications as shown in [51, 52]. This is the same level of trust the client has to have into hardware in general: CPUs, hard drives and other components might be modified as well.*

Assumption §A5: *Denial of service attacks, interruptions or even physical destruction are secondary and more a concern of the providers, because quality-of-service is an important business factor. The security of data and algorithms has the highest priority.*

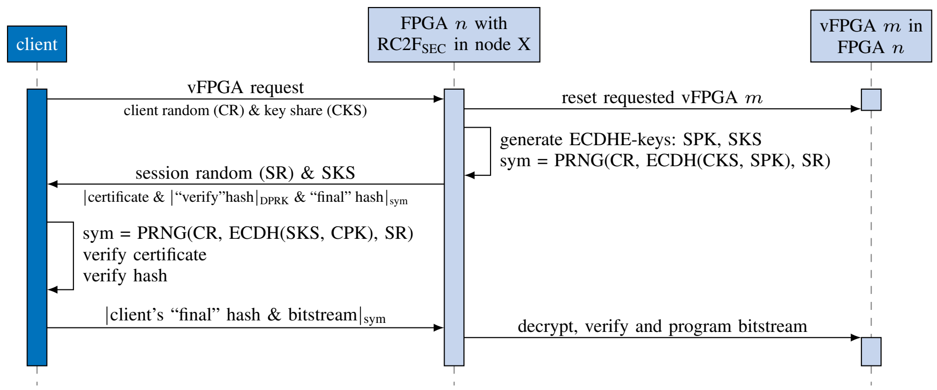

client
FPGA n with RC2F$_{SEC}$ in node X
vFPGA m in FPGA n
vFPGA request
client random (CR) & key share (CKS)
reset requested vFPGA m
generate ECDHE-keys: SPK, SKS
sym = PRNG(CR, ECDH(CKS, SPK), SR)
session random (SR) & SKS
|certificate & |"verify"hash|$_{DPRK}$ & "final" hash|$_{sym}$
sym = PRNG(CR, ECDH(SKS, CPK), SR)
verify certificate
verify hash
|client's "final" hash & bitstream|$_{sym}$
decrypt, verify and program bitstream

Figure 20. The TLS protocol was adapted to enable the secure and authentic transfers of vFPGA bitstreams. The configuration is protected by symmetric encryption with the key "sym", which is created during the TLS handshake.

Based on theses assumptions, the requirements for the system's design can be defined.

Requirement §R1: *A dedicated third party or trusted authority offering special services only for this system should be avoided.*

Requirement §R2: *A client must be able to establish an authenticated and secured connection from a trusted workplace to the system.*

Requirement §R3: *The FPGA cannot rely on software running on the host machine. The untrusted IaaS provider has direct access and can manipulate anything but the chip.*

Requirement §R4: *The allocation of a vFPGA by an attacker should not interfere with a legitimate client. A strict separation of clients' data is mandatory and the reconfigurable partitions have to be isolated to prevent any interference.*

B. Design

The RC2F$_{SEC}$ extension has to provide two fundamental capabilities:

- Authentication: The FPGA can prove its genuineness to a client.
- Confidentiality: Tamper-proof and secure data transfer.

Microsemi FPGAs already offer similar features, but they do not allow partial reconfiguration preventing virtualization and thus, an efficient cloud deployment. Furthermore, their reputation got a big hit when a backdoor was discovered, which reveals sensitive private keys [51]. The impact of this backdoor could have been minimized, if a more sophisticated protocol offering perfect forward secrecy would have been used. Thus, the RC2F$_{SEC}$ extension implements the well established and thoroughly researched TLS protocol for bitstream and data transfers.

Hence, an embedded TLS processor is required, which is implemented as part of the RC2F$_{SEC}$ extension shown in Figure 11. It does not feature all possible algorithms due to resource constraints, only efficient primitives in terms of performance per logic gate were selected. The initial key exchange follows the Diffie-Hellman algorithm using elliptic curves (ECDH). An advantage over RSA is the cheap generation of new key pairs, making perfect forward secrecy available. Thus, every connection is encrypted with a unique key, compromising one does not affect other connections to the same FPGA. The device's permanent private key is only used to authenticate it through the elliptic curve digital signature algorithm (ECDSA). It reuses the elliptic curve primitives, saving a significant number of resources. Resources can also be saved by combining encryption with authentication. AES128-GCM is an authenticated encryption scheme with high performance hardware implementation, which are also available for CPUs. Finally, SHA256 computes hashes during the handshake.

C. Transfer Protocol

Figure 20 shows the handshake procedure. The client initiated it with a vFPGA request, a key share (CKS) and some random data (CR). The unencrypted request can be used by the cloud provider for billing and scheduling. If the request is unjustly blocked by a cloud provider or evicts another legitimate client, only their quality of service suffers, but not the security of clients' data. Through a complete reset of the vFPGA previous configurations are no longer accessible, and even if data remains in buffers or external memory it is still encrypted.

After the vFPGA reset, a TRNG, which is part of the RC2F$_{SEC}$ extension, generates the session random (SR) and an ephemeral session private key (SPK). With the SPK the public session key share (SKS) is calculated. This new key pair is used to complete the ECDHE key exchange. The resulting shared secret is along with the CR and SR feed into a well defined PRNG. Its output is used to derive various symmetric keys and nonces ("sym"), which are right away utilized to encrypt the FPGA's certificate. Additionally, a hash over the transaction so far is calculated, signed through ECDSA with the device's private key, encrypted and then appended to the certificate. Finally, a second hash over the whole handshake including the CR, CKS, SR, SKS, the encrypted certificate and first hash is calculated, then encrypted and the package is send to the client. Upon receiving it, the unencrypted SR and SKS are used in the same way to derive the symmetric keys and nonces ("sym") through ECDHE and the PRNG. With them, the rest can be decrypted, the certificate and public key verified and the hashes checked. At last, the client also calculates a hash over the whole transaction, now including the second hash, and prepends it to the bitstream. Together they are encrypted with "sym" and transferred to the FPGA. There, after the hash was compared to a locally computed one, the vFPGA bitstream is programmed and the partition ready for use.

If any errors occur or the client uses standardized but not implemented functionalities, the handshake aborts, resets and returns the system into a safe state in which it accepts new connections.

D. Configuration Filter

The configuration filter protects the RC2F as well as other clients' vFPGAs from unwanted modifications, thus, satisfying §R 4. This is possible due to the frame based structure of a bitstream, which is a sequence of commands and data. After a synchronization pattern and some set up, the actual configuration is represented by a repeating series of addresses and data. On a Xilinx 7 Series FPGA each frame consists of 101 words and a full bitstream of a XC7VX485T contains 50 176 frames [53]. A vFPGA is smaller and constraint to a specific area on the chip, which is described by a certain set of frames. The addresses of those frames are extracted during the design phase and do not change later on. They determine the allowed area a client's bitstream can influence.

The configuration filter, shown in Figure 21, acts as a proxy and is located before the ICAP. It receives the decrypted bitstream, scans for interrupting commands like global reset or shut down and blocks them. Its analyzer also detects the command to set the frame address. The address is passed to a set of six detectors, one for each vFPGA slot.

Each checker uses a set of predefined ranges, in Figure 21 five ranges are illustrated, to determine if the current address is within the enabled area. Their results are masked by the slot signal so that the check is only valid for the newly allocated vFPGA. If there is a match, i.e., the address is within the allowed ranges of the current slot, the configuration is passed on to the ICAP. Otherwise, this part of the bitstream is replaced with no-operation (NOP) commands.

The bitstream format is designed as a continuous stream with implicit addresses, in other words not each frame has to have a header specifying its address. A modified bitstream could start at a valid location and write a continuous sequence until the implicit address is outside of the allowed ranges. Thus, the configuration filter cannot only scan for commands to set the frame address. Through an internal counter the end of a 101-word frame is detected and, if another one follows directly afterwards, its address is calculated based on the start address and the current offset. This implicit address is than passed to the range checkers for verification.

E. Implementation

The RC2F$_{SEC}$ extension comprises an elliptic curve multiplier [54], which is shared by the ECDH and ECDSA cores. Furthermore, SHA3 and AES cores, developed by Hsing [55, 56], are used. A so called CMD Decoder handles the handshake and manages the other modules. The resource utilization is shown in Table I.

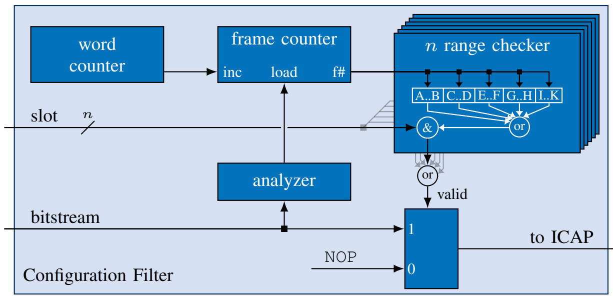


Figure 21. Only if a configuration frame is within the ranges (A..B, ...) of the newly allocated vFPGA slot, it is passed onto the ICAP, otherwise a NOP command is sent instead.

TABLE I. THE RC2F$_{SEC}$ EXTENSION'S RESOURCE UTILIZATION OF A XILINX VIRTEX-7 XC7VX485T.

Submodule	Slice LUTs	Slice Register	BRAM Tile
EC Key Processor[a, b]	30,766	15,158	0
CMD Decoder	7,279	8,714	87
Key Store	269	4,379	0
Configuration Filter	119	99	0
AES De-/Encryption[c]	9,207	11,640	172
Cross clock FIFOs	1,358	3,000	50
Overall	48,878	42,891	309

[a] EC processor by [57] [b] SHA3 core by [55] [c] provided by [56]

IX. 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 11 is shown in Table III. Furthermore, the table introduces the size of homogeneous vFPGA regions as outlined in Figure 16. The FPGA resources of every vFPGA can be described with the vector $\vec{\rho}$, which can be defined as shown in Equation (1):

$$\vec{\rho} = \begin{pmatrix} SliceLUTs \\ SliceRegister \\ BlockRAM \\ DSP \end{pmatrix} \tag{1}$$

The vector $\vec{\rho}$ is used in the following to calculate the FPGA resources inside a vFPGA. The aggregated FPGA resources of the homogeneous vFPGAs $\vec{\rho}_{vFPGA}$ can be calculated using Equation (2):

$$\begin{aligned} \vec{\rho}_{vFPGA}(N_{vFPGA-Slots}, N_{Frontends}) = N_{vFPGA-Slots} \\ \cdot \vec{\rho}_{vFPGA-Slot} + N_{Frontends} \cdot \vec{\rho}_{PPR} \end{aligned} \tag{2}$$

In Equation (2), the vector $\vec{\rho}_{vFPGA-Slot}$ describes the resources of a single vFPGA region, $N_{vFPGA-Slots}$ is the number of aggregated vFPGAs, $N_{Frontends}$ is the number of used frontends for the vFPGA and $\overrightarrow{\rho_{ppr}}$ represents the partition

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 (MB)	4.8	9.0	13.0	17.3	21.3	25.3
Locations	6	5	4	3	2	1
vRAI (MB)	33.6	54.0	65.0	69.2	63.9	50.6

TABLE III. 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).

FPGA-Ressource	Static	Utilization of static region					PPR	Into aggregated vFPGA regions and maximal number of frontends					
	region	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	28,400	56,800	85,200	113,600	142,000	188,400
Slice Register	189,648	11%	2%	1%	4%	**18%**	2,400	59,000	118,000	177,000	236,000	295,000	376,800
Block RAM Tile	369	23%	2%	2%	3%	**30%**	0	105	210	315	420	525	630
DSPs	726	–	–	–	–	**–**	20	340	680	1,020	1,360	1,700	2,040

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

TABLE IV. RECONFIGURATION AND MIGRATION TIMES IN SECONDS FOR DIFFERENT SIZED VFPGA-INSTANCES.

Operation	Size of the vFPGAs					
	Single	Dual	Triple	Quad	Quint	Hexa
(1) Readback (s)	0.76	1.43	2.07	2.76	3.39	4.04
(2) Relocate (s)	0.05	0.07	0.10	0.13	0.15	0.18
(3) Configuration (s)	0.04	0.06	0.09	0.11	0.13	0.15
Migration (s)	1.72	3.15	4.51	5.98	7.34	8.79

pin region (PPR) necessary to exclude the unused frontend interfaces from the grouped vFPGAs. When a frontend is used by a vFPGA, the resources inside the PPR are available to the user design inside the vFPGA. The open frontends, which are not used by the vFPGA are therefore treated as stubs and are securely sealed using a partial vFPGA bitstream. The resources of the corresponding PPR are not available inside a vFPGA. All regions except the largest one (Hexa), which has only one possible position, are homogeneous.

The throughput between vFPGAs and host (PCIe Gen2 8x on a Xilinx VC707) with different numbers of concurrently active vFPGAs is shown in Figure 22. 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 22 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. Table IV shows the times for configuration, design readback and relocation, as well as a complete migration process for different sized vFPGAs.

B. Scenario

In the following, we show a scenario based on a typical real-world application for our virtualization approach. The goal is to migrate vFPGA designs to achieve a high utilization as shown in Figure 23(e). In a system with jobs arriving and being finished at different points in time, situations as shown in Figure 23(c) 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 23(d), an area for a

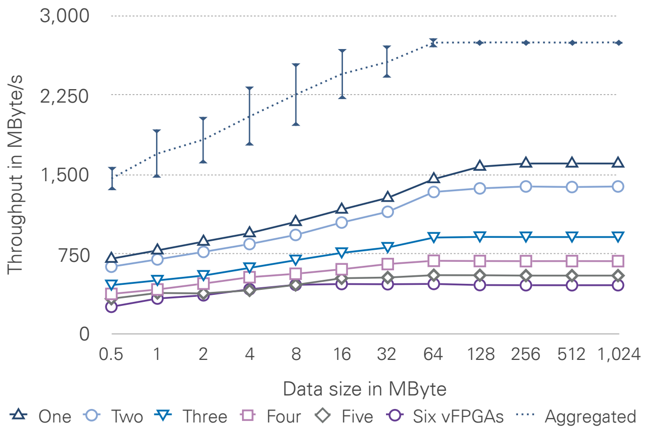


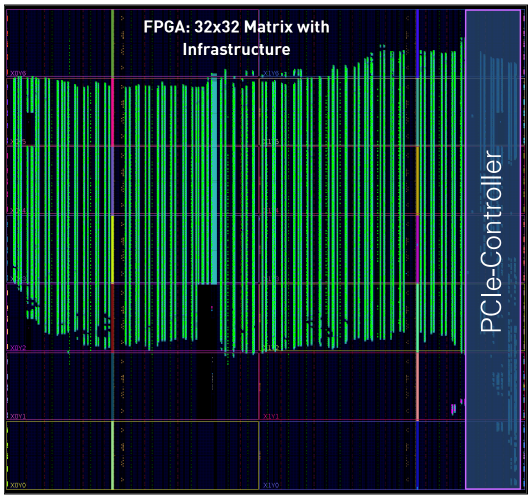
Figure 22. Throughput between host and FPGA with different numbers of concurrent vFPGAs. The diagram shows for each number of vFPGAs the average throughput of one representative vFPGA. The aggregated throughput is thereby the average throughput of all vFPGA compositions on the device.

group of three vFPGAs (triple) becomes available and makes higher utilization of the physical device possible.

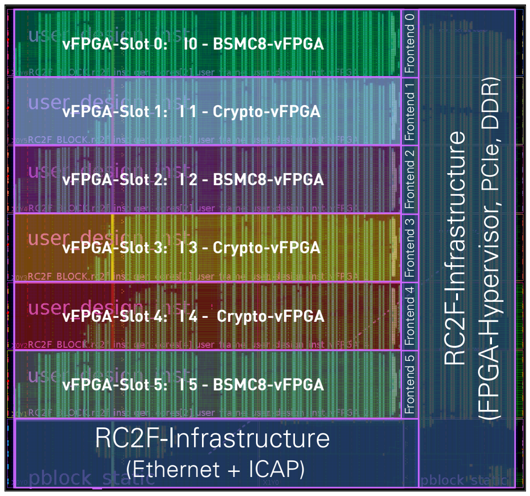
C. RC3E Large-Scale Datacenter Simulation

To evaluate the behavior of resource management, and in particular the benefits of virtualized FPGAs as well as their migration in a cloud, the RC3E simulator was developed. The results of the simulation are shown in Figure 24. In addition to the average number of allocated compute nodes, the table shows their energy requirements and the utilization of the FPGA resources available to the user. The Service Level Agreement (SLA) also specifies what proportion of the work packages will be processed within a certain time period (2.5 s) in order to be able to assess the system behavior with regard to the quality of the provision of the resources.

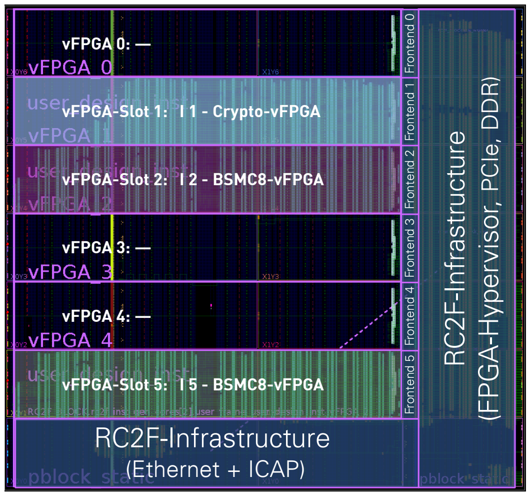
The energy requirement of the cloud is reduced to 69.35 % in the RC3E simulation in the reference scenario in load scenario (I) through the use of FPGAs and the utilization of the physical FPGAs is 27.34 %. RC2F virtualization reduces energy consumption to 24.43 % and increases the physical FPGA utilization to 78.14 %. An additional migration of the vFPGA instances to defragment the system increases utilization to 85.07 % and reduces energy consumption to 22.99 %. The SLA increases slightly by 0.04, or 0.02, as the virtualized resources are available faster than a re-allocating compute node. The additional migration contributes only marginally to saving resources and energy. However, the process of migration



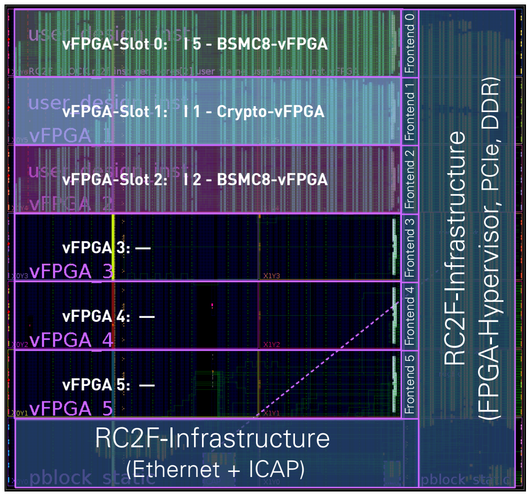
(a) Single userdesign in the classic RSaaS model, which allocates a full physical FPGA without utilizing the entire FPGA.


vFPGA-Slot 0: I0 - BSMC8-vFPGA
vFPGA-Slot 1: I1 - Crypto-vFPGA
vFPGA-Slot 2: I2 - BSMC8-vFPGA
vFPGA-Slot 3: I3 - Crypto-vFPGA
vFPGA-Slot 4: I4 - Crypto-vFPGA
vFPGA-Slot 5: I5 - BSMC8-vFPGA
Frontend 0
Frontend 1
Frontend 2
Frontend 3
Frontend 4
Frontend 5
RC2F-Infrastructure (Ethernet + ICAP)
RC2F-Infrastructure (FPGA-Hypervisor, PCIe, DDR)

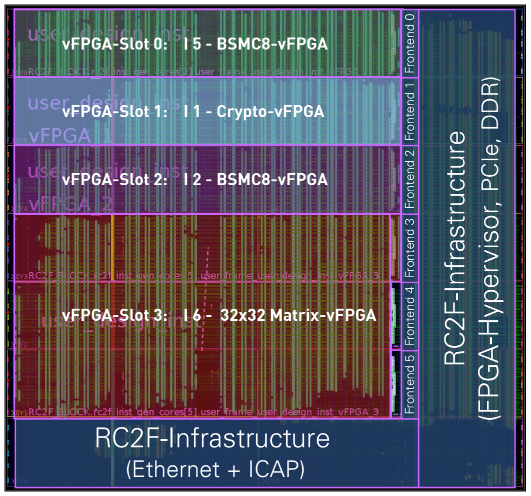
(b) Approximate full utilization of the FPGA with six independent users and designs.


vFPGA 0: —
vFPGA-Slot 1: I1 - Crypto-vFPGA
vFPGA-Slot 2: I2 - BSMC8-vFPGA
vFPGA 3: —
vFPGA 4: —
vFPGA-Slot 5: I5 - BSMC8-vFPGA
vFPGA_0
vFPGA_3
vFPGA_4
Frontend 0
Frontend 1
Frontend 2
Frontend 3
Frontend 4
Frontend 5
RC2F-Infrastructure (Ethernet + ICAP)
RC2F-Infrastructure (FPGA-Hypervisor, PCIe, DDR)

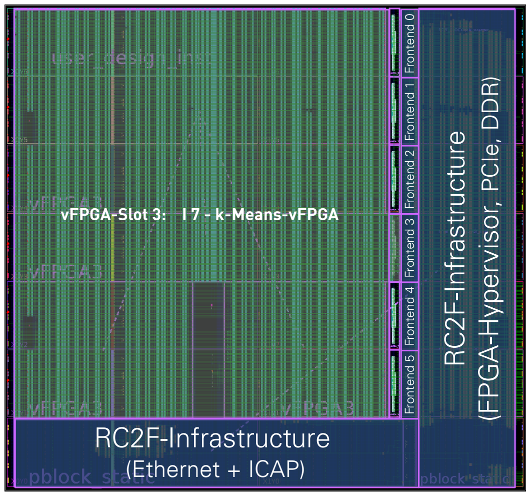
(c) Fragmentation of the physical FPGA caused by dynamic de- and allocation.


vFPGA-Slot 0: I5 - BSMC8-vFPGA
vFPGA-Slot 1: I1 - Crypto-vFPGA
vFPGA-Slot 2: I2 - BSMC8-vFPGA
vFPGA 3: —
vFPGA 4: —
vFPGA 5: —
vFPGA_3
vFPGA_4
vFPGA_5
Frontend 0
Frontend 1
Frontend 2
Frontend 3
Frontend 4
Frontend 5
RC2F-Infrastructure (Ethernet + ICAP)
RC2F-Infrastructure (FPGA-Hypervisor, PCIe, DDR)

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


vFPGA-Slot 0: I5 - BSMC8-vFPGA
vFPGA-Slot 1: I1 - Crypto-vFPGA
vFPGA-Slot 2: I2 - BSMC8-vFPGA
vFPGA-Slot 3: I6 - 32x32 Matrix-vFPGA
Frontend 0
Frontend 1
Frontend 2
Frontend 3
Frontend 4
Frontend 5
RC2F-Infrastructure (Ethernet + ICAP)
RC2F-Infrastructure (FPGA-Hypervisor, PCIe, DDR)

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


vFPGA-Slot 3: I7 - k-Means-vFPGA
Frontend 0
Frontend 1
Frontend 2
Frontend 3
Frontend 4
Frontend 5
RC2F-Infrastructure (Ethernet + ICAP)
RC2F-Infrastructure (FPGA-Hypervisor, PCIe, DDR)

(f) Example of a k-Means design using the largest vFPGAs with six Slots (Hexa).

Figure 23. Scenario with different users and designs on a Xilinx Virtex-7 XC7VX485T with six (vertically) scalable vFPGAs.

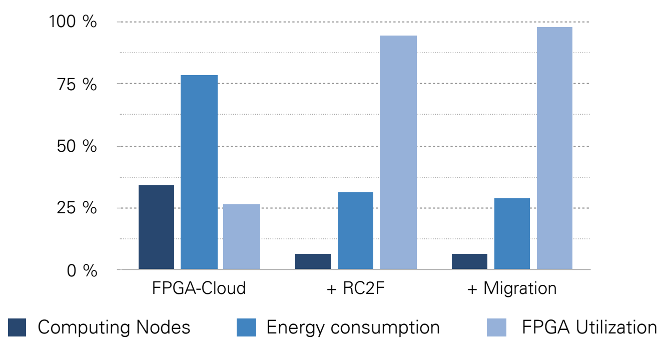


Figure 24. Comparison of the different system configurations within the RC3E simulation.

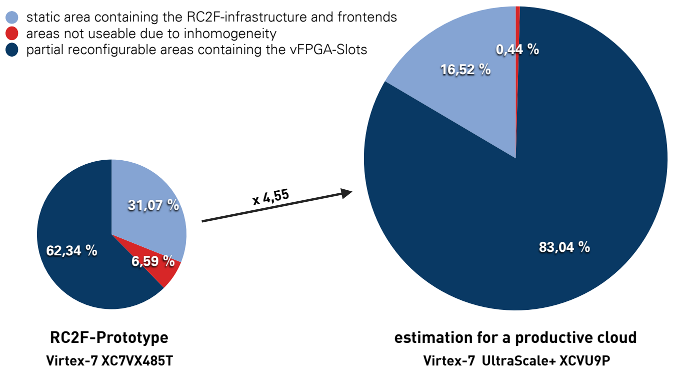


Figure 25. Size of the different regions of the RC2F virtualization transferred to a Xilinx Ultrascale+ FPGA.

TABLE V. RESULTS OF THE RC3E SIMULATION FOR THE SYSTEM CONFIGURATIONS WITH (1) SIMPLE COMPUTE NODES WITHOUT FPGAS, (2) ADDITIONAL FPGAS WITHOUT VIRTUALIZATION, (3) RC2F FPGA VIRTUALIZATION, AND (4) ADDITIONAL MIGRATION. THE SCENARIOS ARE THE LOAD DATA OF A REAL WEB SERVER [58] WITH 47,748 WORK PACKAGES OVER 1,440 MINUTES.

	Cloud (1)	+FPGA (2)	+RC2F (3)	+Mig(4)
Compute Node[a]	376	128	25	24
FPGA Utilization (%)	—	26.74	94.24	97.82
Energy Demand (kWh)	287.37	225.12	89.48	83.06
Energy Demand (%)	100.00	78.34	31.14	28.90
SLA[b]	0.96	0.90	0.92	0.91

[a] Average number of allocated compute nodes.
[b] SLA: Share of work packages being processed within 2.5 s.

adversely affects the SLA because migration is a high priority and new resources are delayed.

Based on the results of the RC3E simulation, it can be expected that both virtualization and the associated migration of vFPGAs can result in resource savings and thus energy without significantly reducing the SLA. The high savings can be explained by the chosen demonstrators and the work packages based on them. If the vFPGA designs completely expose the physical FPGA, virtualization can not save compute nodes and reduce power consumption. However, the migration allows the migration of the vFPGA images to other compute nodes, providing the ability to move parts of the system locally for maintenance, for example.

In addition to evaluating how virtualization and migration affect the optimization of utilization and energy consumption, the RC3E simulator also validated the mapping of vFPGAs to physical FPGAs and compute nodes.

X. 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 homogeneous 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.

The hardware accelerators used by Amazon in the EC2-F1 instances are Virtex-7 UltraScale+ FPGAs [59] on a VCU1525 Acceleration Development Kit with an XCVU9P [60]. A prognostic transfer of RC2F virtualization to an UltraScale+ (XCVU9P) FPGA provides the partitioning of FPGA resources into the different domains shown in Figure 25. The usable range for the vFPGAs is therefore 83.04 % and does not scale linearly with the size of the FPGA, which is 4.55 times larger than the Virtex-7 XC7VX485T. Due to the homogeneous structure of the UltraScale+ FPGAs, the unusable area has dropped to 0.44 % but still exists, so homogenization is still required.

One significant result of this paper is that the provision of homogeneous 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 [59] it is necessary to utilize the hardware efficiently with multiple designs in a scalable frame inside one physical FPGA. Such kind of flexible approach allows for adaption the individual resources to the users' requirements.

In the future, we plan to establish a productive cloud environment based on RC3E and RC2F at the Helmholtz-Zentrum Dresden-Rossendorf. The system should serve for background acceleration (BAaaS) of scientific applications like [61] and also for FPGA-prototyping (RSaaS) in combination with continuous integration (CI) [62] to optimize the process of hardware design and to satisfy the demands for automated tested FPGA designs for advanced research applications such as [63]. Other promising application areas are the mapping of applications and their distribution on a scalable FPGA cluster [64] and the evaluation of dynamic task offloading from CPUs to (virtualized) FPGAs during run-time, which will be developed on a similar system located at the chair of adaptive dynamic systems at Technische Universität Dresden. Furthermore, the systems are used to investigate economic impacts on hybrid (FPGA) cloud systems.

References

[1] O. Knodel, P. R. Genssler, and R. G. Spallek, “Virtualizing reconfigurable hardware to provide scalability in cloud architectures”, *Reconfigurable Architectures, Tools and Applications, RECATA 2017, ISBN: 978-1-61208-585*, vol. 2, 2017.

[2] O. Knodel, A. Georgi, P. Lehmann, W. E. Nagel, and R. G. Spallek, “Integration of a highly scalable, multi-fpga-based hardware accelerator in common cluster infrastructures”, in *42nd International Conference on Parallel Processing, ICPP 2013, Lyon, France, October 1-4*, IEEE, 2013, pp. 893–900.

[3] O. Knodel and R. G. Spallek, “RC3E: provision and management of reconfigurable hardware accelerators in a cloud environment”, *CoRR*, vol. abs/1508.06843, 2015. [Online]. Available: http://arxiv.org/abs/1508.06843.

[4] ——, “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.

[5] O. Knodel, P. Lehmann, and R. G. Spallek, “Rc3e: Reconfigurable accelerators in data centres and their provision by adapted service models”, in *9th Int’l Conf. on Cloud Computing, Cloud 2016, June 27 - July 2, San Francisco, CA, USA*, IEEE, 2016.

[6] 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.

[7] P. Genssler, O. Knodel, and R. G. Spallek, “A New Level of Trusted Cloud Computing - Virtualized Reconfigurable Resources in a Security-First Architecture”, in *Informatik 2017, 47. Jahrestagung der Gesellschaft für Informatik, 25.-29. September 2017, Chemnitz, Deutschland*, 2017.

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

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

[10] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, “The promise of high-performance reconfigurable computing”, *IEEE Computer*, vol. 41, no. 2, pp. 69–76, 2008.

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

[12] 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.

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

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

[15] 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.

[16] 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.

[17] V. Kulanov, A. Perepelitsyn, and I. Zarizenko, “Method of development and deployment of reconfigurable FPGA-based projects in cloud infrastructure”, in *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, May 2018, pp. 103–106. DOI: 10.1109/DESSERT.2018.8409108.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] A. Vaishnav, K. D. Pham, and D. Koch, “A survey on fpga virtualization”, *28th FPL*, 2018.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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. DOI: 10.1109/FCCM.2014.42.

[28] Q. Chen, V. Mishra, J. Nunez-Yanez, and G. Zervas, “Reconfigurable Network Stream Processing on Virtualized FPGA Resources”, *International Journal of Reconfigurable Computing*, vol. 2018, 2018.

[29] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, “Virtualized Execution Runtime for FPGA Accelerators in the Cloud”, *IEEE Access*, vol. 5, pp. 1900–1910, 2017, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2661582.

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

[31] 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.

[32] E. Rossi, M. Damschen, L. Bauer, G. Buttazzo, and J. Henkel, “Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing With FPGAs”, *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 2, 10:1–10:24, Jul. 2018, ISSN: 1936-7406. DOI: 10.1145/3182183.

[33] S. Rachana and H. Guruprasad, “Emerging security issues and challenges in cloud computing”, *International Journal of Engineering Science and Innovative Technology*, vol. 3, 2 2014, ISSN: 2319-5967.

[34] J. Ryoo, S. Rizvi, W. Aiken, and J. Kissell, “Cloud security auditing: Challenges and emerging approaches”, *IEEE Security Privacy*, vol. 12, no. 6, pp. 68–74, Nov. 2014, ISSN: 1540-7993. DOI: 10.1109/MSP.2013.132.

[35] M. A. Will and R. K. L. Ko, “Secure FPGA as a Service - Towards Secure Data Processing by Physicalizing the Cloud”, in *2017 IEEE Trustcom/BigDataSE/ICESS*, Aug. 2017, pp. 449–455.

[36] B. Hong, H.-Y. Kim, M. Kim, L. Xu, W. Shi, and T. Suh, "FASTEN: An FPGA-based Secure System for Big Data Processing", *IEEE Design & Test*, 2017.

[37] OpenStack. (2017). OpenStack - Open Source Cloud Computing Software, [Online]. Available: http://www.openstack.org/ (visited on 2018-11-25).

[38] J. E. Smith and R. Nair, *Virtual machines - versatile platforms for systems and processes*. Elsevier, 2005, ISBN: 978-1-55860-910-5.

[39] ——, "The architecture of virtual machines", *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[40] R. P. Goldberg, "Survey of virtual machine research", *Computer Journal*, vol. 7, no. 6, pp. 34–45, 1974.

[41] M. Rosenblum, "The Reincarnation of Virtual Machines", *ACM Queue*, vol. 2, no. 5, pp. 34–40, 2004.

[42] Xillybus Ltd. (2017). An FPGA IP core for easy DMA over PCIe, [Online]. Available: http://xillybus.com (visited on 2018-11-25).

[43] T. B. Preußer, M. Zabel, P. Lehmann, and R. G. Spallek, "The portable open-source ip core and utility library poc", in *2016 Int'l Conf. on ReConFigurable Computing and FPGAs (ReConFig)*, Nov. 2016, pp. 1–6. DOI: 10.1109/ReConFig.2016.7857191.

[44] Xilinx Inc., *7 Series FPGAs Integrated Block for PCI Express v3.3 – LogiCORE IP Product Guide*, PG054, 5. April, 2017.

[45] ——, *7 Series FPGAs Memory Interface Solutions – User Guide*, UG586, 18. Januar, 2012.

[46] ——, *LogiCORE IP Tri-Mode Ethernet MAC v5.2 – User Guide*, UG777, 18. Januar, 2012.

[47] H. Zimmermann, "Osi reference model - the iso model of architecture for open systems interconnection", *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr. 1980, ISSN: 0090-6778. DOI: 10.1109/TCOM.1980.1094702.

[48] 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.

[49] R. Backasch, G. Hempel, S. Werner, S. Groppe, and T. Pionteck, "Identifying homogenous reconfigurable regions in hetero"-gene"-ous fpgas for module relocation", in *ReConFigurable Computing and FPGAs (ReConFig), Int'l Conf. on*, IEEE, 2014, pp. 1–6.

[50] H. Gross, S. Mangard, and T. Korak, "An efficient side-channel protected aes implementation with arbitrary protection order", in *Cryptographers' Track at the RSA Conference*, Springer, 2017, pp. 95–112.

[51] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip", in *Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 23–40, ISBN: 978-3-642-33027-8. DOI: 10.1007/978-3-642-33027-8_2.

[52] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting", in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 296–310. DOI: 10.1109/SP.2007.36.

[53] Xilinx Inc., *7 series fpgas configuration, User guide 470*, 1.11, Sep. 27, 2016.

[54] C. Rebeiro and D. Mukhopadhyay, "High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms", in *Indocrypt*, Springer, vol. 5365, 2008, pp. 376–388. DOI: 10.1007/978-3-540-89754-5_29.

[55] H. Hsing. (Jan. 29, 2013). Opencores - sha3 core, [Online]. Available: https://opencores.org/project,sha3 (visited on 2018-11-25).

[56] ——, (Dec. 14, 2015). Opencores - tiny aes, [Online]. Available: https://opencores.org/project,tiny_aes (visited on 2018-11-25).

[57] D. Mukhopadhyay, C. Rebeiro, and S. Roy. (Dec. 9, 2008). Elliptic Curve Crypto Processor for FPGA Platforms, [Online]. Available: http://cse.iitkgp.ac.in/~debdeep/osscrypto/eccpweb/index.html (visited on 2018-11-25).

[58] ITA – The Internet Traffic Archive, *EPA-HTTP – A day of HTTP logs from a EPA WWW server.* 2016. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html (visited on 2018-11-25).

[59] Amazon Inc. (2018). Amazon EC2 F1 Instances – Run Custom FPGAs in the AWS Cloud, [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/ (visited on 2018-11-25).

[60] Xilinx Inc., *VCU1525 Reconfigurable Acceleration Platform – User Guide*, UG1268 (v1.0), 13. November, 2017.

[61] H. Burau, R. Widera, W. Honig, *et al.*, "Picongpu: A fully relativistic particle-in-cell code for a gpu cluster", *IEEE Transactions on Plasma Science*, vol. 38, no. 10, pp. 2831–2839, 2010.

[62] A. Schaefer, M. Reichenbach, and D. Fey, "Continuous integration and automation for devops", in *IAENG Transactions on Engineering Technologies*, Springer, 2013, pp. 345–358.

[63] R. Steinbrück, M. Kuntzsch, M. Justus, T. Bergmann, and A. Kessler, "Trigger generator for the superconducting linear accelerator elbe", 2016. DOI: 10.18429/JACoW-IBIC2015-MOPB011.

[64] L. Kalms and D. Gohringer, "Clustering and Mapping Algorithm for Application Distribution on a Scalable FPGA Cluster", in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2016, pp. 105–113.