

TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK
INSTITUT FÜR TECHNISCHE INFORMATIK

REKONFIGURIERBARE HARDWAREKOMPONENTEN IM KONTEXT VON CLOUD-ARCHITEKTUREN

Dissertation

zum Erlangen des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt von

Dipl.-Inf. Oliver Knodel
geboren am 11. Dezember 1984 in Herford

eingereicht am 08.03.2018

verteidigt am 18.05.2018

Betreuer:
Prof. Dr.-Ing. habil. Rainer G. Spallek, Technische Universität Dresden

Zweitgutachter:
Prof. Dr. rer. nat. habil. Mario Schölzel, Universität Potsdam

Fachreferent:
Prof. Dr. Wolfgang E. Nagel, Technische Universität Dresden

*"Begin at the beginning," the King said, very gravely,
"and go on till you come to the end: then stop."*

— LEWIS CARROLL, *Alice in Wonderland*, Chapter XII

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Dissertation zum Thema

Rekonfigurierbare Hardwarekomponenten im Kontext von Cloud-Architekturen

selbstständig verfasst und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt, nur die angegebenen Quellen benutzt und die in den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Oliver Knodel, Dresden, 08. März 2018

Wettbewerbsrechtlicher Hinweis

Die bloße Nennung von Namen, Produkten, Herstellern und Firmennamen dient lediglich als Information und stellt keine Verwendung des Warenzeichens sowie keine Empfehlung des Produktes oder der Firma dar.

Danksagung

Zu allererst möchte ich hiermit meinem Betreuer, Prof. Spallek, für die Unterstützung bei der Anfertigung dieser Arbeit danken. Insbesondere die vielen konstruktiven Anmerkungen in der Endphase und den begleitenden Veröffentlichungen führten immer wieder zu neuen (und teilweise zeitraubenden) Ideen, welche diese Arbeit aber letztendlich ausmachen. Bei meinem Zweitgutachter Prof. Schölzel möchte ich mich für eine konstruktive Diskussion und weitere Hinweise bedanken. Ebenso bedanke ich mich bei meinem Fachreferenten Prof. Nagel, welcher bei meinem Statusvortrag weitere gute Ansätze vorschlug. Besonderer Dank gilt auch meiner Korrekturleserin Catharina, die mich bei dieser Arbeit auf vielfältige Weise unterstützte und in schweren Zeiten immer wieder ermutigte, diese Arbeit fertigzustellen.

Ebenfalls „Danke“ sagen möchte ich den zahlreichen Studenten, die im Rahmen von Beleg- und Abschlussarbeiten Teile des RC3E und RC2F umsetzten und erweiterten und mir auch bei Publikationen hilfreich zur Seite standen. Den Kollegen am Lehrstuhl danke ich für die eine oder andere Diskussion sowie für Unterstützung bei administrativen Schritten, ebenso wie den in den letzten Monaten neu hinzugewonnenen Kollegen und Freunden, welche Auszüge der Arbeit zu dem gemacht haben, was sie sind. Zu guter Letzt möchte ich mich natürlich auch bei meinen Eltern, meiner Oma und allen Freunden für die moralische Unterstützung bedanken. Auch wenn die letzte Phase durchaus sehr anstrengend war, war die Promotion letztendlich eine schöne Zeit...

Kurzfassung

Rekonfigurierbare Schaltkreise wie Field Programmable Gate Arrays (FPGAs) stellen seit Jahren für viele Unternehmen eine Schlüsseltechnologie zur Hintergrundbeschleunigung von Anwendungen und Cloud-Diensten dar. Als weltweit führende Betreiber von Rechenzentren und Anbieter von Cloud-Infrastrukturen setzten mittlerweile Microsoft, IBM und demnächst auch Amazon in ihren Systemen FPGAs auf Anwendungsebene ein, um sowohl die Rechenleistung zu erhöhen als auch die Verlustleistung und damit die Betriebskosten zu reduzieren. Ebenso stellt die Erhöhung der Zugangssicherheit durch Nutzung von FPGAs einen weiteren bedeutenden Aspekt dar. Die zentrale Fragestellung dieser Arbeit besteht darin, wie FPGAs durch Virtualisierung effizient auf der Anwendungsebene nutzbar gemacht werden können. Das Ziel besteht darin, die FPGAs wie andere Komponenten flexibel und dynamisch in der Cloud einzusetzen, und lässt sich wie folgt zusammenfassen:

Evaluation eines flexiblen Einsatzes von FPGAs in der Cloud durch Virtualisierung der Hardwareressourcen und Abstraktion von der physischen Position im Rechenzentrum, um eine adaptive und flexible Architektur für unterschiedliche Anwendungsfälle zu realisieren.

Um ein Cloud-System mit FPGAs evaluieren zu können, werden zunächst Servicemodelle für eine Bereitstellung der virtualisierten FPGAs entwickelt und in eine Ressourcenverwaltung eingebettet. Ziel der Arbeit ist hierbei nicht der Aufbau einer Cloud-Architektur selbst, sondern vielmehr die Untersuchung ausgewählter Aspekte mit Hinblick auf die Integration rekonfigurierbarer Hardware in eine Cloud.

Dabei wird die klassische System-Virtualisierung auf die rekonfigurierbare Hardware übertragen, um eine Abstraktion vom physischen FPGA zu erreichen und diesen möglichst effizient auslasten zu können. Das Ziel besteht hierbei darin, mehrere unabhängige, nebenläufig arbeitende Nutzerkerne auf demselben physischen FPGA zu realisieren und durch Migration auf andere Rechenknoten zu übertragen sowie von der physischen Größe und der Architektur des FPGAs zu abstrahieren. Dabei wird nicht nur der FPGA virtualisiert, sondern – anders als bei der Mehrzahl vergleichbarer Arbeiten – das Gesamtsystem und der Einsatzzweck berücksichtigt.

Ein prototypisch entwickeltes Cloud-System wurde im Rahmen mehrerer Projekte evaluiert. Durch diese prototypische Umsetzung wird nachgewiesen, dass eine FPGA-Virtualisierung auf aktuellen FPGAs möglich ist und welche Kosten dazu erforderlich sind. Ebenso zeigt sich, dass aufgrund bestimmter fester Strukturen eine Etablierung von homogenen Bereichen notwendig ist, um die Migration eines partiellen FPGA-Kontextes zu ermöglichen und eine effiziente Lastverteilung in der Cloud zu realisieren. Die prototypische Implementierung zeigt, dass eine Migration mit aktuellen FPGA-Architekturen möglich, aber mit Kosten in Form von FPGA-Ressourcen verbunden ist. Des Weiteren wird mittels Simulation untersucht, ob die in einem komplexen Anwendungsszenario angewendete Migration auch in einem größeren Cloud-System zu einer effizienteren Auslastung der Ressourcen beitragen kann. Zusammenfassend ist sowohl durch die entwickelte Virtualisierung als auch durch die Möglichkeit einer Migration die Einsparung von Hardware-Ressourcen und somit auch Energie in einem modernen Cloud-System möglich.

Abstract

Reconfigurable circuits (Field Programmable Gate Arrays (FPGAs)) for accelerating applications have been a key technology for many years. Thus, the world's leading data center operators and providers of cloud infrastructures, namely Microsoft, IBM, and soon Amazon, are using FPGAs on their application platforms. The central question of this contribution is how FPGAs can be virtualized for a flexible and dynamic deployment in cloud infrastructures. The question and can be summarized as follows.

Evaluation of the flexible use of FPGAs in cloud infrastructures by virtualization of resources and abstraction from the physical location in the data center to provide adaptive and flexible architectures for a variety of applications.

In addition to the virtualization of FPGA resources, service models for the provision of virtualized FPGAs are developed and embedded into a resource management system in order to evaluate the cloud system's behaviour. The objective of this work is not to build a cloud architecture, but rather to examine selected aspects of cloud systems with regard to the integration of reconfigurable hardware. The FPGAs are not only virtualized but, unlike in many other projects, the entire system and the application are taken into account. As a result, the vFPGAs are used dynamically and adaptively at different locations and topologies in the cloud architecture, depending on the user's requirements.

Furthermore, a prototypical implementation of a cloud system has been developed, and evaluated in several projects. The virtualization using state-of-the-art FPGAs has shown that the establishment of homogenous environments is possible. The Migration of a partial FPGA context is also possible with current FPGA architectures, but is associated with high costs in form of hardware resources. Furthermore, a simulation was carried out to determine whether virtualization and migration, could contribute to a more efficient utilization of resources in a larger cloud system or impair the service level agreement. In summary, both the developed virtualization and the possibility of a migration make it possible to reduce the amount of necessary resources in a modern cloud system.

Inhaltsverzeichnis

Abbildungsverzeichnis	XII
Tabellenverzeichnis	XV
Verzeichnis von Listings	XVII
Verzeichnis der Definitionen	XVII
Abkürzungsverzeichnis	XIX
Verzeichnis der Formelzeichen	XXIII
1 Einleitung	1
1.1 Motivation	1
1.2 Historische Entwicklung	2
1.3 Zielsetzung und Beitrag	3
1.4 Herangehensweise und Strukturierung der Arbeit	4
2 Stand der Forschung und Technik	7
2.1 Hardwarebeschleuniger auf Basis rekonfigurierbarer Hardware	7
2.1.1 Arten von Hardwarebeschleunigern und deren Leistungsbewertung	7
2.1.2 Architektur von FPGAs, Kopplungs- und Konfigurationsarten	13
2.1.3 Einsatzgebiete und typische Anwendungen	17
2.1.4 Entwurfsablauf, dynamische partielle Rekonfiguration und Erweiterungen	18
2.2 Virtualisierung	24
2.2.1 Historische Entwicklung, Definition und Begriffe	24
2.2.2 Prozess-Virtualisierung	26
2.2.3 System-Virtualisierung	27
2.2.4 Virtualisierungsansätze für spezielle Geräte	32
2.2.5 Virtualisierungsarten und deren Relevanz für eine FPGA-Virtualisierung	33
2.3 Cloud-Computing	34
2.3.1 Historische Entwicklung, Definition und Begriffe	34
2.3.2 Wesentliche Bestandteile und Schlüsseltechnologien	39
2.3.3 Herausforderungen im Cloud-Computing	42
2.4 Relevante Forschungsarbeiten zu FPGAs und Cloud	43
2.4.1 Virtualisierung von Hardwarebeschleunigern auf Basis von FPGAs	44
2.4.2 Beschleunigung von Cloud-Diensten mit rekonfigurierbarer Hardware	54
2.4.3 Einsatz von FPGAs zur Erhöhung der Sicherheit	54
2.4.4 Verwaltung und Bereitstellung von FPGAs in Rechenzentrum und Clouds	56
2.5 Einordnung der Arbeit	62
2.5.1 Offene Forschungsfragen	62
2.5.2 Zielsetzung und Beitrag	64

3 Anforderungsanalyse und Zielstellung für virtualisierte FPGAs im Cloud-Kontext	67
3.1 Abstraktionsschichten zur Etablierung von FPGA-spezifischen Diensten in der Cloud	67
3.1.1 Erweiterung der Beteiligten und Charakterisierung der Rollen	68
3.1.2 Entwicklung von FPGA-relevanten Servicemodellen/Clouddiensten	68
3.1.3 Vertikaler Aufbau der Cloud-Dienste	71
3.2 Entwurf einer adaptiven Architektur mit dynamischem Einsatz von FPGAs	72
3.2.1 Virtuelle Sicht der Nutzer auf die FPGAs innerhalb der Cloud	72
3.2.2 Flexibilität im Einsatz von FPGAs in der Cloud	74
3.2.3 Anforderungen an eine FPGA-Virtualisierung	75
3.3 System- und Softwarearchitektur für die Verwaltung virtualisierter FPGA-Ressourcen	77
3.3.1 Analyse einer geeigneten Hardware-Architektur	77
3.3.2 Hierarchische Aufteilung der Verwaltungsaufgaben	79
3.3.3 Anforderung an die Verwaltung von FPGAs in einem Cloud-System	80
3.4 Definition innerhalb der FPGA-Virtualisierung im Cloud-Kontext	81
3.5 Systementwurf und Abgrenzung	82
4 Virtualisierung der FPGAs für den Einsatz in einer dynamischen Cloud-Architektur	85
4.1 Grundbausteine einer Virtualisierung von FPGAs für den Einsatz in Cloud-Architekturen	85
4.2 Konzept zur Virtualisierung eines physischen FPGAs	86
4.2.1 Aspekte der Übertragung einer klassischen Virtualisierung auf FPGAs	86
4.2.2 Mehrbenutzerbetrieb auf rekonfigurierbarer Hardware	88
4.3 Entwurfsraum einer FPGA-Virtualisierung	89
4.3.1 Größe, Position und Flexibilität der vFPGAs	89
4.3.2 Anordnung, Schnittstellen und Architektur der vFPGAs	90
4.3.3 Kommunikationskanäle zwischen den vFPGAs, Host und Netzwerk	92
4.3.4 Skalierung der vFPGAs auf dem physischen FPGA und darüber hinaus	94
4.3.5 Zustände und Kontextmigration der vFPGAs	95
4.3.6 Virtualisierung des Speichers	96
4.3.7 Konfiguration der FPGAs und Rolle des Hypervisors	96
4.4 Virtualisierung von FPGAs im Cloud-Einsatz – RC2F	98
4.4.1 Systemarchitektur für die RC2F-Infrastruktur	98
4.4.2 Zustände der vFPGAs und deren Verwaltung	103
4.4.3 Konfiguration von FPGA und vFPGAs	105
4.4.4 Dynamische Größe der vFPGAs durch partielle Rekonfiguration	107
4.4.5 Kommunikationsschnittstellen und deren Virtualisierung	107
4.4.6 Paravirtualisierung auf dem Host zur Inter-Domain-Kommunikation	109
4.4.7 Speicher-Virtualisierung	110
4.4.8 Erweiterung zur sicheren rekonfigurierbaren Hardware in der Cloud	111
5 Entwurfsprozess und Verwaltungshierarchie der Cloud	115
5.1 Entwurfsprozess und Verwendung der vFPGAs	115
5.1.1 Entwurfsablauf innerhalb der unterschiedlichen Servicemodelle	115
5.1.2 Modifizierter Entwurfsablauf für Entwickler im Modell RAaaS	116
5.1.3 Bereitstellung von Host-API und Bibliotheken zur Integration der vFPGAs	117
5.1.4 Beschreibung einer FPGA-Ressource als RCFG-Datei	118
5.1.5 Erweiterung des Entwurfsprozesses für das Modell BAaaS	120
5.1.6 Kapselung vollständiger Beschleuniger-Umgebungen in vRAI-Pakete	121
5.1.7 Einsatz der vFPGAs und vRAIs für sicherheitsrelevante Anwendungen	123
5.2 Erweiterung einer System-Architektur zur Verwaltung von virtualisierten FPGAs	123
5.2.1 Aufbau und Verwaltungsebenen	123

5.2.2	Hierarchische Verteilung der Anfragen auf die verfügbaren Ressourcen	125
5.2.3	Erweiterung des Host-Hypervisors innerhalb der Rechenknoten	127
5.2.4	Auslastung der vFPGAs im adaptiven elastischen Scheduling	128
5.3	Aufbau einer Umgebung aus virtuellen Komponenten für unterschiedliche Szenarien	129
5.3.1	Abbildung der virtuellen Systembeschreibung auf die physische Cloud-Architektur	129
5.3.2	Hardwarebeschleuniger (BAaaS)	130
5.3.3	FPGA-Cluster (RAaaS)	131
5.3.4	Direkter Zugang über das Netzwerk (RAaaS)	131
6	Prototypische Implementierung und Ergebnisse	135
6.1	Prototypische Implementierung und Aufbau einer Cloud	135
6.1.1	Hardwareaufbau des Cloud-Prototypen zur Integration von FPGAs in eine Cloud .	135
6.1.2	Prototypischer Aufbau der Cloud-Ressourcenverwaltung – RC3E	136
6.1.3	Modellierung des RC3E-Simulators	137
6.1.4	Realisierung eines Prototypen der FPGA-Virtualisierung – RC2F	141
6.2	Demonstratoren und ausgewählte Szenarien	154
6.2.1	Demonstratoren zur Evaluation der FPGA-Virtualisierung RC2F	155
6.2.2	Allgemeiner Einsatz der virtualisierten FPGAs	158
6.2.3	Lastszenarien für die Simulation des RC3E	158
6.3	Evaluation und Validierung der Prototypen	160
6.3.1	Validierung des RC3E-Prototypen auf Ebene der Cloud-Verwaltung	161
6.3.2	Evaluation der Ressourcenverwaltung durch die RC3E-Simulation	161
6.3.3	Evaluation des Lebenszyklus eines virtualisierten FPGAs	163
6.4	Abschließende Bewertung und weiterführende Betrachtungen	169
6.4.1	Erkenntnisse der FPGA-Virtualisierung innerhalb des Cloud-Prototypen	169
6.4.2	Erkenntnisse der RC3E-Simulation für einen Einsatz in einer Cloud	171
6.4.3	Weitere Entwicklungen und Ausblick – Xilinx UltraScale+ FPGA	171
6.4.4	Vergleich mit anderen Arbeiten	172
7	Zusammenfassung und Ausblick	175
7.1	Zusammenfassung	175
7.2	Ausblick	177
7.3	Aspekte für zukünftige FPGA-Architekturen	177
Literaturverzeichnis		XXVII
Übersicht eigener Veröffentlichungen im Kontext der Dissertation		XLIX
Anhang		A-1
A	Grundlagen	A-1
A.1	Rekonfigurierbare Hardware	A-1
A.2	Gerätevirtualisierung allgemein	A-5
A.3	Cloud-Computing	A-7
B	Zeitliche Einordnung	B-1
C	Prototypische Implementierung des RC2F	C-1
C.1	RC2F-Application Programming Interface	C-1
C.2	RC2F-Infrastruktur	C-4
C.3	Übertragung des RC2F auf einen Xilinx UltraScale+ FPGA	C-5
D	RC3E-Simulator	D-1

Abbildungsverzeichnis

2.1	Übersicht unterschiedlicher Hardwarebeschleuniger	9
2.2	Typische Kopplung zwischen Hardwarebeschleuniger und Hauptprozessor	9
2.3	Schematischer Aufbau eines FPGAs	14
2.4	Aufbau eines einfachen Rechnersystems	15
2.5	Möglichkeiten der Kopplung von FPGAs mit Prozessoren	16
2.6	Entwurfsablauf für FPGA-Konfigurationen (Bitstreams)	19
2.7	Prinzip der dynamischen partiellen Rekonfiguration	21
2.8	Aufbau eines (partiellen) Bitstreams	22
2.9	Unterscheidung von Prozess- und System-Virtualisierung	25
2.10	Taxonomie der Virtuellen Maschinen	26
2.11	Übersicht zu nativer und Hosted-Virtualisierung mit Privilegierung	28
2.12	Paravirtualisierung und die Ringe der x86-Architektur	30
2.13	Paravirtualisierung mit modifiziertem Treiber innerhalb des Gast-Betriebssystems	31
2.14	Entwicklung des Cloud-Computing	35
2.15	Cloud Service Modelle	37
2.16	Rollen und beteiligte Akteure	39
2.17	Szenarien zur Auslastung von Ressourcen in einer elastischen Cloud	41
2.18	Cloud-Architektur mit Einsatzgebieten von FPGAs	44
2.19	Überblick des VirtualRC Frameworks	46
2.20	Überblick des Systemaufbaus von BORPH auf Prozessebene	46
2.21	Überblick des pvFPGA Systems	48
2.22	Virtualisierung des physischen FPGAs durch feste Einteilung in Bereiche	49
2.23	Überblick des ReconOS Systems	50
2.24	Hardwarearchitektur eines Knotens von Microsofts Catapult System	58
3.1	Cloud mit virtualisierten FPGAs und VMs	67
3.2	Übersicht über die drei FPGA-spezifischen Dienste: RSaaS, RAaaS und BAaaS	69
3.3	Grundlegende Hardwaredesigns für die drei Servicemodele	70
3.4	Verkettung der FPGA-spezifischen Cloud-Dienste	72
3.5	Logische Sicht auf die virtuellen Ressourcen vFPGA und VM in einer Cloud	73
3.6	Skalierung der vFPGAs in ihrer Größe	75
3.7	Unterschiedliche Möglichkeiten zur Integration von Hardwarebeschleunigern	77
3.8	Physische Zielarchitektur für die unterschiedlichen Szenarien	78
3.9	Logische Systemarchitektur der Verwaltung von FPGA-Ressourcen	79
4.1	Konzept der Paravirtualisierten System-VM übertragen auf FPGAs	87
4.2	Unterschiedliche Ansätze zur Virtualisierung von FPGAs	88
4.3	Paravirtualisierung mit unterschiedlicher Größe der vFPGAs	90
4.4	Entwurfsraum für die Architektur der vFPGAs	91
4.5	Aufteilung von Hypervisor auf Host und FPGA	97
4.6	Systemarchitektur der RC2F-Infrastruktur	99
4.7	Konfigurationsspeicher (HCU) für den FPGA-Hypervisor	101
4.8	Konfigurationsspeicher (vCS) der vFPGAs	102
4.9	Architektur eines vFPGAs des RC2F	102
4.10	Zustände und Zustandsübergänge der vFPGAs	104
4.11	Anordnung der vFPGAs aufgrund der technischen Randbedingungen	106

Abbildungsverzeichnis

4.12 Aufbau des Moduls zur Hardware-Virtualisierung	108
4.13 Logische Sicht der Nutzer auf ein VM/vFPGA-System	109
4.14 Hypervisor auf FPGA und Host mit Paravirtualisierung	110
4.15 Speicher-Virtualisierung mit Seitentabellen	111
4.16 Größe der Seitentabellen in Abhängigkeit der Seitengrößen	112
4.17 Sicherheitskonzept für FPGAs im Cloud-Einsatz	113
5.1 Entwurfsablauf zur Erzeugung eines vFPGA-Image	117
5.2 Beschleuniger-Paket vRAI	122
5.3 Architektur des Ressourcenverwaltung RC3E	124
5.4 Sequenzdiagramm mit Interaktion der Ebenen im RC3E-System	126
5.5 Berechnung der Auslastung der FPGAs	128
6.1 Aufbau des Cloud-Prototypen zur Evaluation der Ressourcenverwaltung RC3E	136
6.2 Aufbau des RC3E-Simulators zur Evaluation der Lastverteilung	138
6.3 Aufbau eines Arbeitspaketes des RC3E-Simulators	140
6.4 Übersicht zu den erforderlichen I/O- und Infrastruktur-Komponenten	142
6.5 RC2F-Prototyp mit Bereichen für sechs virtueller FPGA (vFPGA)-Slots und die Infrastruktur	142
6.6 Einbußen in den Ressourcen der unterschiedlichen vFPGAs	143
6.7 Anteil der Hardware-Ressourcen in den Bereichen innerhalb des RC2F-Prototypen	144
6.8 Prozentualer Bedarf an FPGA-Ressourcen der Komponenten der RC2F-Infrastruktur	146
6.9 Bedarf an FPGA-Ressourcen der RC2F-Infrastruktur in Abhängigkeit von den vFPGA-Slots	148
6.10 Aufbau der aggregierten vFPGAs über mehrere vFPGA-Slots	149
6.11 Erforderliche vFPGA-Images für unterschiedliche Größen von (aggregierten) vFPGAs	150
6.12 Modifizierter Synthesearlauf zur Erzeugung homogener vFPGA-Images	151
6.13 Größe der vRAI-Pakete für den RC2F-Prototypen (Virtex-7)	154
6.14 Aufbau der Matrixmultiplikation auf Basis von vFPGAs und RC2F-API	155
6.15 Einsatz des Crypto-vFPGAs zur Ver- und Entschlüsselung eines Datenstromes	157
6.16 Szenario für den allgemeinen Einsatz der virtualisierten FPGAs	158
6.17 Lastzenario (I) mit einer synthetischen Arbeitslast	159
6.18 Lastszenario (II) mit 47.748 Anfragen auf Basis eines Webservers	159
6.19 RC3E-Simulation für Lastszenario (II) mit FPGA-Virtualisierung und Migration (3)	162
6.20 Vergleich der unterschiedlichen Systemkonfigurationen innerhalb der RC3E-Simulation	163
6.21 Teilschritte des Entwurfsablaufs zur Erzeugung homogener vFPGA-Images am Beispiel	164
6.22 Erreichbarer Netto-Datendurchsatz zwischen Host und vFPGAs	165
6.23 Zeiten für die Migration unterschiedlich Großer vFPGA-Instanzen	166
6.24 Verhältnis von Konfiguration und Migration zur Laufzeit der Arbeitspakete	167
6.25 Szenarien mit unterschiedlich großen vFPGAs	168
6.26 Aufbau eines konzeptionellen SoC-Cloud-FPGAs zur Bereitstellung vFPGAs	170
6.27 Bereitstellung homogener vFPGAs innerhalb des RC2F auf einem Xilinx UltraScale+	171
A.1 Bestandteile eines modernen FPGAs	A-2
A.2 Aufbau eines Xilinx Virtex-7 FPGAs	A-3
A.3 Entwicklung der GMAC-Leistung von CPU, FPGA und GPU	A-4
A.4 Arten des Cloud-Betriebs	A-9
A.5 Einteilung und Charakterisierung von typischen Cloud-Anwendungen	A-11
B.1 Zeitachse mit einer Einordnung relevanter Arbeiten	B-1
B.2 Klassifikation relevanter Arbeiten	B-2
B.3 Klassifikation relevanter Arbeiten und Einordnung des RC3E und RC2F	B-2
C.1 Testanordnung zur Validierung der Kommunikation zwischen Host und vFPGAs	C-4
C.2 Einteilung der Bereiche auf einem Xilinx UltraScale+ FPGA	C-5
D.1 RC3E-Simulation des Lastszenarios (I) mit Virtualisierung und Migration	D-1

Tabellenverzeichnis

2.1	Überblick über Arten von Hardwarebeschleunigern	10
2.2	Qualitativer Vergleich von unterschiedlichen Hardwarebeschleunigern	12
2.3	Überblick zur Partiellen Rekonfiguration	22
2.4	Charakterisierung der Virtualisierungsarten	33
2.5	Überblick über relevante Forschungsarbeiten zur Virtualisierung von FPGAs	53
2.6	Rekonfigurierbare Beschleuniger für rechenintensive Cloud-Dienste	54
2.7	Relevante Arbeiten zur Bereitstellung von FPGAs in der Cloud	61
4.1	Entwurfsraum für die Aufgabenverteilung auf Host- oder FPGA-Hypervisor	97
6.1	Benötigte FPGA-Ressourcen der statischen RC2F-Infrastruktur	147
6.2	FPGA-Ressourcen in Abhängigkeit der Anzahl der vFPGA-Slots	147
6.3	Approximation der Messwerte aus Tabelle 6.2 durch lineare Regression	148
6.4	FPGA-Ressourcen der aggregierten vFPGA-Slots innerhalb des RC2F-Prototypen	149
6.5	Größe der vFPGA-Images und resultierenden vRAIs (Virtex-7)	154
6.6	FPGA-Ressourcen und erforderliche vFPGAs-Slots der Demonstratoren (Virtex-7)	157
6.7	Ergebnisse der RC3E-Simulation an ausgewählten Szenarien	162
6.8	Erzeugte vRAI-Pakete für die vFPGA-Demonstratoren	163
6.9	Laufzeiten zur Konfiguration und Kontextmigration von vFPGA-Instanzen	166
6.10	Qualitativer Vergleich mit ähnlichen Arbeiten	172
A.1	Überblick zu FPGA-Beschleuniger Frameworks	A-4
C.1	Auszug aus der RC2F Host-API	C-3
C.2	FPGA-Ressourcen einer RC2F-Virtualisierung eines Xilinx UltraScale+ FPGA	C-6

Verzeichnis von Listings

5.1	Beispiel einer RCFG für einen vollständigen FPGA im Modell RSaaS.	119
5.2	Beispiel einer RCFG für einen vFPGA im Modell RAaaS.	119
5.3	Beispiel einer RCFG für einen vFPGA im Modell BAaaS.	120
5.4	RCFG-Datei für einen FPGA-basierten Hardwarebeschleuniger mit VM.	130
5.5	RCFG-Datei für einen FPGA-Cluster mit zwei Host-VMs und zugeordneten vFPGAs.	131
5.6	RCFG-Datei für den Einsatz des vFPGA zwischen Netzwerk und VM.	132
C.1	Headerdatei der RC2F-API.	C-1

Verzeichnis der Definitionen

2.1	Definition (Coprocessor)	7
2.2	Definition ((Re)configurable Computing)	13
2.3	Definition (Virtualization (Computer))	24
2.4	Definition (Cloud-Computing)	35
2.5	Definition (Service Level Agreement (SLA))	42
3.1	Definition (vFPGA)	81
3.2	Definition (vFPGA-Slots)	81
3.3	Definition (vFPGA-Design)	82
3.4	Definition (vFPGA-Image)	82
3.5	Definition (vFPGA-Instanz)	82
3.6	Definition (System-Hypervisor)	82
3.7	Definition (RC2F Host-Hypervisor)	82
3.8	Definition (FPGA-Hypervisor)	82

Abkürzungsverzeichnis

ABI	Application Binary Interface	DaaS	Data as a Service
AES	Advanced Encryption Standard	DBaaS	Database as a Service
AFI	Amazon FPGA Image	DDR	Double Data Rate
AMD	Advanced Micro Devices	DMA	Direct Memory Access
AMI	Amazon Machine Image	DP	Double Precision
API	Application Programming Interface	DPI	Deep Packet Inspection
ASIC	Application Specific Integrated Circuits	DR	Design Relocation
AWS	Amazon Web Services	EC2	Elastic Compute Cloud
BAaaS	Background Acceleration as a Service	ECB	Electronic Codebook
BBRAM	Battery Backup Random Access Memory	ECC	Error Correction Code
BEL	Basic Element Location	EEPROM	Electrically Erasable Programmable Read-Only Memory
BLAST	Basic Local Alignment Search Tool	FAR	Frame Adress Register
BORPH	Berkeley Operating System for ReProgrammable Hardware	FIFO	First In First Out
BPI	Byte Peripheral Interface	FPGA	Field Programmable Gate Array
BRAM	Block-RAM	FSB	Front-Side Bus
BSCAN	Boundary-Scan (JTAG)	FU	Functional Unit
BSMC	Monte-Carlo-Simulation des Black-Scholes-Modells	GFLOPS	Giga Floating-Point Operations per second
CAPI	Coherent Accelerator Processor Interface	GMAC	Giga Multiply-Accumulate
CBC	Cipher Block Chaining	GPGPU	General Purpose Graphics Processing Unit
CBE	Cell Broadband Engine	GSR	Global Set/Reset
CB	Connection Block	GTX	Transceiver-Komponente in Xilinx Virtex-7 FPGAs
CLB	Configurable Logic Block	HAaaS	Hardware Accelerator as a Service
CPLD	Complex Programmable Logic Device	HCS	Hypervisor Configuration Space
CPU	Central Processing Unit	HCU	Hypervisor Control Unit
CRC	Cyclic Redundancy Check	HDK	Hardware Developer Kit
CUDA	Compute Unified Device Architecture		

Abkürzungsverzeichnis

HDL	Hardware Description Language	MFT	Multiprogramming with a Fixed number of Tasks
HLL	High-Level Language	MMCM	Mixed-Mode Clock Manager
HLS	High-Level Synthesis	MMU	Memory Management Unit
HMAC	Hashed Message Authentication Code	MPI	Message Passing Interface
HPCaaS	High Performance Computing as a Service	MVT	Multiprogramming with a Variable number of Tasks
HPRC	High Performance Reconfigurable Computing	NASA	National Aeronautics and Space Administration
HP	Hardware Preemption	NAS	Network Attached Storage
HT	HyperTransport-Link	NAT	Network Address Translation
HVM	Hardware Virtual Machine	NCBI	National Center for Biotechnology Information
IaaS	Infrastructure as a Service	NIC	Network Interface Card
ICAP	Internal Configuration Access Port	NIST	National Institute of Standards and Technology
IDF	Isolation Design Flow	NoC	Network-on-Chip
IOB	Input/Output Block	OCRAM	On-Chip Random Access Memory (RAM)
IOMMU	Input/Output-Memory Management Unit	OSI	Open Systems Interconnection
IP	Internet Protocol	OSIF	Operating System Interface
IP-Core	Intellectual Property Core	OpenCL	Open Computing Language
IPC	Inter-Process Communication	PaaS	Platform as a Service
IPSec	Internet Protocol Security	PCIe	Peripheral Component Interconnect Express
ISA	Instruction Set Architecture	PCI-SIG	PCI-Special Interest Group
JTAG	Joint Test Action Group	PLL	Phase Locked Loop
JVM	Java Virtual Machine	PoC	Pile of Cores
KVM	Kernel-based Virtual Machine	PP	Partition Pin
kWh	Kilowattstunde	PPR	Partition Pin Region
LAN	Local-Area Network	PR	Partial Reconfiguration
LL	Logic Location (.ll)	PSM	Programmable Switch Matrix
LOC	Location Constraints	QoS	Quality of Service
LUT	Look Up Table	QPI	QuickPath Interconnect
LVM	Logical Volume Management		
LXC	Linux-Container		
MAC	Media-Access-Control		

RAaaS	Reconfigurable Accelerators as a Service	SPMD	Single Programm Multiple Data
RAM	Random Access Memory	SPMT	Single Program Multiple Thread
RC2F	Reconfigurable Common Computing Frame(work)	SP	Single Precision
RC3E	Reconfigurable Common Cloud Computing Environment	SR-IOV	Single Root-Input/Output-Virtualization
RCFG	Reconfigurable (Device) Config uration	SRAM	Static Random Access Memory
RRaaS	Reconfigurable Region as a Service	SRMP	Short Read Mapping Problem
RSaaS	Reconfigurable Silicon as a Service	SVM	Secure Virtual Machine
RTI	Real-Time-Infrastructure	TA	Trusted Authority
RTL	Register Transfer Level	TDP	Thermal Design Power
s	Sekunde	TPM	Trusted-Platform-Module
S3	Simple Storage Service	VCBM	Virtual Context Bit Mask (.vcbm)
SaaS	Software as a Service	VCC	Virtual Context Content (.vcc)
SAN	Storage Area Network	VCL	Virtual Context Locations (.vcl)
SDR	Software Defined Radio	vCS	Virtual Control Space
SDVM	Self Distributing Virtual Machine	vCU	vFPGA Control Unit
SEAL	Simple Encrypted Arithmetic Library	VE	Verarbeitungseinheit
SecFPGA	Secured FPGA	vFPGA	virtueller FPGA
SEU	Single-Event Upset	VHDL	Very High Speed Integrated Circuit Hardware Description Language
SGX	Software Guard Extensions	VLAN	Virtual Local-Area Network
SHA	Secure Hash Algorithm	VM	Virtuelle Maschine
SIMD	Single Instruction Multiple Data	VMM	Virtual Machine Monitor
SLA	Service Level Agreement	vRAI	Virtual Reconfigurable Acceleration Image (.vrai)
Soc	System-on-Chip	VT	Virtualization Technology
SODIMM	Small Outline Dual Inline Memory Module	W	Watt
SPEC	Standard Performance Evaluation Corporation	XaaS	Everything as a Service
SPI	Serial Peripheral Interface	XDL	Xilinx Design Language
SPLD	Simple Programmable Logic Device		

Verzeichnis der Formelzeichen

Max_{FPGAs}	Anzahl der FPGAs in einem Rechenknoten.	$P_{vFPGA-Slot}$	Leistungsaufnahme (in Watt) eines vFPGA-Slots innerhalb des RC3E-Simulators.
Max_{Kerne}	Anzahl der Rechenkerne des Host-Prozessors innerhalb RC3E-Simulation.	R^2	Bestimmtheitsmaß zum Nachweis der Qualität der linearen Regression.
Max_{Knoten}	Anzahl der Rechenknoten innerhalb der RC3E-Simulation.	ρ_{FPGA}	Auslastung des physischen FPGAs bis zum migrieren von vFPGA-Ressourcen auf andere Rechenknoten.
$Max_{vFPGA-Slots}$	Anzahl der vFPGA-Slots auf einem physischen FPGA.	ρ_{Knoten}	Auslastung eines Rechenknotens.
N_{Agg}	Anzahl der aggregierten vFPGA-Slots.	$\vec{\rho}_{FPGA}$	Beschreibung der FPGA-Ressourcen (Slice Register, Slice LUTs, BRAM Tiles, DSPs).
$N_{Frontends}$	Anzahl der Frontends.	$\vec{\rho}_{DDR3}$	FPGA-Ressourcen des DDR3-Controllers innerhalb der RC2F-Infrastruktur.
N_{Kerne}	Anzahl der erforderlichen Prozessorkerne auf dem Rechenknoten.	$\vec{\rho}_{Ethernet}$	FPGA-Ressourcen des Ethernet-Controllers innerhalb der RC2F-Infrastruktur.
N_{table}	Anzahl der Einträge in einer Seitentabelle innerhalb der Virtualisierung des Speichers.	$\vec{\rho}_{FPGA-Hypervisor}$	FPGA-Ressourcen des FPGA-Hypervisors innerhalb der RC2F-Infrastruktur.
$N_{vFPGA-Images}$	Anzahl der zu erzeugenden vFPGA-Images.	$\vec{\rho}_{ppr}$	FPGA-Ressourcen der Partition Pin Region (PPR).
$N_{vFPGA-Slots}$	Anzahl der (aggregierten) vFPGA-Slots.	$\vec{\rho}_{FPGA-Slot}$	FPGA-Ressourcen eines einzelnen vFPGA-Slots.
$N_{Waiting}$	Anzahl wartender Arbeitspakete bis zur Aktivierung weiterer Rechenknoten.	$\vec{\rho}_{K1}$	FPGA-Ressourcen des konstanten Anteils der linearen Regression.
$N_{Warteschlange}$	Länge der Warteschlange für eingehende Arbeitspakete.	$\vec{\rho}_{K2}$	FPGA-Ressourcen der Steigung der linearen Regression.
$P_{CPU-Last}$	Leistungsaufnahme (in Watt) des Host-Prozessors unter Vollast innerhalb RC3E-Simulation.	$\vec{\rho}_{PCIe}$	FPGA-Ressourcen des PCIe-Controllers innerhalb der RC2F-Infrastruktur.
$P_{CPU-Leerlauf}$	Leistungsaufnahme (in Watt) des Host-Prozessors im Leerlauf innerhalb RC3E-Simulation.	$\vec{\rho}_{RC2F-Infrastruktur}$	FPGA-Ressourcen der statischen RC2F-Infrastruktur.
$P_{FPGA-Last}$	Leistungsaufnahme (in Watt) des physischen FPGAs unter Vollast innerhalb RC3E-Simulation.	$\vec{\rho}_{RC2F-Statisch}(N_{vFPGA-Slots})$	FPGA-Ressourcen des RC2F-Prototypen innerhalb des statischen Bereiches.
$P_{FPGA-Leerlauf}$	Leistungsaufnahme (in Watt) des physischen FPGAs im Leerlauf innerhalb RC3E-Simulation.		
P_{Kern}	Leistungsaufnahme (in Watt) eines Rechenkerns innerhalb des RC3E-Simulators.		

Verzeichnis der Formelzeichen

$\vec{\rho}_{vFPGA}(N_{vFPGA-Slots}, N_{Frontends})$	FPGA-Ressourcen eines (aggregierten) vFPGAs in Abhangigkeit von vFPGA-Slots und Frontends.	$t_{Migration}$ Zeitspanne, nach welcher eine vFPGA-Ressource auf einen anderen Rechenknoten migriert wird.
$\vec{\rho}_{vFPGA-Slots}(N_{vFPGA-Slots})$	FPGA-Ressourcen fur eine bestimmten Anzahl an vFPGA-Slots.	$t_{Migration-FPGA}$ Zeitdauer, welche fur die vollstandige Migration einer vFPGA-Instanz auf einen anderen Rechenknoten benotigt wird.
S_{page}	Die Groe der einzelnen Seiten bei der Virtualisierung des Speichers.	t_{place} Laufzeit zum Platzieren eines vFPGA-Designs.
S_{phy}	Die Groe des externen physischen Speichers (RAM).	$t_{relocate}$ Laufzeit zum Verschieben der Komponenten.
S_{table}	Speicherbedarf einer Seitentabelle innerhalb der Virtualisierung des Speichers.	t_{route} Laufzeit zum Routen der Komponenten.
$S_{vFPGA-Image}$	Dateigroe eines vFPGA-Images.	t_{SLA} Zeitspanne bis zum Beginn der Bearbeitung der Arbeitspakete.
S_{vRAI}	Dateigroe einer vRAI.	t_{VCBM} Laufzeit zum Generieren der Virtual Context Bit Mask (VCBM).
$t_{Arbeitspaket}$	Gesamtaufzeit zur Bearbeitung eines Arbeitspaketes auf vFPGA und Rechenknoten.	$t_{vFPGA-Image}$ Laufzeit zum Erzeugen des vFPGA-Images (Bitstream).
$t_{extract}$	Laufzeit zum Extrahieren der Positionen von FPGA-Ressourcen.	t_{vRAI} Laufzeit zur Erzeugung einer vRAI.
t_{Knoten}	Zeitspanne, welche der Rechenknoten aktiv (im Leerlauf) ist.	$W_{Arbeitspaket}$ Energiebedarf (in kWh) eines Arbeitspaketes innerhalb des RC3E-Simulators.
$t_{Knoten-Start}$	Zeitdauer zum Aktivieren eines weiteren Rechenknotens.	W_{Cloud} Energiebedarf (in kWh) der Cloud innerhalb des RC3E-Simulators.
$t_{Knoten-Stop}$	Zeitdauer zum Stoppen eines Rechenknotens.	W_{Knoten} Energiebedarf (in kWh) eines Rechenknotens innerhalb des RC3E-Simulators.

1 Einleitung

In dieser Arbeit wird die Eignung von rekonfigurierbarer Hardware für den Einsatz in einer Cloud analysiert. Es wird untersucht wie diese spezielle Hardware als Alternative zu Prozessoren in eine moderne Cloud-Architektur eingebettet werden kann, um die Verarbeitungsgeschwindigkeit zu erhöhen, sowie den Energieverbrauch und damit die Betriebskosten zu senken. Dabei steht eine systematische Virtualisierung der Hardware im Vordergrund. Die Zielstellung besteht darin, für die Nutzer virtuelle Systemkonfigurationen entsprechend deren Bedürfnissen bereitzustellen. Die *Virtualisierung* ist eine der Schlüsseltechnologien des Cloud-Computings [Put+15] und ermöglicht so die Abstraktion von der zugrunde liegenden physischen Hardware. Sie bietet eine Möglichkeit Einschränkungen der physischen Hardware aufzuheben und damit die Flexibilität eines Rechnersystems zu erhöhen [SN05b, S. 3]. Neben der Energieeffizienz und der Beschleunigung spielt ebenfalls eine optionale Erhöhung der Zugangssicherheit durch Nutzung der Spezialhardware eine wesentliche Rolle.

1.1 Motivation

Cloud-Computing, die gleichzeitige Bereitstellung umfangreicher Rechen- und Speicherressourcen für unterschiedliche Nutzer über das Internet, findet eine immer größere Verbreitung und führt zu einer erheblichen wirtschaftlichen Bedeutung. Durch die flexible und adaptive Bereitstellung von Ressourcen und Diensten kann eine deutliche Kostensparnis auf Nutzerseite erreicht werden. Die Besonderheit besteht darin, dass die Hardwareressourcen nicht durch den Nutzer selbst physisch vorgehalten werden müssen [Brä+12]. Die Einsatzgebiete reichen hierbei von einfachen Web-Technologien und Datenspeichern über komplexe Geschäftsprozesse bis hin zu datenintensiven Anwendungen und Analysen im Bereich des Big Data oder des Internet of Things (IoT). Wachstumsraten von bis zu 30 Prozent weltweit im Jahr 2016 im Bereich der Bereitstellung von Cloud-Infrastrukturen und -Plattformen [IDC16] führen dabei zu einer stetig steigenden Konzentration an Rechenressourcen und damit auch zu steigender Rechenleistung. Der dadurch steigende Energiebedarf führt zu der Notwendigkeit des Einsatzes effizienterer Systeme und weiterer Ansätzen zur Energieeinsparung [Bal+11]. Die Bearbeitung enormer Datenmassen in der Cloud und die dazu erforderliche rasant steigende Rechenleistung, sowie der damit verbundene zunehmende Energieverbrauch, führen zu der Notwendigkeit alternative Hardware-Architekturen – abseits der traditionellen Prozessoren – in Rechenzentren zu integrieren. Eine deutliche Steigerung der Rechenleistung bei gleichzeitiger Einsparung von Energie wird durch den Einsatz heterogener Systemen mit unterschiedlichen Komponenten wie beispielsweise Field Programmable Gate Arrays (FPGAs) erreicht.

Dabei sind nach einer aktuellen Forbes-Umfrage [Rog16] Field Programmable Gate Arrays (FPGAs), rekonfigurierbare Schaltkreise zur Beschleunigung von Anwendungen, für viele Unternehmen eine Schlüsseltechnologie. Microsoft [Cau+16], IBM [IBM15a] und demnächst auch Amazon [Ama17a] setzen als weltweit führende Betreiber von Rechenzentren und Anbieter von Cloud-Infrastrukturen in ihren neuen Systemen FPGAs auf Anwendungsebene ein, um rechenintensive Aufgaben und die Beschleunigung

1 Einleitung

von Cloud-Diensten auf die Spezialhardware auszulagern. Aufgrund des deutlich geringeren Energieverbrauchs gegenüber herkömmlichen Prozessoren [Hoe16] können dadurch die Betriebskosten ebenfalls gesenkt werden. Die Übernahme von Altera, einem der größten Hersteller von FPGAs, durch den Chiphersteller Intel erfolgte auch aufgrund der wachsenden Nachfrage nach FPGAs von Seiten der Betreiber großer Rechenzentren [Rog16].

In der Vergangenheit beschränkte sich der Einsatz von FPGAs im Rechenzentrum hauptsächlich auf die Netzwerkinfrastruktur und einfachen Coprozessoren, so dass die Integration der rekonfigurierbaren Hardwarekomponenten auf der Anwendungsebene zu neuen Herausforderungen führt. Diese Art der Nutzung der Hardware ist bislang dem Betreiber des Rechenzentrums vorbehalten und nicht effizient von den Anbietern der eigentlichen Dienste nutzbar. Die Gründe dafür sind eine fehlende flexible Integration und Virtualisierung, also eine Abstraktion von der physischen Hardware der FPGAs für den Einsatz in der Ressourcenverwaltung einer Cloud-Architektur.

Eine weitere Einsatzmöglichkeit von FPGAs in der Cloud stellt neben Netzwerkinfrastruktur und Hintergrundbeschleunigung die Erhöhung der Sicherheit von Cloud-Diensten dar die mit vertraulichen Daten arbeiten. Durch den Einsatz spezieller rekonfigurierbarer Hardware bestehen deutlich weniger Angriffs möglichkeiten als bei herkömmlichen Prozessor-Architekturen, da die Software nur eine untergeordnete Rolle spielt und ein sehr effizienter Schutz der Konfiguration der FPGAs möglich ist [TM14]. Die *Konfiguration* eines FPGAs stellt dabei die Beschreibung des Verhaltens und der Struktur der Hardware, analog zum Programm bei Prozessoren, dar.

1.2 Historische Entwicklung

Der erste FPGA wurde im Jahr 1984 von Xilinx vorgestellt und ab dem Jahr 2000 wurden FPGAs als Standardkomponenten in vielen digitalen Systemen eingesetzt. Aufgrund ihrer großen Rechenkapazität wurden sie ab 2008 zunehmend für die Beschleunigung von wissenschaftlichen Anwendungen interessant [Tri15]. Seit 2013 beschäftigen sich vermehrt Forschungsarbeiten mit FPGAs in Cloud-Architekturen, wie beispielsweise Putnam et al. [Put+14], welche FPGAs in bestehende Rechenzentren eingliedern, um die Verarbeitungsleistung zu steigern, oder auch Chen et al. [Che+14] und Byma et al. [Bym+14], welche versuchen, FPGAs als Ressourcen in eine bestehende Cloud-Verwaltung einzubinden.

Die Integration von FPGAs in eine Cloud erfordert neben der physischen Kopplung der Hardware mit bestehenden Prozessoren auch eine Einbettung in das Betriebssystem auf dem Host-System. Die Virtualisierung ist dabei eine der wichtigsten Technologien, auf denen der Bereich des Cloud-Computing aufbaut [Plu+08]. Ein bedeutender Aspekt der Forschung ist dabei die Virtualisierung der physischen Ressource FPGA um mehrere konkurrierende Nutzer zeitgleich auf der selben physischen Hardware arbeiten zu lassen.

Ein dabei nicht zu vernachlässigender Aspekt ist die Virtualisierung der Ressource FPGA selbst und stellt ein über Jahre immer wiederkehrendes Thema in der Literatur dar. Erste Gedanken zur Notwendigkeit einer Virtualisierung von FPGAs veröffentlichten Fornaciari et al. [FP98b] mit dem Ziel, große Anwendungen auf kleinen Schaltkreisen zu realisieren. Mittlerweile werden in der Forschung Wege gesucht die jetzt verfügbaren, großen FPGAs mit unterschiedlichen Nutzern nebenläufig auszulasten. Eine Virtualisierung von FPGAs mit dem Ziel, auf derselben physischen Hardware unterschiedliche Nutzer zu etablieren, ist dabei Gegenstand der Arbeiten von El-Araby et al. [EGE08], Wang et al. in [WBP13] oder Fahmy et al. [FVS15].

Die Abgrenzung meines Forschungsvorhabens zu vergleichbaren Arbeiten ist hierbei die umfangreiche Untersuchung unterschiedlicher Ansätze der Virtualisierung aus den klassischen Ansätzen zur Hardware-, System- oder Treiber-Virtualisierung. Das wesentliche Alleinstellungsmerkmal besteht in einer Übertragung der konventionellen System-Virtualisierung, wie sie auch für Betriebssysteme eingesetzt wird, auf die rekonfigurierbare Hardware. Ausgangspunkt bildet eine prototypische Implementierung mit Abgrenzung und Isolierung der vFPGAs voneinander. Weitere wichtige Punkte sind ein notwendiges Konzept zur Verwaltung von zusammengehörigen vFPGAs und virtuellen Maschinen in einem dynamischen Cloud-System mit flexilem Zugang zu den vFPGAs. Auf diese Weise wird sowohl die Beschleunigung von Cloud-Diensten, als auch der Einsatz des FPGAs für sicherheitskritische Anwendungsfälle ermöglicht, worin auch der wesentliche Beitrag der Arbeit zum Wissenschaftsgebiet liegt.

1.3 Zielsetzung und Beitrag

Das Ziel dieser Arbeit ist es zu analysieren, wie rekonfigurierbare Hardware-Komponenten im Kontext von Cloud-Architekturen eingesetzt und sinnvoll genutzt werden können. Um jetzt verfügbaren sehr großen FPGAs effizient auslasten zu können, ist ähnlich wie bei anderen Komponenten in der Cloud, eine Virtualisierung der FPGAs und die Einbettung in eine Verwaltungsstruktur erforderlich. Die zentrale Idee lässt sich wie folgt zusammenfassen:

Evaluation eines flexiblen Einsatzes von FPGAs in der Cloud durch Virtualisierung der Ressourcen zur Bereitstellung einer adaptiven und flexiblen Architektur für unterschiedlichste Anwendungsfälle.

Die Schwerpunkte liegen dabei auf einer flexiblen Verwaltung der FPGAs und deren dynamischer Rekonfiguration, einer Virtualisierung der Hardware zur Hintergrundbeschleunigung und der Erhöhung der Sicherheit durch einen direkten Zugang zur dynamisch rekonfigurierbaren Hardware. Eine Virtualisierung der FPGAs, welche eine optimale Auslastung der Ressourcen ermöglicht, bietet dabei größtmögliche Flexibilität durch die Nutzung der dynamischen partiellen Rekonfiguration. Nach Vergleich unterschiedlicher Ansätze erfolgt die Virtualisierung in der Arbeit analog zur System-Virtualisierung mit der Möglichkeit, mehrere Nutzer auf der physischen Ressource gleichzeitig zuzulassen und sicher voneinander zu kapseln. Die virtualisierten FPGAs sollen an die Bedürfnisse der Nutzer adaptiert werden und eine höhere Mobilität als bisher bieten, um die Ressource FPGA effizienter auszunutzen. Neben einer prototypischen Implementierung wird ebenso ein Ausblick gegeben, wie zukünftige virtualisierte FPGAs für den Cloud-Einsatz beschaffen sein müssen, um effizient und flexibel eingesetzt werden zu können.

Des Weiteren sollen die FPGAs aus Nutzersicht durch Virtualisierung von ihrer physischen Position im Rechenzentrum losgelöst werden und für unterschiedliche Szenarien und Anwendungsfälle eingesetzt werden können. Auf diese Weise ist es möglich, den Betreiber des Rechenzentrums von den eigentlichen Anbietern eines Dienstes zu entkoppeln, sodass FPGAs flexibel wie andere virtualisierte Ressourcen in der Cloud eingesetzt werden können.

Ziel der Arbeit ist nicht der Aufbau einer Cloud-Architektur, sondern vielmehr die Untersuchung ausgewählter Aspekte im Hinblick auf die Integration von rekonfigurierbarer Hardware. Der Neuigkeitswert besteht in der Untersuchung der Integration und Verwaltung virtueller FPGAs auf unterschiedlichen Abstraktionsebenen in einem Cloud-System. Dabei wird nicht einfach nur der FPGA virtualisiert sondern, anders als bei vielen anderen Arbeiten, das Gesamtsystem und der Einsatzzweck berücksichtigt und die vFPGAs werden, je nach Anforderung, an unterschiedlichen Positionen in der Cloud dynamisch und adaptiv eingesetzt.

1.4 Herangehensweise und Strukturierung der Arbeit

Um sich der Problemstellung anzunähern, wird zu Beginn in Kapitel 2 der Stand der Forschung auf den relevanten Gebieten vorgestellt. Unterschiedliche Hardwarebeschleuniger und insbesondere rekonfigurierbare Architekturen, sowie deren Kopplungsmöglichkeiten mit Prozessoren werden analysiert. Des Weiteren werden zunächst die Begriffe *Virtualisierung* und *Cloud-Computing* definiert und aktuelle Entwicklungen in diesen Bereichen aufgezeigt, um anschließend vergleichbare Arbeiten aus beiden Bereichen im Kontext zu Hardwarebeschleunigern auf FPGA-Basis zu betrachten. Anschließend endet das Kapitel mit einer Literaturanalyse und der sich daraus ableitenden Zielstellung.

Im Hauptteil der Arbeit werden zu Beginn in Kapitel 3 unterschiedliche Abstufungen für den Einsatz von rekonfigurierbaren Ressourcen für Cloud-Dienste herausgearbeitet und entsprechend ihrer Flexibilität charakterisiert. Möglichkeiten der Kopplung zwischen mehreren virtualisierten FPGAs auf einem physischen FPGA und Virtuellen Maschinen auf dem Host-System werden in Kapitel 4 analysiert. Nachdem hier der vorliegende Entwurfsraum aufgezeigt wird und unterschiedliche Konzepte der Virtualisierung mit Hinblick auf klassische System-Virtualisierungen gegenübergestellt werden, wird schließlich eine prototypische Architektur für eine FPGA-Virtualisierung herausgearbeitet. Das Ziel besteht hierbei darin, mehrere unabhängige, nebenläufig arbeitende Nutzerkerne auf demselben physischen FPGA zu realisieren und durch Migration auf andere physische Rechenknoten zu übertragen, sowie von der physischen Größe und der Architektur des FPGAs weitestgehend zu abstrahieren. Kapitel 5 zeigt die Möglichkeiten zur Integration der zuvor virtualisierten FPGAs in eine Cloud-Architektur auf. Die Nutzung des FPGA soll sich dabei nicht auf die Anwendung als einfacher Hardwarebeschleuniger zur Auslagerung rechenintensiver Anwendungen beschränken, sondern dieser soll sich auch virtuell an unterschiedlichen Punkten im System einbetten lassen können. Somit sollen mit Hilfe der Virtualisierung der Architektur auch sicherheitsrelevante Anwendungen, wie ein direkter verschlüsselter Zugang durch den FPGA zu gekapselten Diensten auf den Virtuellen Maschinen des Hosts, ermöglicht werden.

Um die Lösungsansätze bewerten zu können, werden in Kapitel 6 die zuvor beschriebenen Konzepte in ein prototypisches Cloud-System eingebettet. Dabei wird insbesondere die Virtualisierung der Hardware realisiert, um Aussagen zur Machbarkeit geben zu können und erzielbare Leistungsdaten zu bestimmen. In einem Anwendungsszenario wird gezeigt, wie die physischen FPGA-Ressourcen an virtualisierte Nutzerdesigns angepasst werden und wie eine Migration die Auslastung des physischen FPGAs optimieren kann. Die Anforderungen durch eine Etablierung von homogenen und insbesondere migrierbaren FPGA-Designs auf aktuellen FPGAs und Optimierungsansätze für zukünftige, virtualisierte FPGA-Architekturen für den Einsatz in der Cloud werden anhand einer Simulation aufgezeigt.

Im letzten Kapitel werden schließlich die wesentlichen Ergebnisse der Arbeit zusammengefasst und in Hinblick auf die Zielstellungen bewertet. Abschließend wird ein Ausblick auf mögliche Weiterentwicklungen in dem Forschungsgebiet und potentielle zukünftige Arbeiten gegeben.

2 Stand der Forschung und Technik

Die Untersuchung der Möglichkeiten einer flexiblen Integration von spezieller rekonfigurierbarer Hardware wie FPGAs in ein Mehrbenutzersystem oder eine Cloud-Architektur erfordert zunächst die Betrachtung der relevanten Teilgebiete. Der Schwerpunkt liegt dabei auf einheitlichen Begriffsdefinitionen und den aktuellen Forschungsschwerpunkten auf den einzelnen Teilgebiete.

Abschnitt 2.1 stellt zunächst rekonfigurierbare Hardwarebeschleuniger auf Basis von FPGAs, sowie deren Möglichkeiten vor. Da für die effiziente und flexible Integration von Hardware in eine Cloud eine Virtualisierung der Hardware unerlässlich ist, werden die Grundlagen und Konzepte der Virtualisierung in Abschnitt 2.2 erläutert. Darauf aufbauend gibt Abschnitt 2.3 einen Überblick zu den Grundlagen des Cloud-Computing und der Bereitstellung, Integration und Verwaltung von Hardwareressourcen (Prozessor, Speicher etc.). Verwandte Arbeiten aus den Themengebieten der Virtualisierung und der Bereitstellung von FPGA-Ressourcen in einer Cloud werden auf Basis der eingeführten Grundlagen in Abschnitt 2.4 vorgestellt. Abschnitt 2.5 ordnet daraufhin die vorliegende Arbeit in die bestehende Literatur ein und hebt deren Beitrag zum Wissenschaftsgebiet hervor.

2.1 Hardwarebeschleuniger auf Basis rekonfigurierbarer Hardware

Im folgenden Abschnitt 2.1.1 werden zunächst verschiedene derzeitig eingesetzte Hardwarebeschleuniger vorgestellt, bevor zunächst in Abschnitt 2.1.2 auf die grundlegende Architektur von FPGAs sowie auf Kopplungsarten mit Prozessoren eingegangen wird. Abschnitt 2.1.3 erläutert typische Einsatzgebiete für FPGAs. In Abschnitt 2.1.4 wird darauf aufbauend der Ablauf zur Erzeugung eines Bitstreams (Konfiguration) für einen FPGA erläutert und auf aktuelle Entwicklungen eingegangen.

2.1.1 Arten von Hardwarebeschleunigern und deren Leistungsbewertung

Die ersten Hardwarebeschleuniger, beziehungsweise Coprozessoren (siehe Definition 2.1), gehen zurück bis zu den ersten Gleitkomma-Coprozessoren oder auch Direct Memory Access (DMA) Speicher-controllern des von Neumann Rechners [Wol+84]. Aufgrund dessen hoher Flexibilität, jedes beliebige Programm abarbeiten zu können, ist der Hauptprozessor, die Central Processing Unit (CPU), in ihrer Leistungsfähigkeit eingeschränkt und erreicht nicht annähernd die Geschwindigkeit eines spezialisierten Coprozessors, welcher nur für eine bestimmte Klasse von Anwendungen oder Algorithmen entworfen wurde.

Definition 2.1: Coprocessor *Additional processor used in some personal computers to perform specialized tasks such as extensive arithmetic calculations or processing of graphical displays. The coprocessor is often designed to do such tasks more efficiently than the main processor, resulting in far greater speeds for the computer as a whole¹. [Bri16]*

¹Freie Übersetzung von [Bri16]: Zusätzlicher Prozessor, der in Computern verwendet wird um spezialisierte Aufgaben, wie umfangreiche arithmetische Berechnungen oder die Verarbeitung von grafischer Darstellungen. Da der Coprozessor speziell für diese Aufgaben ausgelegt ist, arbeitet er deutlich effizienter als der Hauptprozessor, was in einer deutlich höheren Verarbeitungsgeschwindigkeit des gesamten Rechensystems resultiert.

2 Stand der Forschung und Technik

Dabei ist zu unterscheiden zwischen einem Coprozessor, welcher Instruktionen direkt von einer eng gekoppelten CPU erhält und eine feste Firmware oder einen Mikrocode ausführt und einem Hardwarebeschleuniger, der sein eigenes, veränderbares Programm abarbeitet und lediglich Programm, und Daten von einer CPU erhält, beziehungsweise die Ergebnisse an diese übermittelt. Hardwarebeschleuniger sind demnach nicht zwangsläufig so eng mit der CPU gekoppelt wie Coprozessoren. Der Schwerpunkt dieser Arbeit liegt dabei auf Hardwarebeschleunigern zur Auslagerung von rechenintensiven Anwendungen, welche eine Steigerung der Verarbeitungsgeschwindigkeit durch Ausnutzung massiver Parallelität erreichen.

Coprozessoren zur Ausführung von rechenintensiven arithmetischen Operationen haben sich von einfachen Gleitkomma-Coprozessoren wie dem Intel 8087, welcher im Jahr 1980 vorgestellt wurde [Mue03, S. 115], bis hin zu zunehmend massiv-parallelen Beschleunigern für Single Instruction Multiple Data (SIMD) Gleitkomma-Berechnungen, wie dem ClearSpeed Coprozessor [Pro11], weiterentwickelt. Neben solchen Coprozessoren wurden zunehmend Hardwarebeschleuniger wie beispielsweise IBMs Cell Broadband Engine (CBE) [Kah+05] entwickelt. Die Architektur besteht aus einer PowerPC CPU und bis zu acht speziellen Rechenkernen. Ursprünglich war die CBE nur für die Sony Playstation 3 vorgesehen, wurde aber aufgrund der hohen Rechenleistung auch für die Beschleunigung von wissenschaftlichen Anwendungen interessant, was zu ihrer Weiterentwicklung und schließlich einem professionellen Einsatz als IBM PowerXCell 8i im HPC-Bereich im Jahr 2008 führte [Bar+08].

Neben diesen speziellen Architekturen rückten ab dem Jahr 2000 immer wieder FPGAs als frei konfigurierbare und energieeffiziente Hardwarebeschleuniger für spezielle Anwendungen in den Fokus [VB13], konnten sich aber als Beschleuniger trotz ihrer großen Flexibilität nicht etablieren. Der Grund bestand in der vergleichsweise geringen Rechenleistung für Gleitkommaarithmetik aufgrund derer FPGAs weiterhin lediglich für spezielle kryptografische und Festkommaarithmetik eingesetzt wurden [VB13, S. VI]. Mit dem Erreichen der Taktobergrenze von Prozessoren und dem Übergang zu parallelen Architekturen wurden FPGAs zunehmend für spezielle hochparallele Berechnungen eingesetzt. Ab 2007 wurden zudem Grafikkarten (GPUs) aufgrund des Übergangs von der ursprünglichen Grafikpipeline hin zu vollwertigen Rechenkernen und des Compute Unified Device Architecture (CUDA)-Programmiermodells von Nvidia, welches keine Kenntnisse über Grafikprogrammierung erforderte, vermehrt als Hardwarebeschleuniger mit mehreren Tausend einfachen Rechenkernen eingesetzt [KW12]. Des Weiteren haben sich in den letzten Jahren Manycore-Architekturen wie der Intel Xeon Phi als weitere Alternative etabliert.

2.1.1 Aktuelle Hardwarebeschleuniger und ihre Charakteristiken

Abbildung 2.1 gibt einen Überblick zu den aktuellen Hardwarebeschleunigern und der ihr zugrunde liegenden Architektur, wobei diese nach der Komplexität der Rechenkerne, sowie deren Anzahl eingetragen sind. Ausgehend vom klassischen Prozessor und der weiterhin nach Moore's Law² steigenden Anzahl an verfügbaren Transistoren auf einem Chip bestand die zwangsläufige Konsequenz zur weiteren Steigerung der Rechenleistung in einer Vervielfältigung der Prozessorkerne in Richtung einer vollständigen Multicore-Architektur [Gra03, S. 2].

Hardwarebeschleuniger, welche typischerweise eng mit einem Hauptprozessor gekoppelt sind (siehe Abbildung 2.2) und diesen für die Programmausführung benötigen, können in die drei Gruppen Manycore, Grafikkarte und FPGA unterteilt werden. Der Hauptprozessor ist dabei notwendig um die Daten sowie das auszuführende Programm an den Beschleuniger zu senden, die Verarbeitung durch einen

²Das Gesetz, welches Gordon Moore 1965 formulierte [Moo65], besagt, dass sich die Komplexität integrierter Schaltkreise regelmäßig verdoppelt. Unterschiedliche Quellen geben 18 bis 24 Monate als Zeitraum für die Verdoppelung an [SW04].

2.1 Hardwarebeschleuniger auf Basis rekonfigurierbarer Hardware

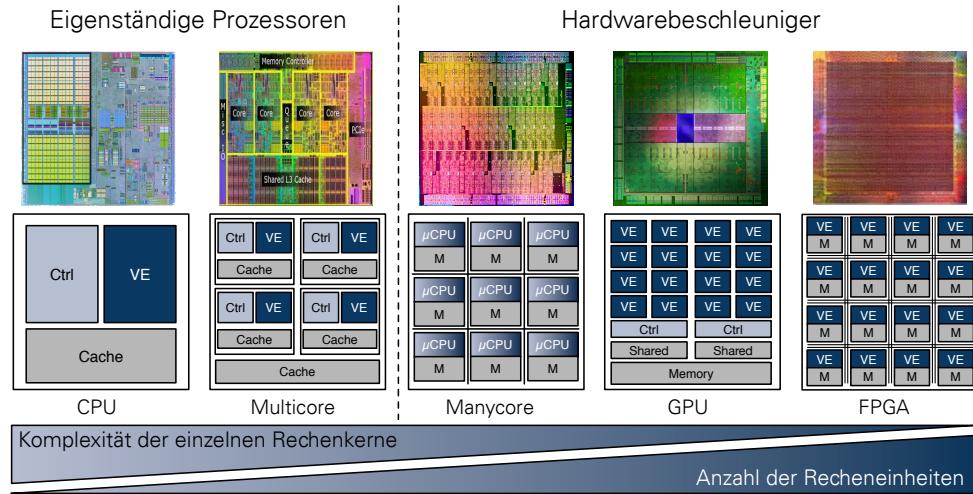


Abbildung 2.1: Auswahl von Hardwarebeschleuniger-Architekturen mit Einordnung nach Anzahl der nebenläufigen Rechenkerne und deren Komplexität, beginnend bei einem einfachen Prozessor mit Steuerwerk (Ctrl) und Verarbeitungseinheit (VE). Nach [Kno14].

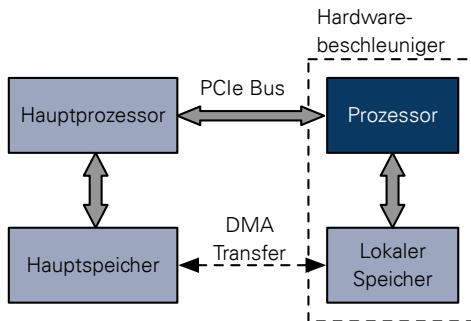


Abbildung 2.2: Typische Kopplung zwischen Hardwarebeschleuniger und Hauptprozessor.

Funktionsaufruf zu starten und die Ergebnisse nach der Berechnung wieder zu empfangen. Der Zugang und die Kommunikation mit dem Beschleuniger werden über eine API gewährleistet, welche die erforderlichen Datentransfers und insbesondere den Funktionsaufruf (Kernelaufruf) der auf den Beschleuniger ausgelagerten Funktion ermöglicht. Typische aktuelle Hardwarebeschleuniger sind unter anderem:

Manycore-Architekturen: Eine Manycore-Architektur besteht aus einer deutlich höheren Anzahl von Rechenkernen als eine Multicore-Architektur. Die Rechenkerne besitzen einen eigenen Kontrollfluss mit eigenständiger Abarbeitung eines oftmals gemeinsamen Programms. In Anlehnung an Flynn [Fly72] kann die Architektur als Single Programm Multiple Data (SPMD) bezeichnet werden. Ein Beispiel aus dieser Gruppe ist der Intel Xeon Phi [Int16c] mit bis zu 72 Rechenkernen, welche komplex und vielseitig sind, so dass eine Programmierung mit beispielsweise OpenMP [Dag98] genauso möglich ist wie mit Open Computing Language (OpenCL) [Mun+09].

Grafikkarten (GPGPUs): Eine deutliche Steigerung der Anzahl der Rechenkerne durch massive SIMD-Parallelität ist mittels moderner Grafikkarten, den sogenannten General Purpose Graphics Processing Units (GPGPUs) möglich. Hierbei sind die Rechenkerne allerdings deutlich weniger komplex und in Multiprozessoren zusammengefasst, welche ein Programm auf vielen parallelen Rechenkernen ausführen. Eine CUDA-fähige Grafikkarte gliedert sich dementsprechend in mehrere Blöcke (welche aus Hardware-Sicht Multiprozessoren entsprechen) und diese wiederum in unterschiedliche Threads (welche auf den Rechenkernen oder Thread-Prozessoren abgearbeitet wer-

2 Stand der Forschung und Technik

Tabelle 2.1: Überblick über Arten von Hardwarebeschleunigern mit theoretischer Floating-Point Rechenleistung in SP (einfache Genauigkeit) oder DP (doppelte Genauigkeit) und dem maximalen Energieverbrauch TDP.

Beschleuniger	GFlops	Taktrate	Prozess	TDP	Kerne	Cache/OCRAM
Xilinx 7 Series (Virtex-7 690t [Xil16c; Xil16i])	833 SP	438 MHz	28 nm	20 - 30 W ^a	3.600 DSPs	8,6 MByte
Xilinx UltraScale+ (VU9P [Xil16c; Xil16i])	4.600 SP	438 MHz	16 nm	20 - 50 W ^a	6.840 DSPs	48 MByte
Intel Stratix 10 (GX 2800 [Int16b])	9.200 SP	450 MHz	14 nm	–	5.760 DSPs	30 MByte
Nvidia Tesla (P100 SP [NVI16])	9.300 SP	1.328 MHz	16 nm	300 W	3.584 Cores	18 MByte
Intel Xeon Phi (7290F [Int16c])	1.208 DP	1.500 MHz	14 nm	260 W	72 Cores	36 MByte
Intel Core i7 (5960X [Int16a])	384 DP	3.000 MHz	14 nm	140 W	8 Cores	20 MByte

^aTDP auf Basis von [Int16b; Xil16c; Xil16h] mit 20 Watt zusätzlich für on-Board Peripherie [KLS16].

den) [KW12]. Die Verarbeitung wird demnach in Anlehnung an Flynn [Fly72] als Single Program Multiple Thread (SPMT) bezeichnet. Die Rechenkerne haben hierbei eine geringere Komplexität, allerdings ist deren Anzahl mit bis zu 3.584 CUDA-Kernen in einer Nvidia Tesla P100 [NVI16] deutlich höher als bei sonst üblichen Manycore-Architekturen. Die Programmierung erfolgt über Sprachen wie zum Beispiel OpenCL [Mun+09] oder CUDA [KW12] mit einer expliziten Kennzeichnung der Parallelität innerhalb des Funktionsaufrufes über die beiden Ebenen Blöcke und Threads, sowie der auf den Beschleuniger ausgelagerten Funktionen (Kernel).

Hardwarebeschleuniger auf Basis rekonfigurierbarer Hardware (FPGAs): Die Nutzung von FPGAs als Hardwarebeschleuniger erfolgte nach Trimberger [TM14] ab etwa 2005 durch die Verfügbarkeit immer größerer FPGAs und deutlich sinkender Preise, welche sie für Anwendungen fernab des ursprünglichen Application Specific Integrated Circuits (ASIC)-Prototyping interessant machten. Im Jahr 2008 stellte Convey mit dem HC-1 [Bak10] ein leistungsfähiges hybrides System mit einem FPGA als Coprozessor vor, wodurch in der Folge FPGAs immer wieder als Hardwarebeschleuniger in Betracht gezogen wurden sich aber aufgrund einer vergleichsweise aufwändigen Programmierung bislang nicht etablieren konnten [KHZ16]. Aus der Realisierbarkeit mehrerer Tausendspezifischer Rechenkerne und deren Anpassbarkeit an die Anforderungen der konkreten Anwendung resultiert eine hohe mögliche Verarbeitungsleistung [KPS11; VB13]. Die Architektur von FPGAs und typische Anwendungen werden detailliert in Abschnitt 2.1.2 erläutert. Die Programmierung, beziehungsweise Konfiguration der FPGAs wird in Abschnitt 2.1.4 betrachtet.

Die oben vorgestellten Beschleuniger sind darauf ausgelegt, hochparallele Berechnungen durchzuführen und verfügen dazu in der Regel über mindestens zwei Ebenen der Parallelität in sowohl der Hardware als auch im Kernel-Programm. Dies sind einerseits eine grobgranulare Aufteilung der Arbeitspakete auf die Blöcke oder Multiprozessoren und des Weiteren eine feingranulare Unterteilung in Threads, welche auf den eigentlichen Rechenkernen ausgeführt werden. Um Plattformunabhängigkeit zu ermöglichen wurde zum Beispiel die Sprache OpenCL für die Programmierung von heterogenen Architekturen entwickelt.

2.1.1.2 Leistungsbewertung von Hardwarebeschleunigern

Als Überblick über ausgewählt Beschleuniger zeigt Tabelle 2.1 wichtige Leistungsdaten für Hardwarebeschleuniger. Die Anzahl der nebenläufigen Rechenkerne ist dabei eine wesentliche Kennzahl für die erreichbare Parallelität. Die vorgestellten Beschleuniger sind dabei in der Regel mittels Peripheral Component Interconnect Express (PCIe) mit dem Host-System verbunden. Im Folgenden werden zunächst unterschiedliche Kriterien zur Bewertung von Hardwarebeschleunigern und ähnlichen parallelen Architekturen aufgezeigt.

Theoretisch erreichbare Rechenleistung – FLOPS und GMACS

Die Leistungsfähigkeit von Prozessoren und Hardwarebeschleunigern kann als einfacher Ausdruck von Arbeit pro Zeiteinheit anhand der Gleitkommaoperation GFLOPS beurteilt werden [Kru09]. Dabei muss zwischen theoretischer Rechenleistung, welche mit der Architektur erreicht werden kann, und der für eine konkrete Anwendung erreichten Leistung unterschieden werden. Ausgewählte Vertreter der vorgestellten Hardwarebeschleuniger sind in Tabelle 2.1 mit Gleitkommaoperationen, Verlustleistung (TDP) und weiteren relevanten Kennzahlen aufgelistet, um einen Vergleich zu ermöglichen. Die Gleitkomma-Rechenleistung ist für eine Vielzahl von wissenschaftlichen Anwendungen mit starkem numerischen Anteil aus Physik, Chemie und Biologie von großer Bedeutung [Kru09]. Tabelle 2.1 zeigt, dass FPGAs eine geringere Taktrate und einen deutlich geringeren Energieverbrauch als andere Hardwarebeschleuniger aufweisen, aber dennoch über eine hohe theoretische Gleitkomma-Rechenleistung verfügen. Wird die theoretisch erreichbare Rechenleistung pro Watt betrachtet, zeigt sich eine höhere Effizienz von FPGAs gegenüber GPGPUs³.

Eine weitere Metrik, welche neben CPUs und GPGPUs insbesondere auch für DSPs und FPGAs genutzt wird, ist die Giga Multiply-Accumulate (GMAC)-Leistung auf Basis der insbesondere die für Matrix- und Vektorarithmetik wichtigen Multiply-Add Operation. In Abschnitt A.1.2 wird des Weiteren die Entwicklung GMAC-Leistung näher betrachtet. Aufgrund der unterschiedlichen Architekturen und typischen Anwendungsbereiche ist eine Bewertung anhand von Rechenkernen schwierig, da beispielsweise auf einem FPGA Rechenkerne nur indirekt in Form der DSPs zur Verfügung stehen, welche eine deutlich höhere Rechenleistung im Festkomma-, als im Gleitkommaformat besitzen.

Ausführungszeit, Speedup, Energieverbrauch und Entwicklungszeit

Neben den nur theoretisch erreichbaren Leistungsdaten sind reale Leistungsdaten für konkrete Anwendungen, beispielsweise Benchmarks wie Linpack für Gleitkommaoperationen [DLP03] oder das Standard Performance Evaluation Corporation (SPEC) [Hen00] Benchmark, von großer Bedeutung, um Architekturen im Kontext ihres Einsatzgebietes bewerten zu können.

Sind keine Benchmarks vorhanden oder bilden diese nicht die benötigten Anwendungen zum Vergleich ab, ist als ein weiteres Kriterium, insbesondere zur Bewertung von Hardwarebeschleunigern, die Ausführungszeit sowie der erzielte Geschwindigkeitsgewinn gegenüber einer Vergleichsimplementierung auf einem Prozessor heranzuziehen.

Bei einer Parallelisierung durch unabhängige Verarbeitungseinheiten innerhalb des Beschleunigers oder durch mehrere Beschleuniger können dabei die klassischen Ansätze zur Abschätzung und Berechnung eines Speedups von Amdahls [Amd67] und Gustafson [Gus88] genutzt werden.

Ein wichtiges Kriterium stellt besonders für FPGAs neben der Leistungsfähigkeit der Energieverbrauch und entsprechend die Rechenleistung pro Watt dar. Beispielsweise zeigen Fowers et al. in [Fow+13], dass FPGAs eine Fourier-Transformation besonders energieeffizient berechnen können und Che et al. zeigen in [Che+08] die hohe Rechenleistung für einfache streaming-orientierte Operationen wie Ver- und Entschlüsselung. Für Anwendungen aus der Biologie erreichen FPGAs aufgrund hoher Parallelität und der dort genutzten einfachen Datentypen ebenfalls hohe Verarbeitungsgeschwindigkeiten, wie in [Hou+16; KDW10; KPS11; Ols+12] gezeigt wurde. Im Bereich der Bildverarbeitung übertreffen FPGAs in ausgewählten Szenarien GPGPUs, wie bei der Clusterung mit Hilfe eines k-Means Algorithmus in der

³Aufgrund der hohen Anschaffungskosten von FPGAs mit mehreren tausend DSPs bieten GPGPUs eine deutlich bessere Effizienz von Rechenleistung bezogen auf Anschaffungskosten [Ber16].

Tabelle 2.2: Qualitativer Vergleich von unterschiedlichen Hardwarebeschleunigern. Nach [Gup15; Put16].

	Rechenleistung	Energieverbrauch	Komplexität	Flexibilität	Entwicklungszeit	Sicherheit
CPU	x	!	✓	✓	✓	x
ASIC	✓	✓	✓	x	x	✓
FPGA	✓	✓	!	✓	x	✓
GPU	✓	x	!	!	!	!

✓: optimal !: eingeschränkt x: problematisch

Arbeit von Hussain et al. [Hus+11]. Für komplexere Anwendungen mit vielen Gleitkommaoperationen eignen sich GPGPUs deutlich besser, wie von Gac et al. [Gac+08] oder Li et al. [LSS11] gezeigt wurde. In [Wil+10] haben Williams et al. dargestellt, dass FPGAs zwar eine hohe Rechenleistung für Festkomma-Operationen aufweisen, aber die Gleitkomma-Rechenleistung bei GPGPUs höher liegt.

Nur wenige Arbeiten ziehen zur Bewertung die Entwicklungszeit für eine Anwendung hinzu, da diese oft nur schwer messbar ist. Jones et al. [Jon+10] und Merchant et al. [Mer+08] zeigen, dass im Vergleich zu anderen Plattformen wie Multicore und GPU die Entwicklungszeit bei rekonfigurierbarer Hardware deutlich höher liegt. FPGAs stellen für bestimmte Anwendungen zwar eine ideale Architektur hinsichtlich Rechenleistung, Latenz und Energieverbrauch dar, aber die Entwicklungskosten für Anwendungen liegen in der Regel deutlich höher, auch wenn mit der High-Level Synthesis (HLS) und OpenCL für FPGAs diese zu Lasten der Leistungsfähigkeit reduziert werden kann [Ska+13].

2.1.1.3 Abschließende Betrachtung von Hardwarebeschleunigern

Tabelle 2.2 zeigt einen qualitativen Vergleich zwischen den Beschleunigern und einem Prozessor als Vergleichsbasis, um die unterschiedlichen Arten von Hardwarebeschleunigern besser charakterisieren zu können. Die in Abschnitt 2.1.1.2 betrachteten Kriterien fließen in den Vergleich ebenso ein wie die grundsätzliche Komplexität der möglichen Algorithmen, die Flexibilität, Entwicklungszeit und ein Einsatz als kryptographischen Hardwarebeschleuniger zur Steigerung der Informationssicherheit.

Prozessoren weisen unter den Vergleichssystemen die geringste Rechenleistung und einen durchschnittlichen Energieverbrauch auf, lassen sich aber aufgrund der komplexen Rechenkerne an eine Vielzahl von Anwendungen und Algorithmen durch Software anpassen. Sie sind daher sehr flexibel und die Entwicklungszeit der Programme damit gering. GPGPUs hingegen haben eine deutlich höhere Rechenleistung bei entsprechend hohem Energieverbrauch. Ihre Rechenkerne sind vom Funktionsumfang und der Programmabarbeitung in ihrer Komplexität eingeschränkt, wodurch die Flexibilität entsprechend geringer ist.

FPGAs hingegen können flexibel an unterschiedlichste Problemstellungen durch Konfiguration der Hardware angepasst werden und verfügen dabei über eine hohe Rechenleistung bei geringem Energieverbrauch. Gerade für den Einsatz in einem Rechenzentrum, welches Cloud-Dienste bereitstellt, eignen sich FPGAs zur Beschleunigung von unterschiedlichsten, sogar komplexen Anwendungen, außerordentlich gut (siehe Abschnitt 2.4.2 für typische Cloud- und sicherheitsrelevante Anwendungen). Eine nähere Betrachtung von FPGAs als Beschleuniger zur energieeffizienten Steigerung von Rechenleistung und ihren möglichen Einsatz in sicherheitskritischen Bereichen innerhalb einer Cloud-Architektur ist daher vielversprechend, wie Putnam et al. in [Put+14], Byma et al. in [Bym+14] und Chen et al. [Che+14] darstellen. Im folgenden Abschnitt 2.1.2 werden Entwicklung, Architektur, Kopplungs- und Konfigurationsarten von FPGAs näher erläutert.

2.1.2 Architektur von FPGAs, Kopplungs- und Konfigurationsarten

Field Programmable Gate Arrays sind Halbleiterbauelemente, auf denen beliebige Schaltungen aufgrund ihrer internen rekonfigurierbaren Struktur realisiert werden können. Zunächst zeigt Abschnitt 2.1.2.1 die historische Entwicklung von FPGAs, wonach in Abschnitt 2.1.2.2 die Hardwarearchitektur detailliert vorgestellt wird. Anschließend werden in Abschnitt 2.1.2.3 Möglichkeiten der Kopplung von FPGA und CPU untersucht. Danach wird in Abschnitt 2.1.4.3 zunächst die partielle dynamische Rekonfiguration und in Abschnitt 2.1.4.4 die Möglichkeiten zur Verschlüsselung eines Bitstreams aufgezeigt.

2.1.2.1 Historische Entwicklung

Der erste FPGA wurde im Jahr 1984 von Xilinx vorgestellt, wobei der Begriff erst von Actel ab 1988 etabliert worden ist [Tri15]. FPGAs sind dabei die komplexesten und leistungsfähigsten Vertreter aus der Reihe der Hardware programmierbaren Schaltkreise⁴. Zu diesem Zeitpunkt waren ASICs weit verbreitet um benutzerdefinierte logische Schaltungen im Silizium umsetzen, hatten aber den Nachteil einer langen Entwicklungs- und Produktionszeit. Durch die zunehmende Anzahl von Prozessschritten und der dafür erforderlichen Masken wurde es zunehmend teurer, einen ASIC zu entwickeln. FPGAs stellten eine kostengünstigere Alternative⁵ [Tri15] für niedrige bis mittlere Stückzahlen dar. Die Besonderheit bei FPGAs ist dabei die Möglichkeit vorgefertigte Schaltkreise zu nutzen und die Funktion während der Laufzeit durch *Konfiguration* einer neuen Schaltung auf die zugrundeliegende rekonfigurierbare Architektur anzupassen. Diese Möglichkeit macht FPGAs für das *rapid Prototyping*, dem Verfahren kostengünstige aber schnelle und effiziente Schaltungen direkt in Hardware zu realisieren, interessant. FPGAs stellen ebenfalls eine geeignete Plattform für das *Reconfigurable Computing*⁶ (siehe Definition 2.2) dar [Bob07; HD08; Tre96]:

Definition 2.2: (Re)configurable Computing *Additional processor used in some personal computers to perform specialized tasks such as extensive arithmetic calculations or processing of graphical displays. The coprocessor is often designed to do such tasks more efficiently than the main processor, resulting in far greater speeds for the computer as a whole⁷. [Bri16]*

Ab dem Jahr 2000 waren FPGAs Standardkomponenten in vielen digitalen Systemen. Aufgrund ihrer hohen Kapazität und der dadurch wachsenden Systeme wurden sie zuerst für Anwendungen aus dem Bereich der Netzwerk- und Telekommunikation interessant [Tri15]. Durch die zunehmende Integrationsdichte der FPGAs wurden diese in der Zahl ihrer Gatter zunehmend umfangreicher als ASICs, wodurch auch komplexe Systeme (Plattform FPGA [FPG05]) auf ihnen realisiert werden konnten. Xilinx führte ab dem Jahr 2000 die dynamische partielle Rekonfiguration zur Laufzeit ein [Kao05; McD08], wodurch Teilkomponenten eines FPGAs ideal an die aktuellen Bedürfnisse der Anwendung angepasst werden konnten, was sie schließlich für den HPC-Bereich als Hardwarebeschleuniger [EGE08] und ab 2013 ebenso als Beschleuniger für das Cloud-Computing [Bym+14; Che+14; Put+14] interessant machte.

⁴Der Bereich der (re)konfigurierbaren Logik reicht von einfachen, mindestens einmalig konfigurierbaren Simple Programmable Logic Devices (SPLDs) über Complex Programmable Logic Devices (CPLDs), welche SPLDs über ein konfigurierbares Verbindungsnetzwerk koppeln, bis zu den rekonfigurierbaren, also mehrmals programmierbaren, FPGAs [HM04].

⁵Der Einsatz von ASICs lohnt sich bei hohen Stückzahlen oder wenn Einschränkungen hinsichtlich der Leistungsaufnahme oder Geschwindigkeit bestehen.

⁶Die ersten Ansätze für das Reconfigurable Computing gehen zurück auf den Prozessor mit rekonfigurierbaren Gitter von Gerald Estrin im Jahr 1960 [Est60] und dem Xputer von Reiner Hartenstein [Har01].

⁷Freie Übersetzung von [Bri16]: Zusätzlicher Prozessor, der in Computern verwendet wird um spezialisierte Aufgaben, wie umfangreiche arithmetische Berechnungen oder die Verarbeitung von grafischen Darstellungen. Da der Coprozessor speziell für diese Aufgaben ausgelegt ist, arbeitet er deutlich effizienter als der Hauptprozessor, was in einer deutlich höheren Verarbeitungsgeschwindigkeit des gesamten Rechensystems resultiert.

2.1.2.2 Definitionen, Begriffe und die Architektur von FPGAs

Durch die Konfiguration der intern vorhandenen Komponenten können in einem FPGA verschiedene Schaltungen und Funktionen direkt in Hardware realisiert werden. Die Möglichkeiten reichen von einfachen logischen Verknüpfungen, über einfache Zustandsautomaten, bis hin zu vollständigen Mehrkern-Prozessoren und hochparallelen Systemen. Um dies zu ermöglichen bestehen FPGAs aus einem Array von konfigurierbaren Logikblöcken (Configurable Logic Blocks (CLBs)) und einer hierarchisch aufgebauten, ebenfalls konfigurierbaren Verbindungsstruktur. Die Möglichkeit der Rekonfiguration ergibt sich aus dem Schreiben von Bits in Static Random Access Memory (SRAM) basierte Look Up Tables (LUTs), FlipFlops und Multiplexer zum Schalten der Verbindungsleitungen.

Das anwenderspezifische Hardwaredesign wird dabei auf der zugrundeliegenden FPGA-Architektur ausgeführt. Sämtliche logischen Verknüpfungen und Verbindungsleitungen ergeben sich durch die Konfiguration und bilden das Hardwaredesign. Diese rekonfigurierbare Zwischenschicht mit Millionen von Multiplexern und Verbindungsleitungen, welche zusammengeschaltet werden können und wovon eine reale Anwendung oft nur wenige benötigt, stellen einen wesentlichen Nachteil von FPGAs dar. Im Vergleich zu ASICs, bei denen die Gatter einfach direkt miteinander verbunden sind, führt die Möglichkeit zur Rekonfiguration bei FPGAs zu einer Reduzierung der Geschwindigkeit und zusätzlichem Energiebedarf [KZH16]. Dieses sogenannte *Technology Gap* zwischen FPGAs und ASICs wurde von Kuon und Rose in [KR07] untersucht. Für einen 90 nm-Prozess ist beispielsweise ein FPGA um den Faktor 35 größer als ein ASIC mit dem selben Anwenderdesign. Um die Effizienz zu steigern befinden sich in modernen FPGAs dedizierte Multiplizierer, DSPs und Speicherblöcke, was zu einer Verringerung des Technology Gaps auf 18 beigetragen hat [KR07]. Die Geschwindigkeit von FPGAs ist mit einer Taktrate von zur Zeit bis zu 300 MHz typischerweise um bis zu vier Mal geringer als die von ASICs [KR07]. Die Möglichkeit, FPGAs durch ihre Rekonfigurierbarkeit durchschnittlich besser auslasten zu können, gleicht aber diese Einschränkungen aus [KZH16; VB13].

Die sich für eine Rekonfiguration ergebenden Grundbausteine eines modernen FPGAs sind in einem Gitter angeordnet, wie in Abbildung 2.3 dargestellt. Nach [KZH16] und [HM04] sind die Basiselemente, aus denen jeder FPGA aufgebaut ist, im Einzelnen Konfigurierbarer Logikblöcke (CLB), Verbindungsnetzwerk mit Connection Block (CB) und I/O-Blöcke. Eine detaillierte Beschreibung der Komponenten ist im Anhang unter Abschnitt A.1.1 zu finden.

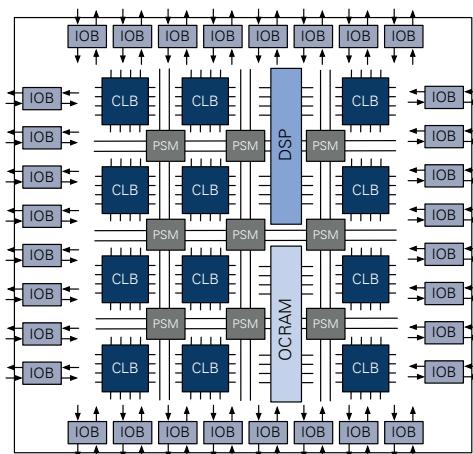


Abbildung 2.3: Schematischer Aufbau eines FPGAs mit Logik- (CLBs), I/O, DSP- und dedizierten Speicherblöcken, sowie den konfigurierbaren Leitungen mit den Verbindungsblöcken (PSMs), DSPs und OC-RAM. Nach [HD08, S. 9].

Um höheren Anforderungen an Integrationsgrad und Geschwindigkeit entgegenzukommen, gibt es eine Reihe weitere fester Komponenten wie DSPs, Multiplexer, Speicherblöcke (OCRAM), PCIe-Endpunkte oder vollständige Prozessoren, welche über das konfigurierbare Verbindungsnetzwerk in das System integrierbar sind [HM04; KZH16].

Die Komplexität und der Aufbau moderner FPGAs werden in Abbildung A.2 veranschaulichen. Der gesamte Chip ist dabei in mehrere lokale Taktregionen eingeteilt. Die Struktur der horizontal angeordneten Komponenten ist nur teilweise homogen und durch die eingebetteten Funktionsblöcke zur Rekonfiguration, weiterer Infrastruktur und PCIe-Endpunkten unterbrochen, was eine beliebige Anordnung und ein Verschieben von Teilkomponenten erschwert. Die Identifikation von homogenen Regionen zur Verschiebung von Komponenten ohne ein erneutes Platzieren und Verknüpfen ist Gegenstand aktueller Forschungen, wie in den Arbeiten von Backasch et al. [Bac+14] oder Ichinomiya et al. [Ich+12] und wird in Abschnitt 2.1.4.3 nochmals aufgegriffen.

2.1.2.3 Kopplung zwischen Prozessoren und FPGAs

Für den Einsatz von FPGAs als Rechenbeschleuniger oder Coprozessor ist eine Kopplung zwischen CPU und FPGA unerlässlich, um einerseits eine hohe Parallelität durch spezielle Hardware und andererseits einen universellen Programmfluss durch einen klassischen Programmablauf zu erreichen. Verschiedene Ansätze zur Kopplung von FPGA und CPU werden im Folgenden aufgezeigt, wobei als Ausgangspunkt eine klassische Systemarchitektur, wie sie in Abbildung 2.4 gezeigt ist, dienen soll. Die unterschiedlichen in der Literatur vertretenen Ansätze werden im Folgenden erläutert und sind ebenso in Abbildung 2.5 dargestellt. Unterschiedliche Möglichkeiten FPGAs mit Prozessoren zu koppeln werden in den Arbeiten von Compton et al. [CH02], aber auch Todman et al. [Tod+05] gezeigt und im Folgenden kurz erläutert.

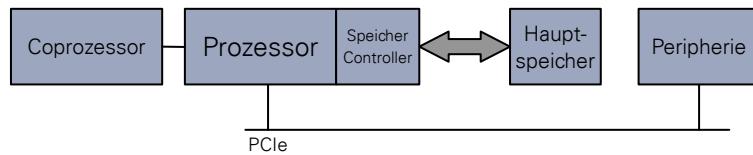


Abbildung 2.4: Aufbau eines einfachen Rechnersystems bestehend aus Prozessor, Speicher und Bus für Peripheriegeräte. Erweiterung von [Tan06, S. 122].

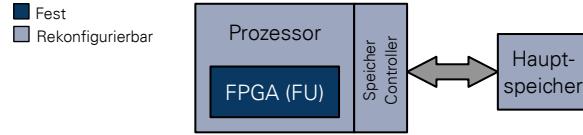
Integration von Prozessor und FPGA auf einem Chip: Eine Möglichkeit, einen FPGA mit einem Prozessor zu koppeln ist die direkte Integration des FPGAs in den Prozessorchip als rekonfigurierbare Funktionseinheit (Functional Unit (FU)), wie in Abbildung 2.5(a) gezeigt. Hierbei sind nach Compton et al. [CH02] die rekonfigurierbaren Komponenten direkt als Funktionseinheit in den ebenfalls rekonfigurierbaren oder erweiterbaren Befehlssatz des Prozessors eingebettet.

FPGA als klassischer Coprozessor: Die enge Kopplung eines FPGAs direkt mit dem Prozessor wie in Abbildung 2.5(b) gezeigt, entspricht einem klassischen Coprozessor, wie die Systeme in den Arbeiten von Razdan [Raz94] oder Niyonkuru et al. [NZ04]. Der FPGA hat hierbei nur eine untergeordnete unterstützende Rolle und wird über speziellen Befehle vom Prozessor administriert, beziehungsweise mit Daten versorgt.

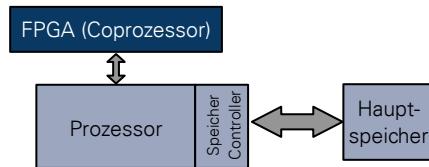
Kapselung eines Prozessors: Eine weitere Möglichkeit besteht im umgekehrten Weg der Integration eines statischen Prozessors als eigenständige Komponente innerhalb eines FPGAs. Dabei liegt der Prozessor im rekonfigurierbaren Gitter und der FPGA kann nicht nur Teile einer Anwendung beschleunigen, sondern auch über den Zugang zum Prozessor ein- oder ausgehende Daten be-

2 Stand der Forschung und Technik

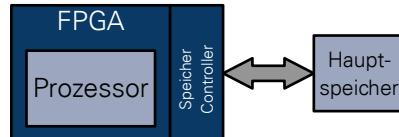
arbeiten. Der FPGA rückt bei dieser Architektur stärker in den Vordergrund als bei den vorherigen Konzepten. Beispiel für einen FPGA mit integrierten festverdrahtetem Prozessor ist die Zynq-Architektur von Xilinx [Cro+14] mit festen Cortex-A9 innerhalb des FPGA oder SoC-Architekturen [Int17b] von Intel (Altera).



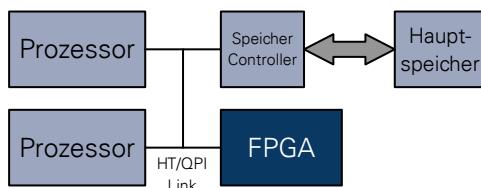
(a) Integration eines FPGAs als FU in einen Prozessoren mit Zugriff auf den gemeinsamen Hauptspeicher.



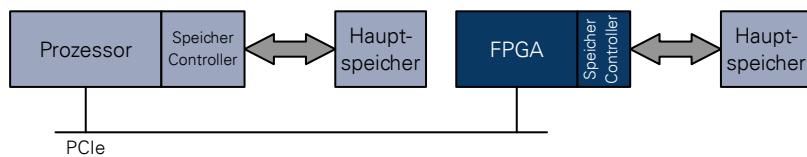
(b) Enge Kopplung zwischen FPGA als Coprozessor und der Prozessor mit Erweiterung des Befehlssatzes.



(c) Kapselung eines Prozessors als feste Komponente innerhalb des FPGAs mit Zugriff auf den Hauptspeicher über den FPGA.



(d) Integration eines FPGAs als (lose gekoppelter) Coprozessor direkt am Speicherbus des Prozessors mit gemeinsamem, einheitlichem Zugriff auf den Hauptspeicher.



(e) Kopplung von Prozessor und (prinzipiell) eigenständigem FPGA (Hardwarebeschleuniger) über den PCIe-Bus mit jeweils separatem Hauptspeicher.

Abbildung 2.5: Möglichkeiten der Kopplung von FPGAs mit Prozessoren.

Einsatz des FPGAs als externer lose gekoppelter Hardwarebeschleuniger: Ein Ansatz zur direkten

Nutzung von FPGAs als leistungsfähige Hardwarebeschleuniger ist in Abbildung 2.5(d) aufgezeigt. Systeme wie Convey HC-1 [Bak10] oder auch die Cray XD1 [Cra16] nutzen eine enge Kopplung über den Speicherbus oder den Front-Side Bus (FSB) [Bre10], um eine schnelle Kommunikation zwischen den Prozessoren und den FPGA-Beschleunigern zu ermöglichen. Die dazu

eingesetzten Protokolle sind abhängig von den Hauptprozessoren, dazu gehören beispielsweise Intels QuickPath Interconnect (QPI) [Zia+10], der von AMD eingesetzte HyperTransport-Link (HT) [Ahm+02] oder IBMs Coherent Accelerator Processor Interface (CAPI) [Stu+15].

FPGA mit eigenem Speichercontroller: Einen weiteren, aber deutlich flexibleren Ansatz der Kopplung eines Prozessors mit weiteren Geräten wie FPGAs stellt der PCIe Bus⁸ dar. Derartige Systeme, wie in Abbildung 2.5(e) gezeigt, stellen den zur Zeit am weitesten verbreiteten Ansatz zur Kopplung von Prozessoren und Hardwarebeschleunigern dar und werden auch von den in Abschnitt 2.1.1.1 vorgestellten Beschleunigern eingesetzt. Der derzeitig erreichbare Datendurchsatz liegt in der selben Größenordnung wie bei den zuvor genannten Systemen. Die Latenz ist aufgrund der lockerenen Kopplung allerdings entsprechend größer. Prozessor und FPGA verfügen hierbei über separate Hauptspeicher, wobei der Zugriff üblicherweise über DMA von Seiten des Prozessors erfolgt. In diese Kategorie fällt ebenfalls der Intel Stellarton (Intel Atom E6x5C) mit zusätzlichem FPGA auf demselben Board, welcher über PCIe mit dem Prozessor gekoppelt ist [Int17a]. Der Axel Cluster von Tsoi et al. [Tso10] basiert auf Hardwarebeschleunigern wie FPGAs und GPGPUs und nutzt PCIe, um eine größtmögliche Flexibilität und Unabhängigkeit von Herstellern zu erreichen. Weitere konkrete, und für diese Arbeit relevante Systeme mit einer Kopplung von Prozessor und FPGA werden in Abschnitt 2.4.4 vorgestellt.

2.1.2.4 Konfiguration

Ein wichtiger Punkt der bei der Kopplung zwischen FPGA und Prozessor beachtet werden muss ist die Konfiguration des FPGAs selbst. Diese erfolgt bei FPGA-Boards typischerweise über die Boundary-Scan oder JTAG-Schnittstelle und einen entsprechenden Controller um den Zugriff über USB oder andere Protokolle zu gewährleisten. Alternativ kann die Konfiguration ohne ein Rechnersystem mit Hilfe eines Serial Peripheral Interface (SPI) oder Byte Peripheral Interface (BPI) zu einem Flash-Speicher erfolgen, in welchem ein vordefinierter Bitstream für den FPGA gehalten wird [Xil16b; Xil16f]. Der Bitstream selbst wird direkt beim Einschalten des Systems auf den FPGA übertragen.

Neben weiteren in [Xil16b] dokumentierten Methoden besteht bei FPGAs des Herstellers Xilinx die Möglichkeiten der Konfiguration über den Internal Configuration Access Port (ICAP). Dieser ermöglicht eine Konfiguration eines partiellen Bereiches des FPGAs über einen statischen Bereich innerhalb des Designs, welcher einen Zugang zum ICAP bereitstellt. Über ein entsprechendes Design kann ein Zugang zum ICAP von der Außenwelt erfolgen und somit eine Konfiguration über ein Netzwerkinterface, QPI, PCIe oder weitere Protokolle ermöglicht werden. Eine notwendige Voraussetzung besteht darin, dass grundlegende Design in statischen und rekonfigurierbaren Bereiche auf dem FPGA zu unterteilen (siehe Abschnitt 2.1.4.3).

2.1.3 Einsatzgebiete und typische Anwendungen

Typische Einsatzgebiete von FPGAs reichen vom einfachen ASIC-Prototyping über eigenständige Netzwerkrouter oder Mobilfunkbasisstationen, bis zur Industrieautomatisierung und einem Einsatz in Fahrzeugen. Insbesondere werden sie auch als Hardwarebeschleuniger zur Unterstützung von Prozessoren in Rechenzentren genutzt, wie bereits in Abschnitt 2.1.2.3 erläutert. Die Anwendungsszenarien für die einzelnen Gebiete sind dabei sehr unterschiedlich und werden im Folgenden erläutert.

⁸PCIe erreicht in Version 3.0 mit 16 Lanes eine Datenrate von bis zu 15.754 MByte/s.

Anwendungsspezifische Schaltungen: FPGAs werden häufig für Anwendungen eingesetzt, bei denen die Latenz von großer Bedeutung ist, Entscheidungen schnellstmöglich getroffen werden müssen und dabei ebenfalls große Datenmengen im Streaming auszuwerten sind. Beispiele sind die direkte Auswertung von Messwerten, um das Datenvolumen zu reduzieren [Mus08; You+09], zeitkritische Entscheidungen im Finanzsektor mit möglichst geringer Latenz zum Netzwerk [LGL11; Mer76; Wes+11], aber auch die Auswertung von sicherheitskritischen Systemen im Auto [SGH15] oder Flugzeug. Im Gegensatz zu ASICs kann die Funktion der Hardware jederzeit adaptiert werden.

Hardwarebeschleuniger für Prozessoren: Für das Einsatzgebiet der Hardwarebeschleunigung eignen sich Anwendungen, welche eine hohe Parallelität erfordern und mit einfachen Datentypen arbeiten wie beispielsweise das Sequenzalignment [KPS11; Ols+12], neuronale Netze [JK07] oder die Analyse von Bilddaten im medizinischen Bereich [Cor+02]. Speziell für kryptografische Probleme sind aufgrund der einfachen, aber zahlreichen Operationen von Brute-Force Ansätzen FPGAs oftmals bedeutend leistungsfähiger als andere Systeme [JTS05]. In der Bild- und Videobearbeitung erreichen FPGAs ebenso eine hohe Rechenleistung und sind für einige konkrete Filteraufgaben deutlich leistungsfähiger als GPGPUs [Bha+09]. Selbst Datenbankanfragen können unter Umständen effizient auf FPGAs übertragen werden [WIA13]. Die Auslagerung von konkreten Anwendungen auf FPGAs im Bereich des Cloud-Computings wird in Abschnitt 2.4.2 näher betrachtet.

Autonom agierende Netzwerkkomponenten: Typische Einsatzorte von FPGAs in Rechenzentren sind weniger auf der Anwendungsebene, sondern vielmehr innerhalb der Infrastruktur zu finden. Beispielsweise werden FPGAs in Netzwerkroutern oder als Zugangshardware eingesetzt, um bei voller Datenrate direkt Netzwerkoperationen auf den Datenströmen auszuführen. Mögliche Operationen sind dabei Analysen der Paketheader oder des kompletten Payloads, wie in der Arbeit von Ramaswamy et al. [RWW09] aufgezeigt wurde. Eine weitere Möglichkeit ist eine lokale Lastverteilung der eingehenden Datenpakete direkt auf Netzwerkroutern oder Netzwerkkarten wie von Oeldemann et al. [OWH17] untersucht. Netzwerkrouter mit integrierter Firewall [AD11; JRV08] und der Möglichkeit einer Deep Packet Inspection (DPI) [Spe05] sind möglich. Ebenso kann die Virtualisierung von Netzwerken mittels FPGAs innerhalb der Router effizient realisiert werden [Unn+10], sowie die Klassifikation von einzelnen Netzwerkpaketen [SL05] oder vollständigen Dokumenten [Van+13] und Anwendungen zur Ver- und Entschlüsselung von Daten im Netzwerk [NWZ13; Rao+16; Rou+04]. Hierbei werden typischerweise spezielle FPGA-Schaltungen mit mehreren Netzwerkinterfaces eingesetzt.

2.1.4 Entwurfsablauf, dynamische partielle Rekonfiguration und Erweiterungen

Neben der Architektur der FPGAs und deren möglicher Kopplung mit Prozessoren ist bei rekonfigurierbarer Hardware das Erzeugen eines Bitstreams mit der Beschreibung des eigentlichen Hardwaredesigns oder dem Verhalten der Hardware ein wesentlicher Aspekt bei der Entwicklung, beziehungsweise Übertragung von Anwendungen auf FPGAs. Im Gegensatz zur Übersetzung eines Programms für einen Prozessor mittels eines Compilers in den herstellerabhängigen Maschinencode ist das Erzeugen eines Bitstreams für einen FPGA deutlich zeit- und rechenaufwändiger. Die Verhaltensbeschreibung wird zunächst durch eine Synthese in eine Netzliste überführt, welche dann auf die realen Logikblöcke abgebildet wird (Mapping). Darauf folgt zunächst die Platzierung der Komponenten (Place) welche anschließend miteinander verknüpft werden (Routing). In einem letzten Schritt wird der eigentliche Bitstream zur Konfiguration des FPGAs erzeugt. Ein Ablauf dieser notwendigen Schritte nach [Gro11, S. 61] ist in Abbildung 2.6 aufgezeigt.

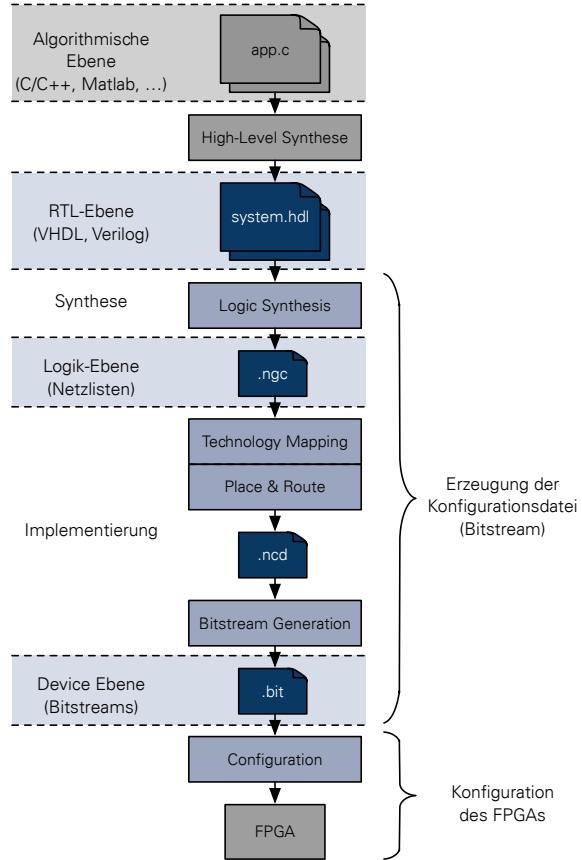


Abbildung 2.6: Entwurfsablauf mit der Erzeugung eines Bitstream aus der Hardwarebeschreibung und anschließender Konfiguration des FPGAs, sowie optionaler Erzeugung des RTL durch eine High-Level Synthesis. Nach [Gro11, S. 61] und [Gaj+12].

Register Transfer Level: Die RTL-Ebene ist die gängige Ebene für klassische Hardware Description Languages (HDLs)⁹ für den Entwurf digitaler Schaltungen. Auf der RTL-Ebene werden somit sämtliche Speicherelemente, Zustandsmaschinen und die logischen Verknüpfungen sowie die Verbindungen zwischen allen Elementen modelliert [KZH16, S. 19].

Logik-Ebene – Synthese: Aus dem RTL wird durch Logiksynthese eine Netzliste mit den realen physischen Elementen der Hardware wie FlipFlops, Gattern und Multiplexern erzeugt. Diese enthält sämtliche Verbindungen, sodass daraus das Verbindungsnetz mit allen Elementen erzeugt werden kann. Die Optimierung und Anpassung an die am besten geeigneten Elemente erfolgt ebenfalls in dieser Stufe [KZH16, S. 19].

Device-Ebene – Mapping, Placement und Routing: Die Übertragung der Netzliste auf einen realen FPGA-Architektur mit den konkreten Primitiven für LUTs, Speicher-, DSP- und I/O-Blöcken erfolgt im Schritt des Mappings, wobei beispielsweise für LUTs unterschiedliche Optimierungsziele, wie Anzahl der Ressourcen oder auch eine geringe Schaltungstiefe und damit geringe Verzögerungszeit wie bei dem Mapping von Cong et al. in [CD94] gelten können. Nach dem Mapping erfolgen mit der Platzierung der Configurable Logic Blocks (CLBs) auf dem FPGA und dem gleichzeitigen Festlegen der Verbindungsleitungen, die zeitaufwändigsten Rechenschritte im Entwurfsablauf. Sobald sämtliche Elemente platziert wurden und das Timing eingehalten wurde, erfolgt das Gene-

⁹Vertreter von HDLs sind beispielsweise wie die Very High Speed Integrated Circuit Hardware Description Language (VHDL) oder Verilog.

2 Stand der Forschung und Technik

rieren des Bitstreams mit sämtlichen internen Konfigurationen für Primitiven und Multiplexern innerhalb des Verbindungsnetzwerkes, damit sich der FPGA wie die gewünschte Hardware verhält.

Da die Entwurfsabläufe bis zum vollständigen Hardwaredesign sehr rechen- und datenintensiv sein können, sowie mit unterschiedlichen Optimierungszielen durchgeführt werden können, werden bereits Dienste für den vollständigen Hardwareentwurf in der Cloud angeboten [Plu16]. Zur Optimierung der Entwurfsabläufe können dabei Konzepte des Machine Learnings wie von Kapre et al. [Kap+15] eingesetzt werden, um optimale Parameter für unterschiedliche Entwurfswerkzeuge zu bestimmen und den Ablauf zu optimieren.

2.1.4.1 Hardware/Software-Codesign

Ein nicht zu vernachlässigender Aspekt bei der Übertragung von Anwendungen auf einen eng mit einem Prozessor gekoppelt FPGA-basierten Hardwarebeschleuniger, ist das Gebiet des *Hardware/Software Codesign* (HW/SW-Codesign). Ausgehend von der Systemspezifikation, beziehungsweise einer Anwendung, erfolgt eine Partitionierung der Anwendung in Software (Host-Prozessor) und Hardware (FPGA). Die Gründe für eine Implementierung von Teilen des Systems in Software sind hauptsächlich die geringere Entwurfszeit, somit geringere Entwicklungskosten und die höhere Flexibilität. Für die Implementierung in Hardware sprechen in der Regel eine höhere Verarbeitungsgeschwindigkeit und ein geringerer Energieverbrauch. Das Ziel des gesamten Prozesses besteht darin, die Gesamtkosten durch den Einsatz von Software zu minimieren und sämtliche Randbedingungen, wie Verarbeitungszeit oder Durchsatz, durch die Nutzung von Hardware einzuhalten [Sch14; Tei13].

Ein typischer Entwurfsablauf des HW/SW-Codesigns umfasst in einem ersten Schritt eine iterative Systemspezifikation mit der Validierung und der Partitionierung des Systems in Software und Hardware. Ausgehend von der Partitionierung erfolgt der Entwurf der beiden Systemteile, eine iterative Validierung und schließlich die High-Level Synthesis, um für den Hardwareentwurf die auszulagernden Funktionen von einer Hochsprache in eine Hardwarebeschreibungssprache zu überführen. Parallel dazu erfolgt eine Optimierung der Softwareseite, gefolgt von einer Validierung des Gesamtsystems oder gegebenenfalls einer erneuten Partitionierung. Die Herausforderung des HW/SW-Systementwurfes ist das sogenannte Design-Gap, welches darin besteht, dass die Komplexität der Hardware stärker zunimmt als die mit Entwurfswerkzeugen beherrschbare Komplexität [Sch14].

2.1.4.2 High-Level Synthesis

Die bereits zuvor erwähnte High-Level Synthesis ist ein automatisierter Entwurfsprozess, welcher eine algorithmische Verhaltensbeschreibung in einen Beschreibung eines digitalen Systems auf Ebene des RTLs überführt [CM08], sodass anschließend der zuvor beschriebene Ablauf bis zum Bitstream abgearbeitet werden kann. Mit Hilfe der HLS kann Programmierern und Softwareentwicklern die Übertragung von Algorithmen auf rekonfigurierbarer Hardware erleichtert werden und entsprechend auch die Produktivität gesteigert werden [Jon+10; Mer+08]. HLS-Werkzeuge erzeugen dabei aus dem Programmcode einen Datenfluss, das Scheduling der Operationen auf die Takschritte, die Allokation der notwendigen Hardwareressourcen in Form von ALUs, Registern, Bussen, sowie Multiplexern und schließlich die Verknüpfung von Operationen und Ressourcen [CW16].

Um im Programm die gewünschten nebenläufigen Bereiche sowie Speicherelemente zu kennzeichnen und somit die Verarbeitungsleistung des Hardwaredesigns zu erhöhen, existieren für die HLS-

Werkzeuge eine Reihe von architektspezifischen Direktiven, welche allerdings eine genaue Vorstellung des zu erzeugenden Hardwaredesigns erfordern. Ein erster lauffähiger Entwurf ist mit Hilfe der HLS in kurzer Zeit möglich, es entsteht aber durch notwendige Optimierungsschritte hinsichtlich Ressourcenverbrauch und Leistungsfähigkeit ein ähnlich hoher Entwurfsaufwand wie bei einer direkten Umsetzung auf RTL-Ebene, wie die Arbeit von Skalicky et al. [Ska+13] anhand einer Matrixmultiplikation zeigt. Beispiele für HLS-Werkzeuge sind VivadoHLS von Xilinx [Xil17e], BlueSpec [Blu17] oder LegUp der Universität Toronto [Can+13]. Die mittels HLS erzeugten Modulen müssen noch in eine Infrastruktur mit Schnittstellen zur Kommunikation eingebettet werden, wie sie weiterführend in Abschnitt A.1.3 aufgeführt sind.

2.1.4.3 Partielle Rekonfiguration sowie darauf aufbauende Konzepte

Die Rekonfiguration von FPGAs kann zunächst in *statische* und *dynamische* Rekonfiguration eingeteilt werden. Die dynamische Rekonfiguration kann zur Laufzeit erfolgen und sich auf Teilbereiche des FPGAs beziehen (partiell). Die dynamische partielle Rekonfiguration zur Laufzeit wurde von Xilinx ab dem Jahr 2000 eingeführt [Kao05; McD08] und ermöglicht es Teilkomponenten eines FPGAs dynamisch an die aktuellen Bedürfnisse der Anwendung anzupassen. Weiterhin wurden dadurch schnell und adaptiv an das Problem anpassbare FPGA-Entwürfe ermöglicht, bei denen beliebige Teile unabhängig voneinander ausgetauscht werden können. Dies erfolgt bei FPGAs von Xilinx mittels einer innerhalb der Infrastruktur zur Rekonfiguration eingebetteten Zustandsmaschine, welche einzelne Spalten (Frames) der FPGA-Architektur unabhängig voneinander neu konfigurieren kann [TM14; Xil17g]. Abbildung 2.7 zeigt exemplarisch einen FPGA mit acht Taktregionen und einem Rekonfigurations-Frame der eine Breite von einer Spalte und eine Höhe von einer kompletten Taktregion hat (siehe Abbildung 7.2(b)). Die Übergänge zwischen statischer und partiell rekonfigurierbarer Region werden als *Partition Pins (PPs)* bezeichnet und automatisch im Entwurfsablauf festgelegt. Der gesamte Bereich außerhalb der partiellen Region bildet den statischen Bereich. Da statische Leitungen unter Umständen durch eine partielle Region verlaufen können, muss das statische Design zur Erzeugung des partiellen Bitstreams vollständig bekannt sein. Durch Maskierung werden die statischen Leitungen im partiellen Bitstream ausgeblendet und bleiben während der Rekonfiguration unaufgetastet.

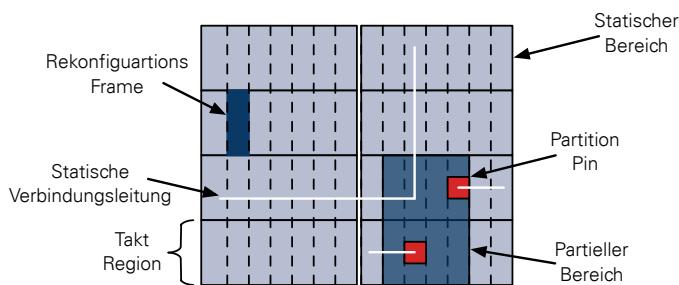


Abbildung 2.7: Prinzip der dynamischen partiellen Rekonfiguration. Bei Xilinx-FPGAs wird ein Rekonfigurations-Frame immer durch eine komplette Spalte einer Taktregion gebildet. Der partielle Bereich ist eine logische Region auf dem FPGA, welche an vollständige Spalten gebunden ist. Leitungen des statischen Teiles können durch den partiellen Bereich führen. Frei nach [Xil16b].

Ein Xilinx-Bitstream beginnt mit einem Wort zur Synchronisierung, gefolgt von einer Sequenz von Befehlen und letztendlich den Konfigurationspaketen. Diese Pakete setzen sich aus einer Reihe von Frames zusammen und werden durch einen Befehl eingeleitet, welcher in das interne Frame Adress Register (FAR)-Register die Startadresse des ersten Frames schreibt, gefolgt von der Anzahl der linear aufein-

ander folgenden Frames in dem Paket. Abbildung 2.8 zeigt exemplarisch einen Bitstream eines Virtex-7 FPGAs. Ein Frame ist dabei eine komplette Spalte innerhalb einer Taktregion (siehe Abbildung 2.7 oder Abbildung A.2). In einem Bitstream können beliebig viele Konfigurationspakete für unterschiedliche Bereiche enthalten sein, wodurch die partielle Rekonfiguration ermöglicht wird. Zum Abschluss wird die Integrität des Bitstreams über einen Cyclic Redundancy Check (CRC) überprüft und mit dem Befehl zur Desynchronisation wird der partielle Bereich des FPGAs zur Nutzung freigegeben [Xil16b].

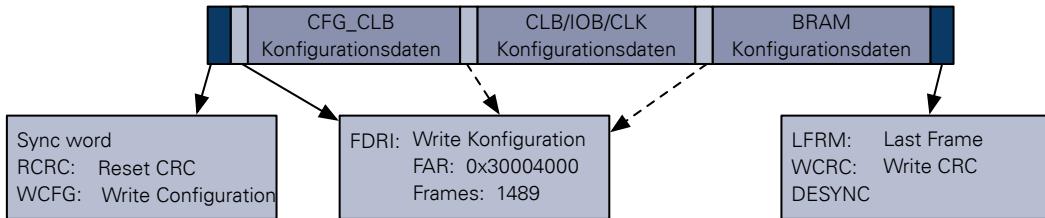


Abbildung 2.8: Aufbau eines (partiellen) Bitstreams für einen Xilinx FPGA. Die Blöcke mit den einzelnen Konfiguration Frames sind aufgeteilt in drei Kategorien (CFG_CLB, CLB/IO/CLK und BRAM). Frei nach [Xil16b].

Eine weitere Besonderheit bei modernen Xilinx FPGAs ist es dabei, dass die partielle Rekonfiguration zur Laufzeit direkt vom statischen Bereich des FPGA über einen direkten Zugang zur Infrastruktur der Rekonfiguration (ICAP [Xil16b]) durchgeführt werden kann. Die Nutzung des ICAP ermöglicht eine Rekonfiguration eines dynamischen Bereiches direkt über PCIe, das Netzwerk oder den angeschlossenen DDR-Speicher, was eine deutlich höhere Geschwindigkeit als eine Konfiguration über die JTAG-Schnittstelle ermöglicht. Tabelle 2.3 zeigt eine Auswahl von auf den ICAP aufbauende Controller, welche zum Teil auch das Auslesen (Readback) der aktuellen Konfiguration mit sämtlichen internen Registerinhalten ermöglichen.

Tabelle 2.3: Überblick zur Partiellen Rekonfiguration. Nach [Gen15].

System	Taktfrequenz	Busbreite	Quelle	Steuerung	Readback
Xilinx (AXI) [Xil10b]	100 MHz	32 Bit	AXI Lite	MicroBlaze	✓
Blodgett et al. [BML03]	50 MHz	8 Bit	OPB Bus	MicroBlaze	✓
Claus et al. [Cla+08]	100 MHz	32 Bit	DDR2	FSM	✓
Liu et al. [Liu+09]	100 MHz	32 Bit	On-Chip RAM (OCRAM)	HWICAP	✗
Vatsolakis et al. [VPP14]	125 MHz	32 Bit	PCIe	Host CPU & DMA	✗
Vipin et al. [V+12]	125 MHz	32 Bit	DDR3-RAM	Host CPU & DMA	✗
Genßler [Gen15]	100 MHz	32 Bit	PCIe	FSM	✓

✓: vorhanden ✗: nicht vorhanden

Aufbauend auf der partiellen Rekonfiguration ist das Framework GoAhead von Beckhoff et al. [BKT12] entstanden, welches einen einfachen Entwurf von Systemen ermöglicht, bei denen flexibel platzierte Komponenten ausgetauscht oder verschoben werden können. Dazu wird unter anderem ein Zwischenformat genutzt, welches das vollständig platzierte und geroutete Hardwaredesign in lesbbarer Form darstellt¹⁰. Somit ist es möglich, Teilkomponenten beliebig auszutauschen um das System adaptiv an die aktuellen Bedürfnisse anpassen zu können [KZH16, S. 9].

¹⁰Bei Xilinx-FPGAs stellt das Xilinx Design Language (XDL)-Format eine Möglichkeit dar, den platzierten und gerouteten Entwurf für einen FPGA vor Generierung der Konfigurationsdatei zu modifizieren [BKT11].

Darauf aufbauend gibt es eine Reihe von Arbeiten zum Gebiet der *Module Relocation*, wie beispielsweise die Arbeit von Beckhoff et al. [BKT14]. Um ein vollständiges Hardwaredesigns verschieben zu können, ist allerdings in der Regel die Nutzung homogener Bereiche auf dem FPGA erforderlich. Wie mehrere Arbeiten gezeigt haben ist dies möglich [Bac+14; Oom+15], allerdings müssen die zu verschiebenden Regionen klein sein und von statischen Leitungen und Komponenten freigehalten werden, wie in der Arbeit von Rettkowski et al. [RFG16]. Ein weiterer Aspekt ist des Weiteren die Nutzung von Overlays wie in der Arbeit von Kapre et al. [Kap+06] am Beispiel eines flexiblen Paketfilters gezeigt, wobei der Ansatz die Rekonfiguration von Verbindungsleitungen darstellt, um einzelne Module eines Hardwaredesigns in ihrem Verhalten anzupassen. Weitere aktuellere Arbeiten in diesem Feld stammen von Koch et al. [KBL13] oder Yue et al. [YKL15].

Um die Zuverlässigkeit von digitalen Schaltkreisen zu erhöhen beziehungsweise eine frühzeitige Erkennung von Fehlern zu ermöglichen, wie es für Prozessoren in Software möglich ist [Sch11], beschäftigen sich Arbeiten wie beispielsweise die von Jones [Jon07] mit Single-Event Upset (SEU)s in FPGAs. Dabei wird mit dem ICAP der eigene Bitstream ausgelesen, der aktuelle CRC berechnet und mit dem Original abgeglichen, um Fehler zu erkennen und zu beheben.

2.1.4.4 Verschlüsselung der Konfiguration

Da moderne FPGAs von Xilinx und Intel auf SRAM-Zellen basieren¹¹, bei denen ohne Versorgungsspannung die interne Konfiguration verloren geht, ist das Speichern des Bitstreams auf einem zusätzlichen Flash-Speicher oder einem Hostsystem zur initialen Programmierung erforderlich [TM14]. Um den Bitstream vor fremden Zugriff zu schützen, liegt dieser in der Regel verschlüsselt vor [Int16d; Xil16b]. Der Entwurfsablauf ist identisch zum unverschlüsselten, nur im letzten Schritt wird mit einem symmetrischen Verschlüsselungsverfahren mit einem geheimen Schlüssel der Bitstream gesichert¹². Über JTAG wird der Schlüssel in einen dedizierten persistenten Speicher¹³ innerhalb des FPGAs übertragen¹⁴. Der FPGA wird im Anschluss mit dem verschlüsselten Bitstream über die JTAG-Schnittstelle oder den ICAP konfiguriert [TM14].

Als weitere Sicherheitsstufe ist es möglich, eine Konfiguration nur nach erfolgreicher Authentifizierung des Nutzers zuzulassen. Das hierzu verwendete Verfahren basiert auf einem One-way Hashcode-Algorithmus¹⁵, mit dem die Identität des Senders und die Validität des Bitstreams überprüft wird [TM14]. Durch partielle Rekonfiguration über den ICAP ist das Auslesen eines zuvor verschlüsselten Bitstreams partiell und unverschlüsselt möglich [Xil16b, S.94], wodurch sich neue Angriffsszenarien bilden, welche beispielsweise in der Arbeit von Trimberger et al. [TM14] ausführlich beschrieben werden.

¹¹FPGAs von Microsemi oder Lattice nutzen internen Flash-Speicher für die Programmierung [TM14].

¹²Moderne Xilinx FPGAs ab dem Virtex-4, aber auch Microsemi nutzen zur Verschlüsselung den 256 Bit Advanced Encryption Standard (AES) [TM14].

¹³Der Schlüssel um den Bitstream auf dem FPGA zu lesen wird in einem Battery Backup Random Access Memory (BBRAM) auf dem FPGA-Board oder in einem eFUSE Register innerhalb der FPGA-Infrastruktur gehalten.

¹⁴Die Übertragung des Schlüssels erfolgt bei den meisten Herstellern im Klartext und muss daher in einer sicheren Umgebung erfolgen [TM14].

¹⁵Xilinx nutzt seit den Virtex-6 FPGAs den Secure Hash Algorithm (SHA)-256, um einen 256-Bit Hashed Message Authentication Code (HMAC) zu erzeugen. Die Algorithmen sind innerhalb des FPGAs direkt in dedizierter Hardware implementiert [Xil16b].

2.2 Virtualisierung

Die Integration spezieller Hardwarekomponenten in Mehrbenutzersysteme und insbesondere in Cloud-Architekturen erfordert die Virtualisierung der Hardware. Der folgende Abschnitt 2.2.1 erläutert zunächst die historische Entwicklung der Virtualisierung, sowie wichtige Definitionen und Begriffe. Abschnitt 2.2.2 stellt die Prozess- und Abschnitt 2.2.3 die System-Virtualisierung vor. In Abschnitt 2.2.4 werden Virtualisierungsansätze für spezielle Geräte diskutiert und ein Vergleich der Virtualisierungsarten in Hinblick auf eine spätere FPGA-Virtualisierung erfolgt abschließend in Abschnitt 2.2.5.

2.2.1 Historische Entwicklung, Definition und Begriffe

Die Grundlagen zur *Virtualisierung* (siehe Definition 2.3) gehen zurück bis in die 1960er Jahre, in denen IBM mit der Virtual Machine Facility/370 (VM/370) einen der ersten Ansätze prägte. Auf der Plattform wurde ein Mehrbenutzerbetrieb realisiert, bei dem Instanzen in *Virtuellen Maschinen* (VMs) ausgeführt wurden. Jede VM stellte dabei eine vollständige Nachbildung der zugrundeliegenden Hardware dar [Cre81]. Die Arbeit von Goldberg [Gol74] etablierte schließlich eine Vielzahl der heute gebräuchlichen Begriffe wie beispielsweise *Virtual Machine Monitor* (VMM). Popek und Goldberg [PG74] lieferten die formalen Grundlagen und den mathematischen Hintergrund mit Abbildungen als Beschreibung der Virtualisierung sowie die Anforderungen an die grundlegende Architektur, um VMs mittels eines VMMs oder *Hypervisors*¹⁶ zu unterstützen. Der Hypervisor verwaltet dabei die Ressourcenverteilung (CPU, Speicher etc.) vom Gast- auf das Host-System und verteilt die Ressourcen so, dass für jedes einzelne Gastsystem die Ressourcen bei Bedarf verfügbar sind. Virtualisierung ist demnach nach [SN05b, S. 3] die Abbildung eines nachgebildeten virtuellen Systems über eine Schnittstelle auf ein reales physisches System und ist wie folgt definiert:

Definition 2.3: Virtualization (Computer) *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 it¹⁷. [SN05b, S. 3]*

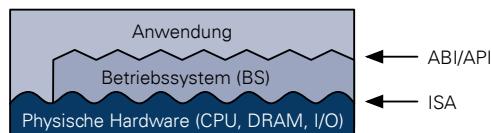
Ein modernes Rechensystemen ist aufgrund seiner Komplexität in Hierarchien auf unterschiedlichen Ebenen unterteilt, wobei diese über Schnittstellen miteinander verbunden sind, um eine Abstraktion zwischen den Ebenen zu ermöglichen. Abbildung 2.9(a) zeigt ein klassisches System mit einer Schnittstelle zur Hardware (Instruction Set Architecture (ISA)) und einer weiteren für Systemaufrufe (Application Binary Interface (ABI)) und für Bibliotheken (Application Programming Interface (API)), welche von High-Level Languages (HLLs) aufgerufen werden können. Nach der zuvor erfolgten Definition kann die Virtualisierung an diesen beiden Schnittstellen erfolgen [SN05b, S. 3]. Eine übliche Art der Virtualisierung stellen hierbei nach [SN05b] die Prozess- und die System-Virtualisierung dar, wie sie in Abbildung 2.9 aufgezeigt sind.

¹⁶Der Begriff *Hypervisor* wird als Synonym für einen VMM genutzt und setzt sich zusammen aus dem griechischen Wortbestandteil „Hyper“ (= über), sowie dem lateinischen Wortteil „videre“, was *sehen* bedeutet. Ein Hypervisor ist somit frei übersetzt ein System, welches andere Systeme *überblickt*.

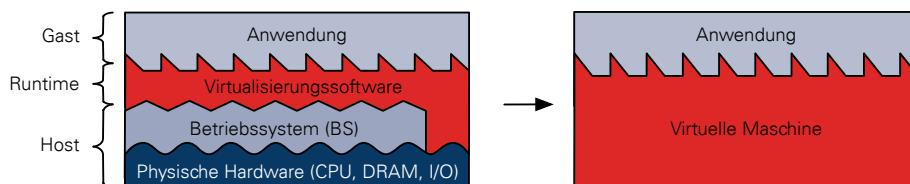
¹⁷Freie Übersetzung von [SN05b, S. 3]: Virtualisierung bietet eine Möglichkeit, Einschränkungen aufzuheben und die Flexibilität zu erhöhen. Wenn ein System (...) virtualisiert wird, werden die Schnittstellen und alle Ressourcen, die durch die Schnittstelle sichtbar sind, auf die Schnittstelle und Ressourcen eines realen Systems abgebildet, welches sie tatsächlich implementiert.

2.2.1.1 Unterscheidung zwischen Prozess- und System-Virtualisierung

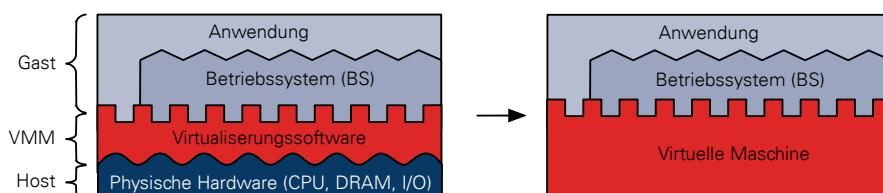
Eine *Virtuelle Maschine (VM)* kann sich sowohl auf der Ebene von Prozessen als auch auf der Ebene des Betriebssystems selbst befinden. Dabei ist die Perspektive dieser Systeme auf die ihnen zugrundeliegenden Ebenen entscheidend für die Einordnung. Aus Sicht eines Prozesses gibt es zum Beispiel einen logischen Speicher, die Ausführung eines Programms erfolgt über das Betriebssystem und somit das ABI [SN05b], wie in Abbildung 2.9(a) gezeigt. Durch eine Virtualisierungsschicht zwischen Anwendung und Betriebssystem kann eine Abstraktionsschicht, wie in Abbildung 2.9(b) gezeigt, eingeführt werden, wodurch die Abstraktion vom eigentlichen Betriebssystem erfolgt. Ein Beispiel für eine *Anwendungs-Virtualisierung* ist die Java Virtual Machine (JVM), welche als Zwischenschicht zwischen der Java-Anwendung und dem Betriebssystem agiert und die Plattformunabhängigkeit auf API-Ebene ermöglicht. Die Zwischenschicht wird als Laufzeitumgebung oder *Runtime* bezeichnet und nutzt einen Zwischenencode (Bytecode), was zur Folge hat, dass es sich bei der Virtualisierung um eine *Emulation* handelt. Allgemein gesprochen kann Virtualisierung somit die Simulation von Software oder Hardware sein, auf der wiederum andere Software ausgeführt wird [Sca11; SN05b].



(a) System ohne Virtualisierung mit Kennzeichnung der Schnittstellen. ABI und API bilden dabei eine Schnittstelle für Bibliotheken.



(b) Prozess-Virtualisierung über Runtime zwischen Betriebssystem/Hardware und Anwendung. Nach [SN05b, S. 11].



(c) System-Virtualisierung zwischen physischer Hardware und Betriebssystem. Nach [SN05b, S. 12].

Abbildung 2.9: Unterscheidung von Prozess- und System-Virtualisierung.

Entsprechend ist eine *System-Virtualisierung* aus Sicht des Betriebssystems eine Abstraktion von allen Ressourcen eines Rechnersystems unterhalb der ISA-Ebene. Durch eine Virtualisierungsschicht, wie in Abbildung 2.9(c) gezeigt, kann über einen Hypervisor das Betriebssystem von der realen Maschine gelöst werden. Dadurch wird es möglich, ein System durch mehrere unabhängige Betriebssystem-Instanzen zu nutzen [Sca11]. Jede VM stellt einen Container für ein Betriebssystem bereit und verhält sich wie ein vollwertiges System, welches aber vom physischen System abstrahiert ist. Dieser Ansatz er-

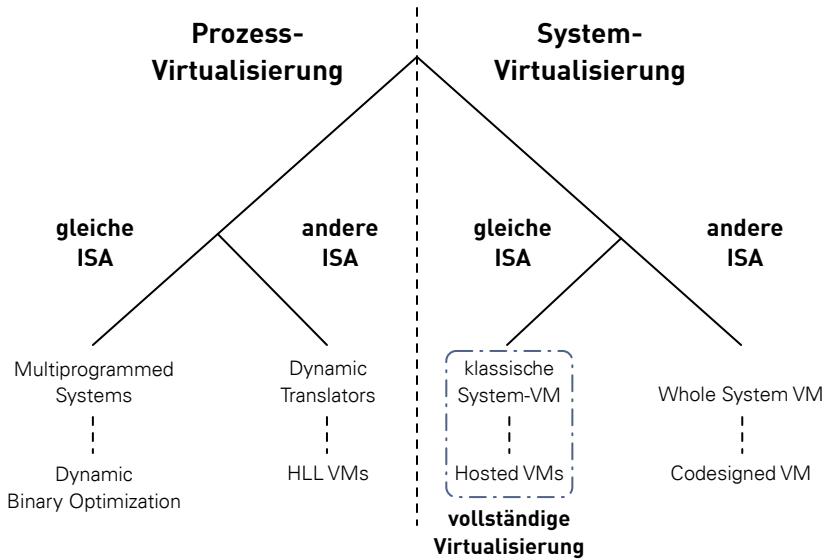


Abbildung 2.10: Taxonomie der Virtuellen Maschinen mit Kennzeichnung der vollständigen Virtualisierung. Taxonomie nach [SN05b, S. 23].

möglich demnach mehrere voneinander abgeschottete Umgebungen auf einer gemeinsam genutzten physischen Maschine.

2.2.1.2 Konzepte und Taxonomie von Virtuellen Maschinen

Die Unterteilung in Prozess- und System-Virtualisierung bildet die erste Stufe in der Taxonomie von VMs, welche in Abbildung 2.10 dargestellt ist. Innerhalb dieser Einteilung sind weitere Unterscheidungen der VMs anhand der ISA möglich. Die Emulation einer ISA ist dabei eine entscheidende Besonderheit für eine flexible Virtualisierung. In den folgenden Abschnitten werden unterschiedliche Virtualisierungskonzepte innerhalb dieser beiden Varianten vorgestellt.

2.2.2 Prozess-Virtualisierung

Prozess-VMs stellen ein virtuelles Interface (API) für Anwendungen bereit, um diese in einer abgekapselten Umgebung auszuführen. Nach der Taxonomie in Abbildung 2.10 sind zum Beispiel sämtliche modernen Betriebssysteme Prozess-VMs, wenn sie Multiprogrammbetrieb unterstützen. Prozess-VMs mit unterschiedlichen ISAs nutzen die dynamische Übersetzung eines Programmcodes, um diesen auf einem System mit einer anderen ISA zu emulieren. Das Ziel eines solchen Vorgehens ist in der Regel die Loslösung einer Anwendung oder des ausführbaren Programms (Binärdatei) von dem zugrundeliegenden Betriebssystem oder der physischen Architektur. Es wird hierbei unterschieden zwischen den nachfolgend erläuterten Varianten.

2.2.2.1 Betriebssysteme mit Multiprogrammbetrieb – gleiche ISA

Betriebssysteme selbst sind die älteste Form der Prozess-Virtualisierung zur Bereitstellung eines Multiprogrammbetriebes, da sie jedem Prozess eine eigene virtuelle Umgebung bereitstellen. Die gemeinsam genutzte Hardware (CPU, Speicher etc.) wird den Prozessen durch das Betriebssystem in Zeit-

scheiben zugeteilt. Damit wird den Prozessen der Eindruck einer jeweils exklusiven Nutzung suggeriert. Das Betriebssystem dupliziert virtuell die verfügbare Hardware für sämtliche Prozesse und agiert als Prozess-VM [SN05b, S. 13].

2.2.2.2 Emulatoren und dynamische binäre Übersetzung – unterschiedliche ISAs

Ein anspruchsvoller Problem als der Multiprogrammbetrieb stellt die Ausführung einer Anwendung auf einem System mit einer anderen ISA dar. Eine solche Nachbildung eines Systems entspricht einer *Emulation*, was allerdings nicht gleichbedeutend mit einer Virtualisierung ist. Typischerweise wird Emulation durch Interpretation durchgeführt, wobei der Interpreter einen Befehl lädt, dekodiert und entsprechend den Gast-Befehl mit einem oder mehreren Host-Befehlen ausführt, was unter Umständen auch deutlich zeitaufwändiger sein kann. Eine höhere Leistung wird mit der *dynamic binary translation* erreicht, bei der komplette Blöcke von Befehlen übersetzt, im Cache gehalten und wiederverwendet werden. Durch das Caching wird der Mehraufwand durch die Übersetzung reduziert [SN05b, S. 13].

2.2.2.3 Dynamische binäre Optimierung – gleiche ISA

Ein weiteres Anwendungsfeld der dynamischen Übersetzung ist – neben der Übersetzung von Anwendungen aus einer in eine andere Maschinensprache – die Programmoptimierung innerhalb derselben ISA. Dabei werden VMs genutzt, bei denen Quell- und Ziel-ISA von Host und Gast identisch sind und eine Optimierungsschicht die eigentliche Virtualisierung darstellt. Von den unoptimierten ausführbaren Dateien werden zunächst Profile erzeugt, welche zur Ausführungszeit dynamisch zur Optimierung von Codeabschnitten genutzt werden oder direkt Funktionen für das Caching bereit halten [SN05b, S. 15]. Ein Beispiel für einen solchen *same-ISA dynamic binary optimizer* ist das Dynamo System, ein Forschungsprojekt von Bala et al. [BDB00].

2.2.2.4 High-Level Language VMs – unterschiedliche ISAs

Die Portabilität von Programmen auf unterschiedlichste Architekturen mit verschiedenen ISAs mittels einer *High-Level Language (HLL)*-VM stellt eine weitere Form der Prozess-Virtualisierung dar. Dabei wird ein abstrakter Zwischencode (zum Beispiel Bytecode) in einer virtuellen ISA erzeugt, welche die Schnittstelle zu einer Runtime darstellt. In der Runtime wird der Zwischencode entsprechend auf die physische API übersetzt und schließlich im Betriebssystem ausgeführt. Somit stellt die Runtime die Virtualisierungsschicht zwischen Programmcode und dem Betriebssystem dar. Bekannte Beispiele dafür sind die JVM [MD97] oder die Common Language Infrastructure als Basis des .NET frameworks [Wig+02]. Ein Vorteil dessen ist die Plattformunabhängigkeit, wobei allerdings die Ausführungsgeschwindigkeit geringer ist als bei nativem Programmcode [SN05b, S. 15].

2.2.3 System-Virtualisierung

Die System-Virtualisierung ist die bekannteste Form der Virtualisierung, bei der die Ressourcen eines Rechnersystems von einer oder mehreren Betriebssystem-Instanzen genutzt werden, um damit die Auslastung der Ressourcen zu steigern. Jede VM verhält sich dabei wie ein vollständiges Rechnersystem mit eigenen Komponenten und voneinander abgeschotteten Umgebungen. Zugriffe aus der VM auf externe Geräte werden von der Virtualisierungsschicht zwischen VM und physischer Hardware (ISA)

abgefangen und entsprechend verwaltet, bevor diese schließlich auf der Hardware ausgeführt oder emuliert werden. Die VM selbst fasst ihre abgekapselte Umgebung als eigenständige physische Hardware auf.

Innerhalb der System-Virtualisierung sind weitere Unterscheidungen anhand einer gemeinsamen oder unterschiedlichen ISA möglich, wie Abbildung 2.10 zeigt. Hierbei existieren einerseits vollständig virtualisierte VMs mit VMM zur Bereitstellung isolierter Umgebungen, aber auch Whole-System-VMs mit der Möglichkeit der Emulation unterschiedlicher ISAs auf derselben physischen Hardware [SN05b, S. 19].

2.2.3.1 Vollständige System-Virtualisierung – direkt oder als Guest

Die Virtualisierungsschicht wird, wie bereits in Abschnitt 2.2.1.1 erläutert, bei der vollständigen Virtualisierung als *Hypervisor* bezeichnet und direkt auf dem Host-System ausgeführt [Gol73]. Der VMM ermöglicht eine Reproduktion der eigentlichen Hardware-Plattform und entsprechend auch der physischen Geräte. Hardware, die nicht für einen gleichzeitigen Zugriff von mehreren Betriebssystemen ausgelegt ist, wird vom VMM emuliert. Als zentraler Bestandteil werden die gemeinsamen Hardwareressourcen mehrerer abgekapselter Betriebssystem-Umgebungen zur Verfügung gestellt, wobei der VMM die Ressourcen verwaltet und den Zugang zu diesen bereitstellt.

Die vollständige Virtualisierung mit gleicher ISA wird nach Goldberg [Gol73] in zwei grundlegende Architekturen von Systemen unterteilt:

- Typ 1: klassische System-VM oder Bare-Metal-Virtualisierung¹⁸ (über VMM oder Hypervisor) und
- Typ 2: Hosted VM¹⁹ (VMM eingebettet in Host-Betriebssystem).

Das Verhalten zwischen VMM (Host) und VM (Gast) ist daher vergleichbar mit einem konventionellen Betriebssystem mit unterschiedlichen Sicherheitsstufen in Form von privilegierten Betriebssystem-(System-Mode) und nicht privilegierten Prozessen (User-Mode) [TB14, S. 76], wie in Abbildung 2.11(a) gezeigt. Befindet sich der VMM wie in Abbildung 2.11(b) im Modus mit den höchsten Privilegien, wird von einer nativen (oder Bare-Metal) Virtualisierung gesprochen. Der VMM wird in diesem Fall auch als *Typ 1-Hypervisor* bezeichnet [PG74].

Führen die VMs Befehle aus, welche höhere Privilegien erfordern, erzeugen diese eine Ausnahme (Exception) innerhalb des VMMs, der jede dieser Ausnahmen einzeln emuliert, um kritische Hardwareressourcen vor unkontrolliertem Zugriff zu schützen²⁰.

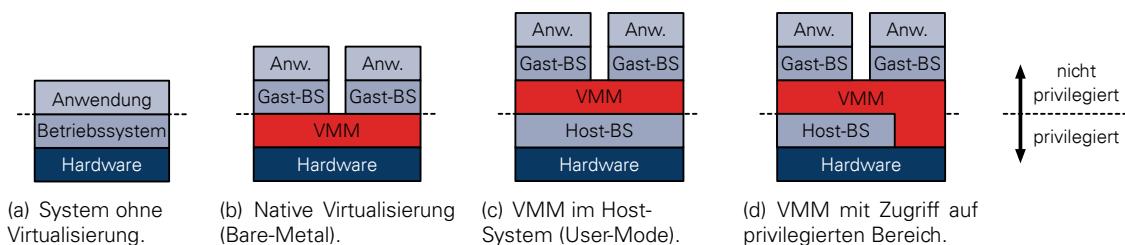


Abbildung 2.11: Übersicht zu nativer und Hosted-Virtualisierung mit Privilegierung. Nach [SN05b, S. 380].

¹⁸Eine klassische Typ 1-Virtualisierung sind Hardware Virtual Machine (HVM)-Instanzen auf einem XEN-Hypervisor [Bar+03].

¹⁹Varianten der Typ 2-Virtualisierung sind beispielsweise VMware [Bug+12] oder die Kernel-based Virtual Machine (KVM) [Kiv+07].

²⁰Bei der x86-Architektur wird die Nutzung des Befehlssatzes des Prozessors über das Schutzkonzept der Ringe kontrolliert (siehe Abschnitt 2.2.3.3).

Befindet sich der VMM selbst als Anwendung innerhalb eines Host-Betriebssystems und somit im nicht-privilegierten Modus, während das Betriebssystem selbst sich im privilegierten Modus befindet, wie in Abbildung 2.11(c) dargestellt, wird von einer *Hosted Virtualisierung* und einem *Typ 2-Hypervisor* gesprochen [PG74]. Hat der VMM zusätzlich noch direkten Zugriff auf die physische Hardware und somit auf den Bereich mit höheren Privilegien, liegt eine Dual-mode Hosted-xVirtualisierung vor, wie in Abbildung 2.11(d) gezeigt. Zugriffe auf die Hardware mit entsprechend höheren Privilegien werden komplett vom VMM über das Host-Betriebssystem verwaltet [SN05a].

Ein Typ 2-Hypervisor innerhalb eines Host-Betriebssystems bietet eine hohe Flexibilität, allerdings wird dadurch die Sicherheit deutlich reduziert, da sämtliche Schwachstellen des Host-Betriebssystem für Angriffe auf das Gesamtsystem mit allen Gast-VMs ausgenutzt werden können. Ein Typ 1-Hypervisor bietet aufgrund seiner geringen Komplexität deutlich weniger Angriffsfläche und somit ein höheres Maß an Sicherheit, wodurch das Gesamtsystem mit allen Gast-VMs und deren Daten besser abgesichert sind. Aufgrund dessen sind gerade in Rechenzentren vermehrt Typ 1-Virtualisierungen im Einsatz [Sca11].

2.2.3.2 Whole-System und Codesigned VMs

Bei den bisherigen System-Virtualisierungen nutzen sowohl Gast als auch VMM oder Host-Betriebssystem dieselbe ISA. Sind die ISAs unterschiedlich, ist ähnlich wie in Abschnitt 2.2.2.2 eine Emulation von Betriebssystem und Anwendungen durch den VMM erforderlich. Diese Art der Virtualisierung benötigt typischerweise einen Typ 2-Hypervisor, welcher die Emulation durchführt und Whole-System-VMs ermöglicht [SN05a].

Im Gegensatz zu den bisher betrachteten System-Virtualisierungen besteht das Ziel von *Codesigned VMs* darin, die Leistungsfähigkeit durch neuartige ISAs zu erhöhen. Der VMM ist dabei in einer speziellen Speicherregion des Prozessors angesiedelt und seine Aufgabe besteht in der Übersetzung von Gast- in Host-ISA [SN05a].

2.2.3.3 Privilegien, hardwareunterstützte System-Virtualisierung und Paravirtualisierung

Wie bereits in Abbildung 2.11 gezeigt, unterscheiden sich die Privilegien von Host-Betriebssystem oder VMM von denen der Gast-VMs. Die Ausführung von Befehlen aus den VMs ist nur im Modus mit geringeren Privilegien möglich. Führt die VM aber einen Befehl aus, der höhere Privilegien erfordert, führt dieses zwangsläufig zu einer Exception innerhalb des VMMs. Dieser muss darauf entsprechend reagieren und den Befehl emulieren, um Zugriff auf Hardweareressourcen abzusichern oder entsprechend zu verwalten [SN05a].

Privilegienstufen

Bei x86-kompatiblen²¹ Architekturen werden vier Privilegierungsstufen in Form von Ringen unterschieden, wobei der innerste Ring (Ring 0), die höchste Sicherheitsstufe darstellt (Kernel-Modus) und die Privilegierung der weiteren Ringe (1 - 3) immer weiter abnimmt (Nutzer-Modus), siehe Abbildung 2.12(a). Die Ringe schränken den von Prozessen nutzbaren Befehlssatz und ebenso den verwendbaren Speicherbereich ein, um Stabilität und Sicherheit des Systems zu erhöhen. Nur in Ring 0 hat das Betriebssystem

²¹Die x86-Kompatibilität steht für eine kompatible Befehlssatzarchitektur (ISA-Kompatibel).

2 Stand der Forschung und Technik

Zugriff auf den vollen Befehlssatz und die komplette physische Hardware²². Mit Hilfe der Ringe ist es möglich, Prozesse in ihren Privilegien voneinander abzuschotten [Spr+07, S. 40].

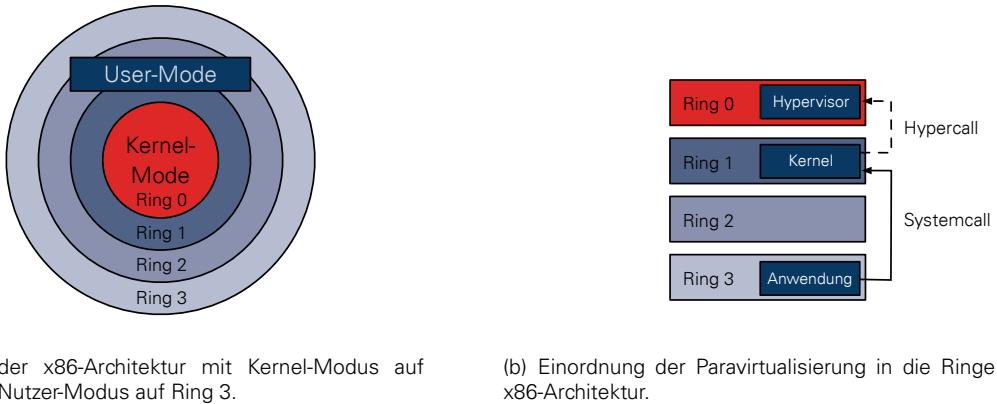


Abbildung 2.12: Paravirtualisierung und die Ringe der x86-Architektur. Nach [Spr+07, S. 41].

Die Virtualisierung von x86-Systemen bedient sich der Tatsache, dass typische Systeme lediglich zwei Ringe nutzen. Bei der Typ 2-Virtualisierung befindet sich das Host-Betriebssystem mit dem Hypervisor in Ring 0, die Gast-Betriebssysteme in Ring 1 oder Ring 2 und die Anwendungen auf Ring 4. Der VMM stellt für jede Ausnahme bei Zugriffen, welche höhere Privilegien erfordern, eine entsprechende Behandlung bereit, welche den Befehl oder Zugriff interpretiert und ausführt [Spr+07, S. 40].

Hardwareunterstützte Virtualisierung – Hardware Virtual Machine (HVM)

Erst mit der Einführung der *hardwareunterstützten Virtualisierung* ab 2005 durch *Virtualization Technology (VT)* von Intel und *Secure Virtual Machine (SVM)* von AMD ist die Ausführung von höher privilegierten Operationen der Gast-VMs direkt in Hardware möglich. Eine wesentliche Änderung besteht darin, dass zusätzlich ein Ring -1 für den Hypervisor in Hardware geschaffen wurde und Betriebssysteme dadurch wie gewohnt Befehle in Ring 0 ausführen können. Die VMs, welche durch hardwareunterstützte Virtualisierung völlig unverändert ausgeführt werden können, werden als HVMs bezeichnet. Da der Hypervisor Befehle nicht mehr emulieren muss, ist die Leistungsfähigkeit des Gesamtsystems höher als bei anderen Virtualisierungen [Spr+07, S. 44].

Paravirtualisierung

Im Rahmen der Typ 1-Virtualisierung kann eine *Paravirtualisierung* eingesetzt werden, um auf physische Geräte über virtuelle Treiber zugreifen zu können. Es wird eine API bereitgestellt, die sich ähnlich, aber nicht identisch zu der Schnittstelle mit der physischen Hardware verhält. Der Hypervisor ermöglicht den Zugriff auf die physische Hardware über die von der Gast-VM bereitgestellte API. Der VMM beinhaltet in diesem Fall ein auf ein Minimum reduziertes, sogenanntes *Metabetriebssystem*. Das eigentliche Gast-Betriebssystem muss übertragen werden, um auf der VM ausgeführt werden zu können und die entsprechende API für den Treiber innerhalb des Hypervisors zu nutzen, wie in Abbildung 2.13 dargestellt. Der virtuelle Treiber als Modifikation des Gast-Betriebssystems wird dabei als *Frontend-Treiber* und die

²²Die meisten Betriebssysteme arbeiten lediglich auf Ring 1 und 3, weshalb eine Vielzahl von Prozessor-Architekturen lediglich zwei Ringe bietet [Spr+07, S. 40].

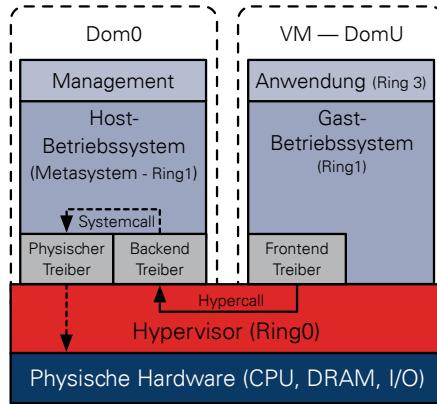


Abbildung 2.13: Paravirtualisierung mit Treiber innerhalb des Gast-Betriebssystems. Nach [Spr+07, S. 43].

Schnittstelle (API) innerhalb des Hypervisors als *Backend-Treiber* bezeichnet. Der Backend-Treiber greift mittels Systemaufruf auf den physischen Treiber zu.

Sowohl Host-, als auch Gast-Betriebssystem werden in Ring 1 ausgeführt, der Hypervisor in Ring 0 stellt sogenannte *Hypercalls* zur Verfügung, um die Kommunikation vom Gast-System mit dem Hypervisor zu ermöglichen. Die Hypercalls bilden eine zusätzliche Abstraktionsschicht zwischen Gast und Hypervisor und stellen eine Erweiterung des x86-Befehlssatzes dar. Wie Abbildung 2.12(b) zeigt, nutzt eine Anwendung in Ring 3 klassische Systemaufrufe, um mit dem Frontend-Treiber des Gast-Systems in Ring 1 zu kommunizieren. Dieser spricht wiederum über den Hypercall den Hypervisor in Ring 0 mittels des Backend-Treibers an, um Befehle im höherprivilegierten Modus ausführen zu können.

Für die Paravirtualisierung sind demnach sowohl modifizierte Host- als auch Gast-Betriebssysteme notwendig [Spr+07, S. 41]. Die Virtualisierung mittels eines Xen-Hypervisors [Pic09] ermöglicht neben der Paravirtualisierung auch eine klassische Typ 1-Virtualisierung mit Hardware Virtual Machine-Instanzen²³. Das Host-Betriebssystem wird dabei als *Dom0* bezeichnet und hat als privilegierte Domain vollen Zugriff auf die Hardware. Die Gast-Systeme innerhalb der VMs werden als *DomU* bezeichnet [Pic09, S. 19].

2.2.3.4 Zustände und Migration von System-VMs

VMs können wie normale Systeme gestartet, heruntergefahren, neu gestartet und abgebrochen werden. Speziell bei VMs besteht des Weiteren die Möglichkeit, ein System vollständig anzuhalten und den kompletten Zustand mit Prozessor und Speicher einzufrieren oder ein System zwischen physischen Rechnern zu *migrieren*. Besonders für Virtualisierungen in Rechenzentren oder Cloud-Systemen ist es aufgrund von Wartungen der Hardware oder Lastschwankungen mitunter notwendig, eine VM von einem physischen Host auf einen anderen Host zu übertragen. Die Migration lässt dabei sich in drei unterschiedliche Konzepte, je nach Zwischenzustand und Erreichbarkeit, unterteilen [Spr+07, S. 58]:

Kalte Migration: Die VM wird auf Host 1 komplett heruntergefahren und anschließend auf Host 2 neu gestartet. Die Hosts haben dabei Zugriff auf ein gemeinsames Dateisystem. Das Herunterfahren und der Neustart benötigen zwar mehrere Sekunden, die Übertragung ist aber insofern sicher, als dass keine inkonsistenten Daten entstehen können.

²³Bei Prozessoren mit Hardware-Virtualisierung ist eine zusätzliche Ausführung von unmodifizierten Betriebssystemen in einer paravirtualisierten Xen-Umgebung möglich [Pic09, S. 115].

Warne Migration (bei Xen Offline Migration): Die VM wird auf Host 1 lediglich gestoppt und anschließend eingefroren. Der komplette Zustand der VM mit Arbeitsspeicher und Registern wird auf Host 2 übertragen, und die VM wird anschliessend auf Host 2 aufgetaut und fortgesetzt. Die Migration benötigt hierbei deutlich weniger Zeit als bei der kalten Migration. Wie bei Letzterer wird ebenfalls ein gemeinsames Dateisystem zur Datenübertragung eingesetzt.

Live Migration (bei Xen Online Migration): Der Arbeitsspeicher von Host 1 wird während des Betriebs der VM kopiert und auf Host 2 übertragen; Änderungen im Speicher während des Vorganges werden als verändert markiert. Anschließend wird die VM gestoppt und auf Host 2 fortgesetzt, nachdem die letzten Änderungen übertragen worden sind. Das Vorgehen wird als *Pre-copy Memory Migration* bezeichnet. Der Wechsel ist dabei in wenigen Millisekunden möglich [Cla+05]. Das unter Umständen mehrfache Übertragen von Speicherbereichen wird in der *Post-copy Memory Migration* vermieden, indem zunächst der Wechsel auf Host 2 erfolgt und erst dann die benötigten Speicherbereiche übertragen werden [Liu+13].

2.2.3.5 Container-basierte Virtualisierung von Betriebssystemen

Neben den bisher erläuterten System-Virtualisierungen, welche komplett Betriebssysteme in VMs kapseln, existiert die Möglichkeit einer *container-basierten Virtualisierung*. Damit können voneinander abgekapselte und isolierte, identische Systemumgebungen auf demselben Betriebssystem bereitgestellt werden, wobei vom Betriebssystem aber mehrere Instanzen erzeugt werden [Sol+07]. Anwendungen können nur mit anderen Anwendungen im gleichen Container oder im bereitgestellten Betriebssystem interagieren. Die Container gewährleisten des Weiteren die Trennung der auf einem Rechner gemeinsam genutzten Ressourcen, wodurch ein Container keinen Zugriff auf Ressourcen hat, welche von anderen Containern genutzt werden [Mer14].

Die container-basierte Virtualisierung besitzt den Vorteil, dass durch die zugrundeliegenden gleichartigen Ressourcen des Betriebssystems die Verwaltung und Nutzung von gemeinsamen Geräten ohne komplexe Inter-Domain Kommunikationen erfolgen kann. Entsprechend sind die erreichbaren Leistungswerte für Linux-Container (LXC) [Can17] annähernd wie die in einem System ohne Virtualisierung, wie Xavier et al. in [Xav+13] gezeigt haben.

2.2.4 Virtualisierungsansätze für spezielle Geräte

Die Virtualisierung von Ein- und Ausgabegeräten (I/O) ist besonders anspruchsvoll, da jedes Gerät seine eigenen Charakteristiken und Bedürfnisse hat, welche in einem VMM oder Hypervisor berücksichtigt werden müssen. Techniken, um I/O-Geräte zu teilen und in Zeitscheiben auf sie zuzugreifen, wurden schon in den frühen Betriebssystemen eingesetzt, und viele der Ansätze wurden auch in der Virtualisierung entsprechend berücksichtigt.

Die häufigste Methode, um I/O-Geräte zu virtualisieren, besteht in der Konstruktion einer virtuellen Version des Gerätes und der reinen Virtualisierung der I/O-Aktivität des Gerätes. Der VMM ordnet die Zugriffe auf das virtuelle Gerät dem physischen Gerät zu und verwaltet ebenso Zugriffe aus den verschiedenen VMs. Die Paravirtualisierung aus Abschnitt 2.2.3.3 stellt dabei eine Sonderform dar, da das virtuelle Gerät lediglich über eine API/Hypercalls mit dem Hypervisor und somit dem realen physischen Gerät kommuniziert. Die grundlegenden Techniken und theoretischen Ansätze zur I/O-Virtualisierung werden in der Arbeit von Smith et al. [SN05b, S. 404-415] und in Abschnitt A.2 vorgestellt.

2.2.4.1 Hardwareunterstützte Virtualisierung von PCIe-Geräten

Eine spezielle Rolle nehmen aufgrund ihrer typischerweise hohen Datenraten PCIe-Geräte ein. Um Geräte exklusiv zu nutzen, ist des Weiteren ein direkter Zugriff auf die Hardware - am Hypervisor vorbei - erforderlich. In der Regel erfordert das einen Prozessor mit hardwareunterstützter Virtualisierung [Spr+07, S. 197]. Bei PCI-basierter Hardware ist bei einer Xen-Virtualisierung ein Durchreichen an eine unterprivilegierte VM mit einem sogenannten *PCI-Backend-Treiber* oder *PCI-Passthrough* ohne Einbußen in Kommunikations-Latenz und Bandbreite möglich. Dabei wird bei DMA-Geräten der Speicherbereich des Gerätes durch die *Input/Output-Memory Management Unit (IOMMU)* nicht in den Speicher des Hypervisors eingeblendet und an die VMs weitergereicht, sondern über I/O-MMU-Virtualisierung direkt in den Speicher der VM eingeblendet [TB14, S. 596].

Partitionieren oder die gemeinsame Nutzung von unterschiedlichen VMs ist oftmals nur über eine Paravirtualisierung wie in [Kir+12] möglich. Der Hypervisor verwaltet dabei die unter Umständen zeitgleichen Zugriffe auf das Gerät. Eine andere Möglichkeit bei PCIe-Geräten besteht in der *Single Root-Input/Output-Virtualization (SR-IOV)*²⁴, die es ermöglicht, ein partitioniertes Gerät an unterschiedliche VMs durchzurichten [Suz+10]. SR-IOV wird typischerweise von Netzwerkkarten in Rechenzentren oder im HPC-Bereich [Lac+14], aber auch von ausgewählten Xilinx Virtex-7 FPGAs mit PCIe 3.0-Endpunkt unterstützt [Sur13].

2.2.5 Virtualisierungsarten und deren Relevanz für eine FPGA-Virtualisierung

Die in den vorherigen Abschnitten vorgestellten Virtualisierungsarten nach der Taxonomie aus Abschnitt 2.2.1.2 sind in Tabelle 2.4 zusammengefasst. Bei der Charakterisierung der Virtualisierungsarten sind zum einen die ISAs eine wesentlicher Aspekt, wie auch die Kapselung der VMs bei der System-Virtualisierung beziehungsweise der Prozesse bei der Prozess-Virtualisierung. Die Eignung der jeweiligen Virtualisierung für eine Übertragung auf einen FPGA wird in der Tabelle ebenso qualitativ aufgezeigt.

Tabelle 2.4: Charakterisierung der Virtualisierungsarten und Relevanz für eine FPGA-Virtualisierung.

Prozess-Virtualisierung	gleiche ISA	gekapselte VMs	Eignung für FPGAs
Multiprogrammbetrieb	✓	✓	!
Dynamische binäre Optimierung	✓	✗	✗
Dynamische binäre Übersetzung (Emulatoren)	✗	✗	✗
HLL VMs	✗	✗	✗
System-Virtualisierung			
Klassische System-VMs (nativ, Bare-Metal)	✓	✓	✓
Hosted VMs	✓	✓	!
Whole System VM	✗	✓	✗
Codesigned VM	✗	✓	✗

✓: optimal !: eingeschränkt ✗: problematisch

Da die Hardwaredesigns innerhalb der VMs direkt auf der physischen FPGA-Hardware ausgeführt werden sollen, sind Virtualisierungen mit gleicher ISA von VM und Host-System notwendig. Eine Eignung der Prozess-VMs ist aufgrund des erforderlichen Betriebssystems zwischen dem Prozess und der physischen Hardware mit den in Abschnitt 2.1.4.3 vorgestellten Overlay-Architekturen vergleichbar. Durch ih-

²⁴Das PCI-Special Interest Group (PCI-SIG)-Konsortium stellt unterschiedliche standardisierte I/O-Virtualisierungsmethoden basierend auf PCI Express bereit.

re Abstraktion kann die Prozess-Virtualisierung mit Overlays auf rekonfigurierbarer Hardware verglichen werden. Der Aufbau einer eigenen FPGA-Architektur durch Overlays entspricht somit der Emulation innerhalb einer Prozess-Virtualisierung für eine fremde ISA.

Die System-Virtualisierung bildet hingegen eine Abstraktionsschicht zwischen Anwendung und Betriebssystem sowie der physischen Hardware. Im Fall von FPGAs bedeutet dies, dass sowohl die eigentliche Anwendung als auch Strukturen innerhalb der gekapselten vFPGAs in den Verantwortungsbereich des Nutzers fallen und ein zusätzlicher Hypervisor auf dem FPGA die physischen Schnittstellen und Zugriffe verwaltet. Da ein Äquivalent zu einem vollwertigen Betriebssystem in den unterschiedlichen FPGA-Virtualisierung nicht zweifelsfrei identifiziert werden kann, ist eine Unterscheidung zwischen Prozess- und System-Virtualisierung nicht eindeutig möglich.

Bei der System-Virtualisierung sind zusätzliche Zwischenschichten zu vermeiden, wie sie bei einer Hosted-VM notwendig sind. Als möglicher Kandidat für eine FPGA-Virtualisierung eignet sich nach Abwägung der unterschiedlichen Virtualisierungen in Tabelle 2.4 die klassische System-VM als möglicher Ausgangspunkt. Die Paravirtualisierung kann unter den System-VMs eingeordnet werden und stellt als Virtualisierung für die Treiber und den Zugriff auf die physischen Schnittstellen bei einer FPGA-Virtualisierung eine weitere Möglichkeit dar.

2.3 Cloud-Computing

Im Folgenden werden in Abschnitt 2.3.1 zunächst die Entwicklung des Cloud-Computings sowie die wichtigsten direkt in Zusammenhang stehenden Begriffe vorgestellt. Darauf folgt in Abschnitt 2.3.2 eine Übersicht zu den wesentlichen Bestandteilen und Schlüsseltechnologien des Cloud-Computings. Aktuelle Herausforderungen innerhalb des Themengebietes werden schließlich in Abschnitt 2.3.3 skizziert.

2.3.1 Historische Entwicklung, Definition und Begriffe

Die ursprünglichen Ansätze zu einer *Cloud* gehen zurück auf die ersten Ideen zum *Utility Computing* von John McCarthy 1961 [McC61], Parkhill 1966 [Par66] und Vorschläge zu deren zukünftiger Vernetzung von Leonard Kleinrock im Jahre 1969 [Kle69].

Das Konzept des Cloud-Computing basiert darauf, dass Nutzer Zugang zu gemeinsamen Ressourcen oder Diensten (Services) erhalten, die nach Belieben allokiert/genutzt und wieder freigegeben werden können. Dabei ist ein wesentliches Kriterium eine möglichst geringe Interaktion mit dem Anbieter der Cloud oder dem Betreiber des Rechenzentrums [MG11]. Ein essenzieller Unterschied zu traditionellen Rechenzentren, bei denen der Nutzer eine feste Anzahl von Ressourcen mietet, besteht im Konzept der Elastizität der Ressourcen in der Cloud. Dabei ist die Anzahl der Ressourcen für den Nutzer theoretisch unbegrenzt, da immer genau die erforderliche Menge an Rechenknoten und Speicher zur Verfügung steht [Brä+12; EPM13].

Die neueren und konkreteren Entwicklungen zu Clouds in ihrer heutigen Form sind in Abbildung 2.14 dargestellt [EPM13]. Ein erster Schritt war dabei die Umsetzung der alten Idee des *Utility Computings* [McC61] im *Grid Computing*, wobei verteilte Rechenressourcen Wissenschaftlern – ähnlich wie in einem Stromnetz – in Batch-Verarbeitung bereitgestellt wurden [Fos+08]. Der Begriff des *Utility Computing* wurde 1997 erneut aufgegriffen, wobei vermehrt die Bereitstellung von Rechenressourcen und ganzer Geschäftsprozesse [Plu+08], welche nach tatsächlicher Nutzung (Zeit, Auslastung, Energiebedarf etc.) abgerechnet wurden, an Bedeutung gewann. Mit der direkten Bereitstellung von Software, aber auch

von kompletten Diensten wie den Amazon Web Services (AWS), welche über das Internet bereitgestellt werden, etablierte sich das System *Software as a Service (SaaS)*. Der entscheidende Unterschied zum traditionellen Cloud-Computing besteht darin, dass bei diesem die Bereitstellung von Diensten und Hardware bis auf die untersten Ebenen reicht, SaaS-Computing aber lediglich die oberste Ebene des Cloud-Computing abdeckt [Plu+08].

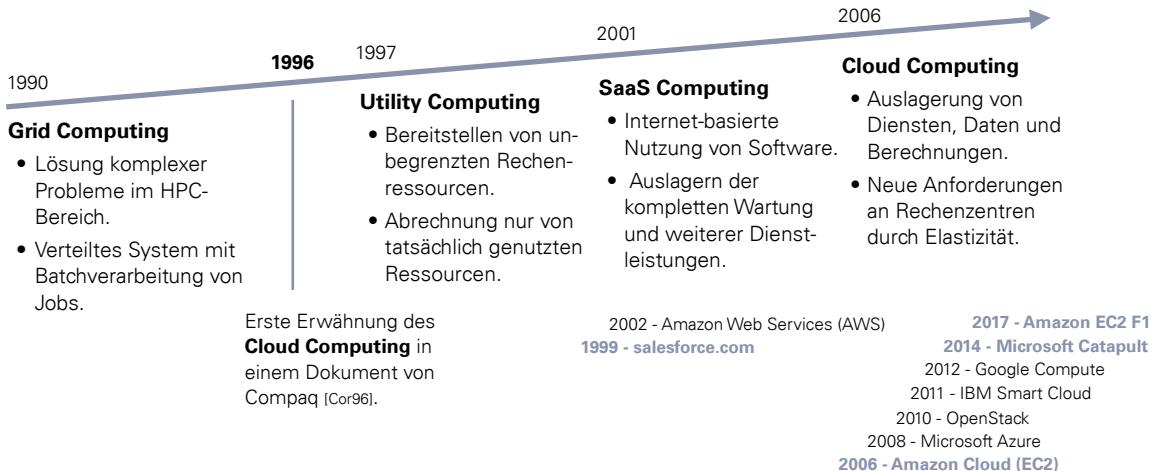


Abbildung 2.14: Historische Entwicklung des Cloud-Computings und seiner Vorgängersystemen. Daten aus [BF16; Brä+12; EPM13; Kav14].

Der Begriff Cloud-Computing umfasst die Bereitstellung von Ressourcen auf sämtlichen Ebenen, also auch das direkte Anbieten von (virtuellen) Rechenressourcen und Infrastruktur. Der eigentliche Begriff des Cloud-Computings geht dabei zurück auf ein internes Dokument von Compaq aus dem Jahr 1996 [Cor96] und hat sich etabliert, da es die Ansammlung von Ressourcen ohne direkte physische Repräsentation ideal beschreibt. Durch eine Definition von Gartner [Fos+08], Foster et al. [Fos+08] und das National Institute of Standards and Technology (NIST) von Mell und Grace aus dem Jahr 2009²⁵ [MG09] erlangte der Begriff schließlich allgemeine Akzeptanz.

2.3.1.1 Definition und Begriffe des Cloud-Computing

Bedeutung und Abgrenzung des Cloud-Computing wurden unter anderem definiert von Gartner durch Plummer et al. [Plu+08] und Forrester Research durch Staten [Sta+08]. Letztendlich durchgesetzt hat sich folgende Definition (Definition 2.4) von Mell und Grace [MG09], welche auch für diese Arbeit genutzt wird, um Aussagen über die Möglichkeiten der Integration von FPGAs in eine Cloud-Architektur treffen zu können:

Definition 2.4: Cloud-Computing *Cloud-Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models²⁶. [MG11, S. 2]*

²⁵Überarbeitete Version aus dem Jahr 2011 [MG11].

²⁶Freie Übersetzung von [MG11, S. 2]: Cloud-Computing ist ein Modell zur Realisierung eines allgegenwärtigen, bequemen und bedarfsgerechten Netzwerkzugriffs auf einen gemeinsamen Pool von konfigurierbaren Rechenressourcen (Netzwerke, Server, Speicher, Anwendungen und Dienste), welche schnell und mit minimalem Verwaltungsaufwand bereitgestellt oder als Service freigegeben werden können. Dabei ist keine direkte Interaktion mit dem Anbieter der Cloud notwendig. Das Cloud-Modell besteht aus fünf wesentlichen Kriterien, drei Servicemodellen und vier Arten des Cloud-Betriebs.

2 Stand der Forschung und Technik

Die Ressourcen, welche in Form einer Cloud bereitgestellt werden können, sind demnach auf unterster Ebene Rechenressourcen wie Server, Hardwarebeschleuniger (in der Regel GPGPUs), Netzwerkressourcen und Datenspeicher sowie auf den höheren Ebenen VMs, Software, aber auch entsprechende Dienste.

Dabei gibt es fünf elementare Kriterien, sogenannte *Cloud Characteristics* (siehe Abschnitt 2.3.1.2), drei Modelle für Dienste, welche als *Cloud Service Models* bezeichnet werden (siehe Abschnitt 2.3.1.3) und das Cloud-Computing von ähnlichen Modellen abgrenzen, sowie vier Arten des Cloud-Betriebs (siehe Abschnitt A.3.2), die *Deployment Models* nach [MG11].

2.3.1.2 Kriterien

Um ein System als Cloud bezeichnen zu können, listet die Definition des Cloud-Computings von Mell und Grace [MG11] (siehe Definition 2.4) wesentliche Kriterien zur Abgrenzung zu traditionellen Rechenzentren und Angeboten auf. Im einzelnen sind die Kriterien [MG11] für das Cloud-Computing²⁷:

Selbstbedienung nach Bedarf (on-demand self service): Beschaffung von Ressourcen nach Bedarf in Eigeninitiative. Typischerweise ist dabei keine direkte Interaktion mit dem Anbieter der Cloud notwendig.

Gemeinsame Nutzung physischer Ressourcen (Resource Pooling): Der gesamte Pool an Ressourcen steht allen Nutzern gleichermaßen zur Verfügung. Wichtig ist dabei die Trennung der Daten, welche in der Regel eine Form der Virtualisierung (siehe Abschnitt 2.2) erfordert. Die Ressourcen können einem Nutzer dynamisch zugeteilt werden. Der Nutzer hat typischerweise durch die Virtualisierung keine Kenntnis oder Kontrolle über die physische Position der bereitgestellten Ressource.

Anpassung an den Ressourcenbedarf (Rapid Elasticity): Aus Nutzersicht sind die physischen Ressourcen unbegrenzt in ihrer Anzahl und Größe und können zu jeder Zeit beliebig erweitert oder reduziert werden.

Messung der Servicenutzung (Measured Service): Nur die tatsächliche Ressourcenauslastung wird gemessen und dem Kunden in Rechnung gestellt, wobei die Nutzung der Services vollkommen transparent sein soll.

Umfassender Netzwerkzugriff (Broad Network Access): Die bereitgestellten Ressourcen sind typischerweise über das Internet erreichbar, was die Verwendung standardisierter Netzwerkzugriffe erfordert, um möglichst vielen Endgeräten gleichzeitig Zugang zu gewähren.

2.3.1.3 Servicemodelle

Typische Angebote auf Cloud-Systemen können nach Mell und Grace [MG09] in drei Gruppen von Geschäftsmodellen eingeteilt werden. Die Modelle unterscheiden sich hinsichtlich der Ebene, auf welcher der Dienst angeboten wird, und sind essenziell für die Beschreibung eines Interfaces beziehungsweise einer Abgrenzung zum Nutzer. Diese sogenannten Servicemodelle (*Service Models*) werden im Folgenden kurz erläutert und sind in Abbildung 2.15 aufgezeigt.

²⁷Deutsche Übersetzungen nach Bräuninger et al. [Brä+12].

Software as a Service (SaaS): Der Nutzer erhält Zugang zu einer umgehend nutzbaren Software (Mail, Kalender, Dokumente, Speicher etc.), für deren Verwaltung und Betrieb der Anbieter des Dienstes vollständig verantwortlich ist. Die Abrechnung erfolgt typischerweise pro Aufruf, nach Nutzungsumfang oder -volumen (benötigter Speicherplatz). Die reale Anzahl an virtuellen und physischen Maschinen sowie deren Konfiguration oder das konkrete Betriebssystem bleiben dem Nutzer üblicherweise verborgen.

Platform as a Service (PaaS): Der Nutzer erhält die Möglichkeit, eigene Programme auf einer Plattform in der Cloud bereitzustellen. Die Nutzer erhalten eine Entwicklungs- oder Laufzeitumgebung (üblicherweise in Form von Virtuellen Maschinen), wobei der Nutzer allerdings keine Möglichkeit hat, die Art der zugrundeliegenden Virtualisierung oder die physischen Systeme zu sehen oder zu konfigurieren. Ein Beispiel für einen PaaS-Dienst ist Microsoft Azure [Mic16c] auf Basis von Containern oder Virtuellen Maschinen.

Infrastructure as a Service (IaaS): Der Anbieter der Cloud stellt in diesem Modell ausschließlich die gesamte (virtuelle) Infrastruktur bereit, welche der Nutzer nach seinen Wünschen konfigurieren kann. Auf diesem Level ist prinzipiell sogar Zugriff auf die physische Infrastruktur der Cloud möglich. Ein Beispiel hierfür stellt Amazons Elastic Compute Cloud (EC2) [Ama17b] oder Simple Storage Service (S3) [Ama17c] dar. Ein Teil des IaaS ist dabei die Real-Time-Infrastructure (RTI) mit einem möglichst tiefgreifenden Zugang zu den physischen Ressourcen mit einer hohen Elastizität [Plu+08].

Wie Abbildung 2.15 zeigt, bauen die Modelle aufeinander auf, wobei das Modell SaaS zur Bereitstellung eines Dienstes sowohl eine Infrastruktur (IaaS) als auch die reale physische Plattform (PaaS) benötigt. Stellt ein Service-Provider einen einfachen Dienst mittels des Modells SaaS bereit, sind die tieferliegenden Modelle automatisch mit inbegriffen, beziehungsweise werden diese durch andere Anbieter bereitgestellt und sind somit indirekt im Dienst mit inbegriffen, wobei der Nutzer keine detaillierten Kenntnisse über Plattform oder Infrastruktur hat. Konfigurationsmöglichkeiten auf den unteren Ebenen sind dabei ebensowenig möglich.

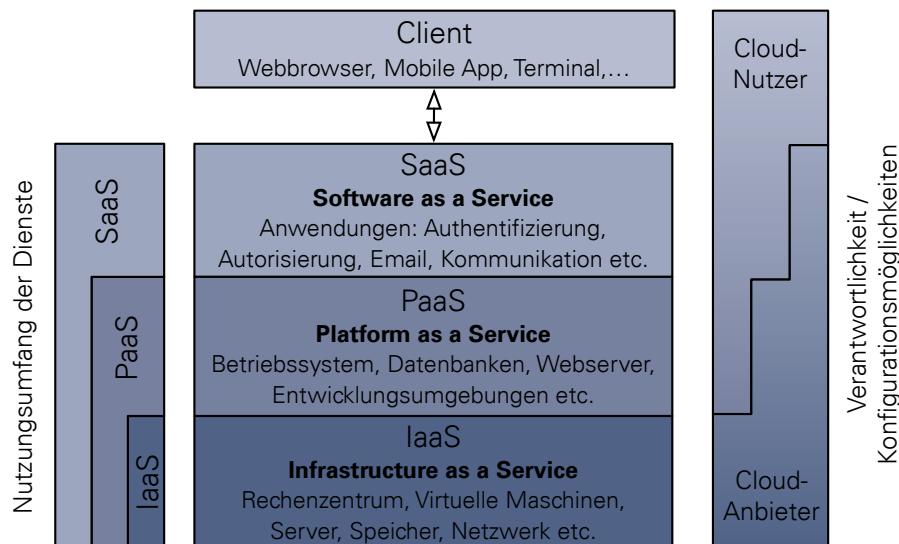


Abbildung 2.15: Cloud Service Modelle mit Umfang und Verantwortlichkeit, sowie Konfigurationsmöglichkeiten entsprechend des Service Modells. Nach [Kav14, S. 14].

Neben den aufgezeigten drei Geschäftsmodellen haben sich in den letzten Jahren weitere Dienste etabliert, welche sich unter dem Begriff *Everything as a Service (XaaS)* zusammen fassen lassen [Plu+08].

2 Stand der Forschung und Technik

Die Dienste in diesem Bereich lassen sich nicht konkret einem der anderen Geschäftsmodelle zuordnen, beziehungsweise sollen sie sich explizit von den etablierten Dienstmodellen abheben [Ban+11]. Zum Teil umfassen die hier inbegriffenen Dienste deutlich mehr Funktionen als die ursprünglich von Mell und Grace beschriebenen [Ban+11]. Beispiele dafür sind die Bereitstellung kompletter HPC-Ressourcen, wie in *High Performance Computing as a Service (HPCaaS)* [MKH12], oder auch spezielle Dienste mit stärkerem Fokus auf Daten und deren Verarbeitung ((Big)Data as a Service (DaaS)) [ZZL13] oder reinen Datenbanken (Database as a Service (DBaaS)) [AI 13]. Eine hierbei auftretende Schwierigkeit besteht in der Messung des Services und demzufolge in dessen Abrechnung. Die Komplexität von Messung und Abrechnung von XaaS wird dabei auch in der Literatur als neue Herausforderung des Cloud-Computing gesehen, wie Duan in [Dua12] oder Lenk et al. in [Len+09] aufzeigen.

2.3.1.4 Horizontale und vertikale Skalierung

Um den Begriff der *Elastizität* besser eingrenzen zu können, wurden im Cloud-Kontext die Begriffe *Horizontal Scaling* und *Vertical Scaling* [EPM13, S. 37] eingeführt, welche im Folgenden näher erklärt werden.

Horizontale Skalierung (Horizontal Scaling): Der Begriff bezeichnet das Allokieren zusätzlicher und die Freigabe ungenutzter Ressourcen desselben Typs. Horizontale Skalierung steht im Cloud-Computing mit dem Begriff *Elastizität*, also der automatischen Anpassung der Ressourcen an die aktuellen Bedürfnisse, in Zusammenhang.

Vertikale Skalierung (Vertical Scaling): Bei dieser Art der Skalierung werden einzelne Ressourcen durch größere oder leistungsfähigere Hardware ersetzt. Vertikale Skalierung wird aufgrund der Komplexität ihrer Verwaltung und Bereitstellung eher selten eingesetzt.

Wird im Bereich des Cloud-Computing von Elastizität gesprochen, ist somit typischerweise eine horizontale Skalierung mit Ressourcen des gleichen Typs gemeint. Skalierbarkeit und Elastizität sind wesentliche Anforderungen an Clouds [Plu+08] und von besonderer wirtschaftlicher Bedeutung für Unternehmen, welche Clouds einsetzen [Brä+12].

2.3.1.5 Rollenverteilung

Die unterschiedlichen Rollen der an dem Betrieb und der Nutzung einer Cloud beteiligten Akteure sind nicht zu vernachlässigen und auch für diese Arbeit von besonderen Interesse. Abbildung 2.16 liefert einen Überblick über eine typische Rollenverteilung nach Bohn et al. [Boh+11, S. 4]. Im Folgenden werden die beteiligten Akteure erläutert, um für die Arbeit eine einheitliche Begriffsdefinition sowie eine einheitliche Übersetzung zu etablieren. Die Basis der Klassifizierung bilden die von Erl et al. [EPM13, S. 52] und Bohn et al. [Boh+11] verwendeten Begriffe und Definitionen, die zunächst existentielle Rollen vorstellen:

Cloud-Nutzer (Cloud-Consumer): Eine Person oder eine Organisation, welche die Ressourcen der Cloud entsprechend der Modelle aus Abschnitt 2.3.1.3 von einem *Cloud-Anbieter* in Anspruch nimmt, wird als *Cloud-Nutzer* bezeichnet [Boh+11, S. 4]. Nutzer, welche auf dem Service aufbauend einen eigenen Dienst anbieten, können gleichzeitig ihrerseits zum Cloud-Anbieter werden.

Cloud-Anbieter (Cloud-Provider): Der *Cloud-Anbieter* ist eine Organisation, die Ressourcen entsprechend der in Abschnitt 2.3.1.2 gezeigten Kriterien anbietet. In vielen Fällen benötigen Anbieter weitere Cloud-Anbieter, um ihren Dienst entsprechend auf deren Services aufbauen zu können.

Der Anbieter ist für das Management der physischen Infrastruktur, beziehungsweise für die darüber liegenden Schichten verantwortlich, insbesondere für die Komponenten zur Bereitstellung der Dienste (siehe Abschnitt A.3.1) in Abhängigkeit des dem Kunden bereitgestellten Servicemodells [Boh+11, S. 7]. Abbildung 2.15 rechts gibt einen Überblick über die wachsende Verantwortung des Anbieters in Abhängigkeit des Servicemodells von IaaS bis zu SaaS.

Besitzer eines Dienstes (Cloud Service Owner): Der eigentliche Dienst, der dem Nutzer bereitgestellt wird, muss nicht einem speziellen Cloud-Anbieter gehören. Beispielsweise gibt es Dienste eines Anbieters, welche von mehreren Cloud-Anbietern eingesetzt werden, wie Werkzeuge zur Visualisierung oder Bearbeitung von Dokumenten [EPM13, S. 53].

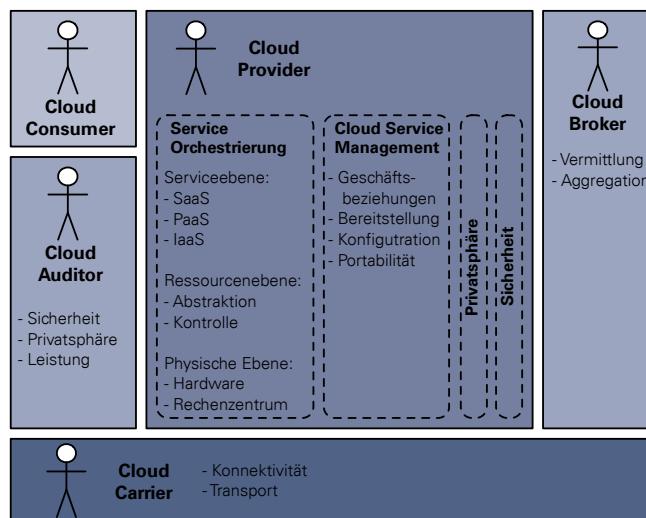


Abbildung 2.16: Rollen und beteiligte Akteure aus der Architektur-Definition des NIST von Bohn et al. [Boh+11, S. 4]. Neben Nutzer und Anbieter sind weitere optionale Rollen aufgelistet.

Aufbauend auf diesen Rollen haben sich noch weitere zusätzliche beziehungsweise optionale Rollen in den vergangenen Jahren herausgebildet, wie im Anhang unter Abschnitt A.3.3 aufgezeigt. Um Abgrenzungen bezüglich der Sicherheit der Nutzerdaten zu ermöglichen, wurden aufgrund der Vielzahl von Akteuren die Begriffe *Organization Boundary* und *Trust Boundary* etabliert [EPM13, S. 54].

2.3.2 Wesentliche Bestandteile und Schlüsseltechnologien

Die wesentlichen Technologien, welche das Cloud-Computing erst ermöglichen und daher auch essenzielle Komponenten für Cloud-Architekturen darstellen, ergeben sich direkt aus den Kriterien aus Abschnitt 2.3.1.2 und umfassen:

- Leistungsfähige und energieeffiziente Prozessoren,
- Effiziente Virtualisierungstechniken (für Prozessoren, Speicher, Netzwerke etc.),
- Große verteilte Speicher,
- Automatisiertes Management,
- Externer Zugriff auf Ressourcen zur Konfiguration und
- Breitband-Internetzugang.

2 Stand der Forschung und Technik

Hierbei kommt der Virtualisierung ein besonderer Stellenwert zu [Put+15] und diese wurde daher auch bereits in Abschnitt 2.2 näher betrachtet. Die sich daraus ergebenden Komponenten, welche ebenso in den etablierten Verwaltungssystemen OpenNebula [Ope16a] oder OpenStack [Ope16d] eingesetzt werden, sind unter anderem:

- Verwaltung der Rechenressourcen,
- Lastverteilung der Ressourcen (Load Balancer),
- Netzwerk-Manager (Networking),
- Image-Datenbank, Block- und Objekt-Speicher (Storage),
- Nutzerverwaltung (Identity),
- Verwaltungsinterface (Dashboard),
- Abrechnung (Billing/Accounting) und
- Überwachung der Qualität (SLA-Monitor oder Metering).

In Abschnitt A.3.1 wird weiterführend erläutert, wie diese Komponenten in Clouds eingesetzt werden, um die in Abschnitt 2.3.1.2 definierten Kriterien umzusetzen.

2.3.2.1 Lastverteilung und Scheduling

Das Verteilen der Last (Load Balancing) ist ein wichtiger Bestandteil innerhalb der Cloud-Architekturen. Algorithmen zur Lastbalance verfolgen unterschiedliche Optimierungsziele, wie beispielsweise die Priorisierung einer bestimmten Arbeitslast, eine asymmetrische Verteilung oder eine Bündelung der Arbeitslasten. Typischerweise beinhaltet die Lastverteilung mehrere Quality of Service (QoS)-Parameter, um entsprechend den Durchsatz zu maximieren oder Überlastung zu vermeiden. Das Modul zur Lastverteilung kann hierbei an unterschiedlichen Stellen angebunden sein, wie beispielsweise mittels Netzwerkswitches, spezieller Hardware, dedizierter Module im Hypervisor oder Service Agenten im Kern der Cloud-Management-Software [EPM13, S. 176].

Um in einem großen System das Freigeben von ungenutzten Ressourcen effizient betreiben zu können, ist unter Umständen auch die Migration von Diensten [KM11] oder kompletten VMs auf andere physische Rechenknoten zur Optimierung der Auslastung des Gesamtsystems [BBA10] erforderlich. Diese Optimierungsprobleme führen zur Entwicklung von komplexen Algorithmen zur Verteilung der Nutzeranfragen auf die Ressourcen, wie von Lee et al. in [LCK11] beschrieben, oder zu einem effizienten Scheduling wie in der Arbeit von Shainer et al. in [Sha+09] vorgestellt. Ein Scheduling stellt dabei eine genauere Ablaufplanung dar als die einfache Lastverteilung. Zhong et al. [ZTZ10] stellen unterschiedliche Algorithmen zum statischen Scheduling vor, welche auf die Infrastruktur (IaaS) aufsetzen. Effizienter ist die Ablaufplanung von Fang et al. [FWG10] mit Berücksichtigung der aktuellen Auslastung. Das zugrundeliegende Problem ist nicht cloud-spezifisch, sondern wird für verteilte Systeme im Allgemeinen bereits seit Jahrzehnten thematisiert [SRC85]. Die Untersuchung von Algorithmen zur Lastverteilung erfolgt typischerweise mit Cloud-Simulatoren wie CloudSim [Cal+11] oder Simulatoren für Rechencluster wie CHERUB [KZS13], mit denen die Simulation kompletter Rechenzentren möglich ist.

Für energieeffiziente Clouds gewinnt das Problem der Lastverteilung zunehmend an Bedeutung, da ein Ansatz darin besteht, ungenutzte physische Ressourcen in Zeiten schwacher Nachfrage komplett abzuschalten, wie in den Arbeiten von Kertscher et al. [KS15a] und Knauth et al. [Kna14] vorgeschlagen.

2.3.2.2 Elastizität in Cloud-Architekturen

Wichtige Anforderungen an Cloud-Architekturen sind die Kriterien der *Elastizität*. Die Elastizität von Cloud-Ressourcen ist dabei nach Owens [Owe10] das bedeutendste Alleinstellungsmerkmal des Cloud-Computings. Die Notwendigkeit der Elastizität ergibt sich direkt aus dem Cloud-Kriterium (siehe Abschnitt 2.3.1.2) der schnellstmöglichen und automatisierten Anpassung der nutzbaren Ressourcen an den aktuellen Ressourcenbedarf [Boh+11, S. 15].

Elastizität in der Cloud bezeichnet nach [HKR13] die adaptive Anpassung der Menge der Ressourcen an die aktuelle Last durch Skalierung, mit dem Ziel, dabei möglichst effizient exakt die Menge an Ressourcen bereitzustellen, welche im Moment gerade benötigt wird. Die Bereitstellung einer elastischen Cloud, das Buchen und Freigeben von Ressourcen und die Kostenersparnis in Abhängigkeit unterschiedlicher Zielstellungen und SLAs (siehe Abschnitt 2.3.2.3) werden ausführlich von Sharma et al. in [Sha+11] untersucht. Typischerweise wird die Elastizität auf der Ebene IaaS durch das schnelle Starten und Herunterfahren von VM-Instanzen ermöglicht, wobei auf den Ebenen PaaS und SaaS für den Nutzer völlig intransparent eine automatisierte, elastische Cloud bereitgestellt wird, sobald steigende Anfragen von Nutzern in Spitzenzeiten eine entsprechend größere Rechenleistung, oder mehrere zusätzliche VMs erfordern.

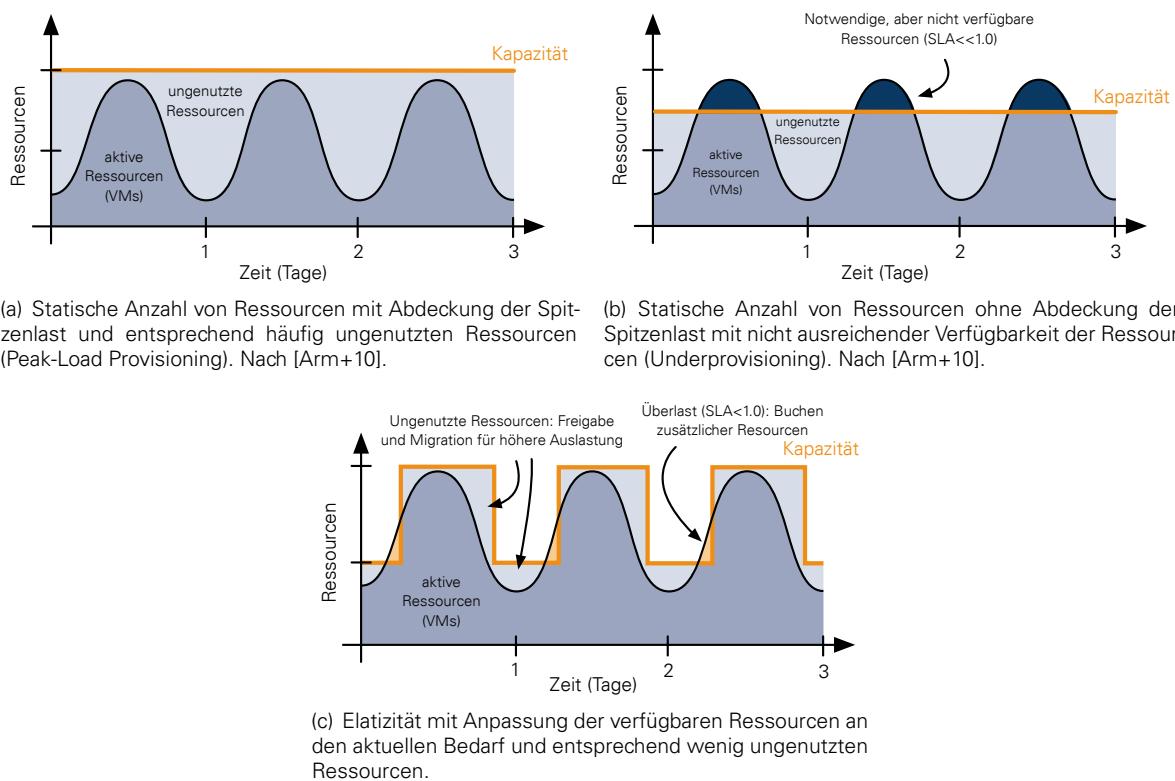


Abbildung 2.17: Szenarien zur Auslastung und reale Nutzung von Ressourcen in einer elastischen Cloud mit Service Level Agreement (SLA) (siehe Abschnitt 2.3.2.3).

Abbildung 2.17 zeigt die typischen Probleme bezüglich der Abdeckung der Anforderungen in Spitzenzeiten, aber mit ungenutzten Ressourcen in Zeiten schwacher Nachfrage (Peak-Load Provisioning) in Abbildung 2.17(a). Abbildung 2.17(b) hingegen zeigt eine Cloud, welche der Nachfrage in Zeiten hoher Auslastung nicht nachkommen kann und somit zu einer nicht ausreichenden Verfügbarkeit (siehe SLA in

2 Stand der Forschung und Technik

Abschnitt 2.3.2.3) führt. Durch flexibles, automatisiertes und horizontales Skalieren der Ressourcen (siehe Abschnitt 2.3.1.4) wird dagegen die *Elastizität* einer Cloud ermöglicht, wie sie in Abbildung 2.17(c) dargestellt ist. Der Nutzer hat nur die Ressourcen zur Verfügung, welche aktuell benötigt werden [Arm+10].

2.3.2.3 Metrik zur Bewertung von Cloud-Architekturen – Service Level Agreement (SLA)

Die Bewertung und insbesondere die Messung der Leistungsmetriken von Cloud-Diensten und kompletten Cloud-Architekturen ist typischerweise ein komplexes Problem, welches häufig mit Hilfe von Benchmarks gelöst wird [SK12] und mittlerweile durch Frameworks für unterschiedliche Clouds untersucht werden kann [GVB13]. Da die Bereitstellung von Diensten eine wesentliche Aufgabe darstellt, ist der Erfüllungsgrad des SLA (siehe Definition 2.5) eine der wichtigsten Messgrößen zur Beurteilung der Qualität einer Cloud [EPM13, S. 403]. Weiterhin ist eine feingranularere Einteilung in Qualität, Verfügbarkeit, Zuverlässigkeit, Leistungsfähigkeit und Skalierbarkeit eines Dienstes möglich [EPM13, S. 403-412]. Für Nutzer und Betreiber stellt das SLA ein wichtiges Instrument dar, um auch den Quality of Service (QoS) zu verbessern.

Definition 2.5: Service Level Agreement (SLA) *A service level agreement (SLA) is a contract between a service provider (either internal or external) and the end user that defines the level of service expected from the service provider. SLAs are output-based in that their purpose is specifically to define what the customer will receive. SLAs do not define how the service itself is provided or delivered²⁸. [Pal16]*

Ein SLA von 1,0 entspricht der vollständigen Erfüllung der Vereinbarungen, wie beispielsweise die Bereitstellung ausreichender Ressourcen wie in Abbildung 2.17(a). Ist das SLA hingegen < 1,0, bedeutet das einen Einbruch in der vereinbarten Leistungsfähigkeit des Systems, wie in Abbildung 2.17(b) zu sehen.

2.3.3 Herausforderungen im Cloud-Computing

In [Arm+10] erläutern Armbruster et al. zehn Punkte und aktuelle Herausforderungen, die sich im Bereich des Cloud-Computings ergeben. Das bereits in den Abschnitten 2.3.2.1 und 2.3.2.2 angesprochene Problem der optimalen Lastverteilung und Ablaufplanung ist dort ebenso aufgelistet wie eine schnelle Skalierung zur Gewährleistung der Elastizität und das Problem der Sicherheit des Cloud-Computing, welches von der Zugangssicherheit über die Datensicherheit bis hin zur Ausfallsicherheit reicht [Fen+11].

Puthal et al. [Put+15] sehen ebenfalls die Lastverteilung und insbesondere die Eignung von Clouds für wissenschaftliche Anwendungen und den HPC-Bereich als Herausforderung an. Nach Zhang et al. [ZCB10] bestehen weitere Herausforderungen in der dynamischen Migration von virtuellen Maschinen und verteilten Anwendungen mittels MapReduce [DG08], um das Cloud-Computing für den HPC-Bereich attraktiver zu machen [VPB09]. Zwei weitere entscheidende Punkte sind nach [Arm+10] und [Put+15] der Energieverbrauch der Cloud oder die Sicherheit von Nutzerdaten.

²⁸Freie Übersetzung von [Pal16]: Eine Service-Level-Vereinbarung (SLA) ist ein Vertrag zwischen einem Service-Provider (entweder intern oder extern) und dem Endverbraucher, welcher das Niveau des Services, welches vom Service-Provider erwartet wird, definiert. SLAs sind eine Ausgangsbasis, um die Leistungen, welche der Kunde erhält, festzuschreiben. SLAs definieren nicht, wie der Dienst selbst zur Verfügung gestellt wird.

2.3.3.1 Sicherheitsaspekte im Cloud-Computing

Die Gewährleistung der Zugriffssicherheit von Daten in der Cloud bildet eine der größten Herausforderungen im Cloud-Computing, da die Daten der Nutzer auf einem fremden Rechnersystem gelagert werden, auf welches wiederum eine Vielzahl von unterschiedlichen Nutzern Zugriff haben [KV10]. Die Abschottung der VMs voneinander ist bereits weitestgehend durch die eingesetzten Techniken zur Virtualisierung von Rechen- und Speicherressourcen gewährleistet, die in Abschnitt 2.2.3 aufgezeigt wurden.

Eine Möglichkeit, die Daten in der Cloud noch weitreichender zu schützen, besteht auch in einer Verschlüsselung der Daten, wobei selbst dabei noch das Problem existiert, dass die Daten innerhalb der VM und im Arbeitsspeicher im Klartext vorliegen. Ein Ansatz, auch diese Daten zu schützen, besteht einerseits in der Anonymisierung der Daten durch Zusatzhardware, wie in der Arbeit von Mondol et al. [Mon11] diskutiert, und andererseits in der ausschließlichen Arbeit mit homomorphen Daten, wie es ursprünglich von Gentry et al. in [Gen+09] gezeigt und von Atayero et al. in [AF11] oder Tebaa et al. in [TEE12] für den Einsatz in der Cloud evaluiert wurde. Der Einsatz von FPGAs zur Erhöhung der Sicherheit von Nutzerdaten in der Cloud wird weiterführend in Abschnitt 2.4.3 untersucht.

2.3.3.2 Energieeffizienz von Cloud-Systemen

Nach Beloglazov et al. [BAB12] ist effizientes Energiemanagement heutzutage von zunehmend großer Bedeutung und bildet die Basis der von Baliga et al. [Bal+11] angestrebten *Green Cloud* mit einer Optimierung des Energieverbrauches über die gesamte Architektur. Nach einem Bericht des Lawrence Berkeley National Laboratory aus dem Jahr 2008 [Bro+08] wurden im Jahr 2006 1,5 % der in den USA erzeugten Energie für Rechenzentren benötigt und es wird davon ausgegangen, dass dieser Wert langfristig auf 18 % anwachsen wird. Ein großer Anteil dieser benötigten Energie ist jedoch der schlechten Auslastung der Systeme zuzuschreiben, von denen sich ein großer Anteil im Leerlauf befindet²⁹, sowie einer Ressourcenplanung, welche auf einer Maximierung des Durchsatzes ausgelegt ist, und nicht darauf, Energie zu sparen [Li+09].

Eine weitere Möglichkeit zur Steigerung der Energieeffizienz besteht in der Nutzung von speziellen Hardwarebeschleunigern in der Cloud. Dabei werden rechenintensive Anwendungen zum Beispiel auf energieeffiziente rekonfigurierbare Hardware [Bym+14; Che+14; FVS15; MS11; OCC16; Wee+15] oder GPGPUs [Nap+14; Rav+11] ausgelagert. Abschnitt 2.4 dieser Arbeit setzt sich ausführlich mit Beiträgen auf diesem Gebiet auseinander.

2.4 Relevante Forschungsarbeiten zu FPGAs und Cloud

Aufbauend auf den Grundlagen und derzeitigen Forschungsansätzen zu rekonfigurierbarer Hardware in Abschnitt 2.1, den grundlegenden Virtualisierungstechniken in Abschnitt 2.2 und dem Konzept des Cloud-Computing in Abschnitt 2.3 widmet sich dieser Abschnitt der zentralen Fragestellung einer möglichst flexiblen Einbettung von rekonfigurierbaren Hardwarebeschleunigern in Rechenzentren und Cloud-Systeme zur Erhöhung der Rechenleistung, Senkung des Energieverbrauches und der Steigerung der Sicherheit. Wie in Abschnitt 2.1.1.3 erläutert, stellen Hardwarebeschleuniger auf Basis von FPGAs eine vielversprechende Option dar, um sowohl die Rechenleistung als auch die Energieeffizienz zu erhöhen. Neben der Beschleunigung von Anwendungen sind FPGAs auch zur Steigerung der Sicherheit einsetzbar.

²⁹Im Leerlauf benötigen Server noch 69-70 % der Energie, welche typischerweise unter Vollast benötigt wird [Li+09].

Daraus ergibt sich eine Vielzahl von Möglichkeiten, wie FPGAs in eine Cloud integriert werden können, wie Abbildung 2.18 veranschaulicht. Dabei liegt der Schwerpunkt der Arbeit und insbesondere dieses Abschnitts auf der Integration der FPGAs auf der Anwendungsebene der Cloud.

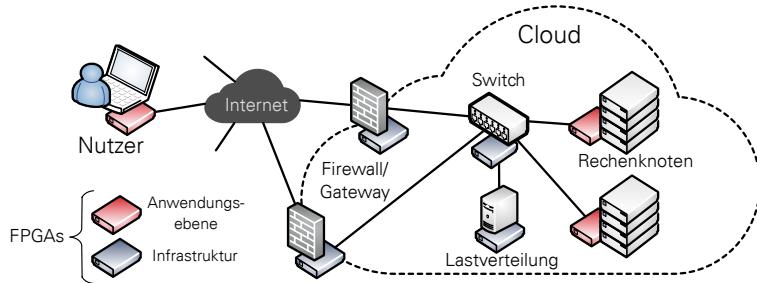


Abbildung 2.18: Cloud-Architektur mit Einsatzgebieten von FPGAs.

Im Folgenden werden in Abschnitt 2.4.1 Ansätze zur Virtualisierung rekonfigurierbarer Hardware ohne speziellen Cloud-Bezug diskutiert. In Abschnitt 2.4.2 werden Arbeiten vorgestellt, welche sich der Beschleunigung von konkreten Cloud-Anwendungen mit Hilfe rekonfigurierbarer Hardware widmen. Um einen möglichst universellen Einsatz von FPGAs zur Steigerung der Sicherheit von Daten innerhalb der Cloud und beim Zugang zu dieser zu untersuchen, widmet sich Abschnitt 2.4.3 den aktuellen Forschungsthemen in diesem Bereich. Darauf aufbauend werden in Abschnitt 2.4.4 aktuelle Arbeiten und Forschungsprojekte mit dem Ziel der Bereitstellung von FPGAs in einer Cloud detailliert erläutert.

2.4.1 Virtualisierung von Hardwarebeschleunigern auf Basis von FPGAs

Der Grundgedanke der Virtualisierung ist eine Abstraktion von der realen physischen Hardware mit der Absicht, eine größere Unabhängigkeit von dieser zu erreichen (siehe Abschnitt 2.2.1). Das Ziel bei der Virtualisierung von Hardwarebeschleunigern kann also in der Abstraktion von der physischen Struktur, aber auch in einer Abkapselung zur Erhöhung der Auslastung bestehen. Im Folgenden wird primär die Virtualisierung von rekonfigurierbarer Hardware als wesentlicher Schwerpunkt der Arbeit betrachtet.

2.4.1.1 Überblick zu relevanten Forschungsarbeiten

Erste Gedanken zur Notwendigkeit einer Virtualisierung von FPGAs veröffentlichten Fornaciari et al. im Jahr 1998 [FP98a; FP98b]. Hierbei ist die wesentliche Aussage, dass aufgrund der geringen Größe der verfügbaren FPGAs diese mit Hilfe von Methoden aus dem Bereich der Betriebssysteme virtuell vergrößert werden können, indem die Funktionen zur Laufzeit ausgetauscht werden (Dynamic Loading). Die Arbeit stellt allerdings keine konkreten Konzepte oder Ergebnisse vor. Eine Umsetzung der Konzepte *Partitionierung*³⁰ und *Überlagern* von Funktionsblöcken zur Bereitstellung eines virtuellen FPGAs, der größer als der tatsächliche physische FPGA erscheint, wird später in [FPR00] aufgezeigt.

Erst durch die Bereitstellung einer dynamischen partiellen Rekonfiguration von FPGAs zur Laufzeit (siehe Abschnitt 2.1.4.3) wurden die von Fornaciari et al. beschriebenen Ansätze für eine größere Nutzergruppe praktisch zugänglich. Darauf aufbauend beschreiben im Jahr 2008 El-Araby et al. in [EGE08] praktische Ansätze der Virtualisierung von FPGAs als Hardwarebeschleuniger im Bereich des High Performance

³⁰Einteilung des Hardware-Entwurfes in einen statischen (Module, welche immer benötigt werden, wie Infrastruktur oder spezielle Verarbeitungseinheiten) und einen dynamischen Teil (Komponenten, welche nicht durchgehend benötigt werden und dynamisch ausgetauscht werden können). Des Weiteren ist eine Aufteilung des prinzipiell zu großen dynamischen Hardware-Designs in kleinere Module, beziehungsweise Partitionen möglich, welche sequenziell abgearbeitet werden können. [FPR00]

Reconfigurable Computing (HPRC). Die Möglichkeit, FPGAs dynamisch partiell neu zu konfigurieren und der Versuch, durch statische Schnittstellen die Portabilität von Hardwaredesigns zu erhöhen, führte in den folgenden Jahren zu unterschiedlichen Ansätzen der Virtualisierung von FPGAs. Eine Vielzahl von Beiträgen basiert auf FPGAs von Xilinx und dem ICAP für den Zugriff auf die interne Konfiguration. Das Ziel, den physischen FPGA sowie die Schnittstelle zu diesem vor dem Nutzer zu verstecken und somit die Portabilität, Wiederverwendbarkeit und letztendlich Produktivität zu erhöhen, wird oftmals unter dem Begriff der Virtualisierung zusammengefasst.

Eine der einfachsten Möglichkeiten der *Virtualisierung* besteht in einer Abstraktion von den Hardware- und Softwareschnittstellen auf Seiten des FPGAs, wie sie bereits in Abschnitt 2.4.1.2 vorgestellt werden. Eine Kapselung unterschiedlicher Nutzer auf demselben physischen FPGA ist Gegenstand der in Abschnitt 2.4.1.3 vorgestellten Arbeiten. Hardwarebetriebssysteme, wie sie in Abschnitt 2.4.1.4 aufzeigt, stellen eine weitere Variante der Virtualisierung von FPGAs dar. Die in Abschnitt 2.4.1.5 vorgestellten Systeme etablieren eine Zwischenschicht zwischen dem Hardwaredesign und dem physischen FPGA, was einer Virtualisierung für unterschiedliche ISAs von Hostsystem und VM entspricht. Die relevanten Forschungsarbeiten werden schließlich in Abschnitt 2.4.1.6 miteinander verglichen.

2.4.1.2 Abstraktion von Schnittstellen und Treibern

Eine einfache Möglichkeit der Virtualisierung ist eine Vereinheitlichung beziehungsweise eine Abstraktion von den realen physischen Schnittstellen und die Schaffung eines einheitlichen Interfaces und Protokolls zur Erhöhung der Portabilität. Dabei ist die Ansatzperspektive entscheidend, da die Virtualisierung sowohl auf dem FPGA als auch auf dem Host, welcher den Zugang zum FPGA bereitstellt, ansetzen kann, wie im Folgenden weiter analysiert wird.

Portabilität durch einheitliche Schnittstellen – VirtualRC von Kirchgessner et al.

Ein Versuch, eben diese Portabilität zu erhöhen, stellt beispielsweise das Framework VirtualRC von Kirchgessner et al. [Kir+12] dar. Die Portabilität von HDL-Designs und HLS-Tools wird über unterschiedliche FPGA-Boards durch ein Framework und eine Middleware zur Virtualisierung von FPGAs erreicht. VirtualRC von Kirchgessner et al. [Kir+12] ist ein erweitertes Konzept der FPGA Middleware für ein Software Defined Radio (SDR) von Reves et al. [Rev+05]. Den prinzipiellen Aufbau von VirtualRC veranschaulicht Abbildung 2.19. Anhand von Konfigurationen wie der Anzahl von Speicherkanälen oder der Datenwortbreite generiert das Framework eine virtuelle Plattform in Form eines leeren HDL-Moduls mit entsprechenden Schnittstellen. Die Nutzeranwendung wird in das bereitgestellte HDL-Modul eingebettet. Im Anschluss generiert das Framework aus dem virtuellen Nutzerdesign und den HDL-Modulen der gewählten physischen Plattform die Konfigurationsdatei für den FPGA. Die Konvertierung der virtuellen Interfaces auf die physischen Interfaces einer spezifischen Plattform wird durch einfache Kontrolllogik, First In First Outs (FIFOs) und Speicher ermöglicht. Das Framework erzeugt hiermit im Wesentlichen eine generische Schnittstelle (Wrapper) von virtuellen zu physischen Interfaces.

Durch die Virtualisierung tritt bei ungünstigen Datenwortbreiten ein Protokolloverhead von bis zu 6 % auf. Da entsprechend wenig zusätzliche Logik von Seiten des Frameworks in das Design eingebracht wird, liegt der zusätzliche Ressourcenbedarf der Fläche bei nur 1 %. Da der Schwerpunkt dieser Arbeit auf der Virtualisierung der Schnittstellen liegt, sind keine Aussagen zu Konfigurationszeiten, konkurrierenden Nutzern und dem detaillierten Entwurfsprozess vorhanden. Die Möglichkeiten der partiellen Rekonfiguration bleiben ebenfalls unberücksichtigt, da der Nutzer immer eine vollständige Konfigurationsdatei für

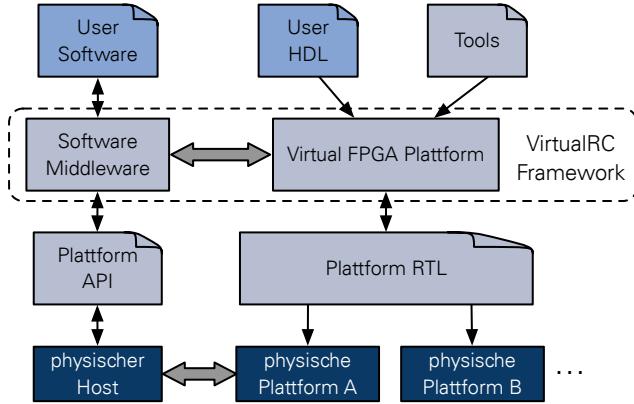


Abbildung 2.19: Überblick des VirtualRC Frameworks. Nach [Kir+12].

die physische Plattform erzeugt, wobei das VirtualRC-Framework die entsprechenden Schnittstellen und vordefinierten Module liefert.

Hardware als Software-Prozess – BORPH von So et al.

Ein ähnlicher Ansatz, allerdings mit einer höheren Abstraktion auf Prozessebene, wurde von So et al. [SB08] mit in BORPH³¹ eingeführt. BORPH erweitert das Betriebssystem (Linux) um Treiber und Bibliotheken, um *rekonfigurierbare Ressourcen* einfacher einbinden zu können. Für den Nutzer erfolgt Kommunikation und Interaktion mit einem *Hardware-Prozess* wie mit einem traditionellen *Software-Prozess* (UNIX-Prozess), jedoch wird der Hardware-Prozess auf einem FPGA ausgeführt [SB08]. Ein Hardware-Prozess hat dabei die gleichen Möglichkeiten wie ein Software-Prozess, beispielsweise die, einen Zugriff auf das Dateisystem zu realisieren. Abbildung 2.20 stellt in einer Übersicht die prinzipielle Systemarchitektur dar, wobei die Grenze zwischen Hard- und Software auf die Ebene des Betriebssystems (BORPH-Kernel) verlagert wurde. Der BORPH-Kernel ist ein modifiziertes Linux, welches auf einem PowerPC 405 ausgeführt wird [SB08].

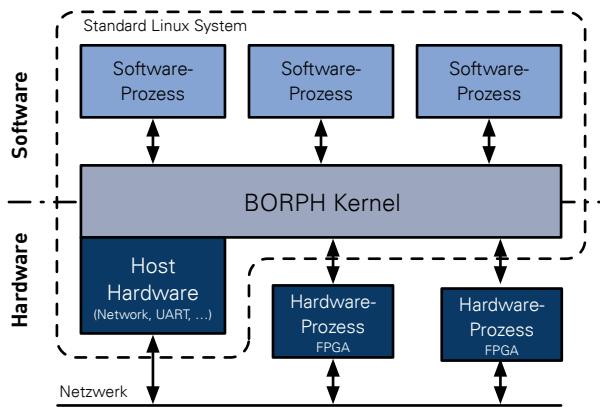


Abbildung 2.20: Überblick des Systemaufbaus von BORPH auf Prozessebene. Nach [SB08].

Wird ein Hardware-Prozess gestartet, erfolgt zunächst die Konfiguration des FPGAs. Speicherbereiche und Register des Hardwaredesigns werden über den Kernel im Betriebssystem eingeblendet, und ein

³¹Akronym für Berkeley Operating System for ReProgrammable Hardware (BORPH).

entsprechender Prozess zur Verwaltung wird angelegt. Die ausführbare Datei, welche einem Hardware-Prozess entspricht (BORPH Object File), enthält sämtliche Informationen, die zur Ausführung notwendig sind, wie die geeigneten FPGAs, den Bitstream und die Daten zur Einbettung der Register und Speicherbereiche. Die Kommunikation der Hardware-Prozesse untereinander wird über einen Kontroll-FPGA mittels konventioneller Kommunikation unter UNIX-Prozessen umgesetzt³². Dieses Vorgehen stellt eine Schwachstelle des Ansatzes dar, ist allerdings auch nicht das Hauptaugenmerk des Entwurfs.

BORPH entspricht somit einer Virtualisierung, bei der nicht nur die physischen Schnittstellen abstrahiert werden, sondern auch die Kommunikation im Betriebssystem für den Nutzer wie bei einem typischen UNIX-Prozess erscheint. Der Vorteil für spätere Entwicklungen besteht in der vollständigen Abstraktion von der realen Hardware durch Verlagerung der spezifischen Funktionalität in den BORPH-Kernel.

2.4.1.3 Virtualisierung zur Erhöhung der Anzahl an Ressourcen

Um begrenzte Hardwareressourcen unterschiedlichen Nutzern zur Verfügung zu stellen, besteht die Möglichkeit der Virtualisierung. Dabei wird bei jedem einzelnen Nutzer der Eindruck geschaffen, dass die Hardware jeweils ihm exklusiv zur Verfügung steht. Es existieren hierbei zwei grundlegende Möglichkeiten der Realisierung der Virtualisierung, ähnlich wie in der Klassifikation der Parallelität: eine sequenzielle Abarbeitung der Anfragen durch deren Serialisierung (Pipelining), oder eine Partitionierung in unabhängige Bereiche (Nebenläufigkeit). Eine dritte, ergänzende Möglichkeit kann die sequenzielle Abarbeitung der Anfragen mit zusätzlichem Kontextwechsel darstellen. Im Folgenden werden diese Möglichkeiten der Virtualisierung an unterschiedlichen Arbeiten erläutert.

Mehrere Nutzer durch Serialisierung der Anfragen von VMs – pvFPGA von Wang et al.

Eine weit verbreitete Variante zur Erhöhung der Anzahl der Hardwareressourcen durch Virtualisierung besteht darin, die Anfragen in eine Warteschlange einzureihen und nacheinander abzuarbeiten. Die Nutzer sind immer nur mit einem virtuellen FPGA konfrontiert. Das Problem dieses Ansatzes liegt in der hohen Bearbeitungszeit bei entsprechend langer Warteschlange, verursacht durch zu viele Nutzeranfragen. Das Pipelining ermöglicht die Nutzung des gesamten physischen FPGAs mit einem hohen Durchsatz in der Verarbeitung.

Das System pvFPGA von Wang et al. [WBP13] erlaubt einen konkurrierenden Zugriff unterschiedlicher paravirtualisierter Virtueller Maschinen auf einen gemeinsamen FPGA zur Rechenbeschleunigung. Ein Durchreichen des PCIe-Gerätes mittels *PCI-Passthrough* [Pra+05] ist üblicherweise nur an eine VM möglich und daher nicht für den Zugriff unterschiedlicher VMs geeignet³³. Abbildung 2.21 zeigt die Systemarchitektur. Dabei greifen sämtliche Nutzer auf denselben Rechenkern zu. Die Kommunikation mit dem Rechenkern erfolgt über eine einfache Schnittstelle.

Die Virtualisierung beschränkt sich auf den Zugriff von unterschiedlichen VM-Instanzen auf einen gemeinsam genutzten Beschleuniger. Im Gegensatz zu den bisher betrachteten Arbeiten erfolgt die Virtualisierung des FPGAs auf Seiten des Host-Systems und ermöglicht den gleichzeitigen Zugriff von unterschiedlichen VMs. auf den virtualisierten Beschleuniger. Der Zugang zum FPGA wird reproduziert beziehungsweise virtuell vervielfältigt. Ein Nutzerdesign auf dem FPGA und die Kapselung eines entsprechenden Bereiches mit partieller Rekonfiguration ist nicht vorgesehen.

³²Konventionelle UNIX-Mechanismen zur Kommunikation unter Prozessen (Inter-Process Communication (IPC)) sind zum Beispiel Pipelines, Signale, gemeinsame Dateien und Speicherblöcke [TB14, S. 100].

³³Mit einem PCIe 3.0-Endpunkt, welcher die I/O-Virtualisierung unterstützt (SR-IOV), erscheinen im PCI-Baum mehrere virtualisierte Geräte, welche an unterschiedliche VMs durchgereicht werden können [Sur13].

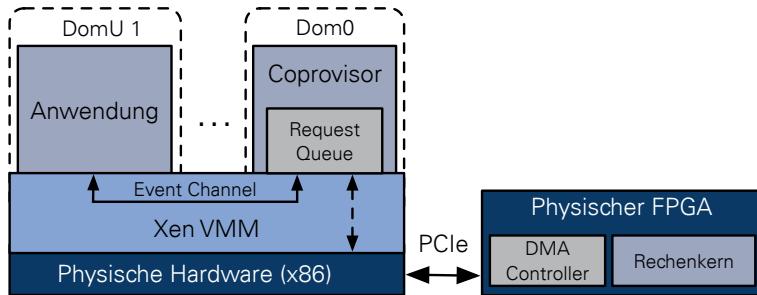


Abbildung 2.21: Überblick des pvFPGA Systems. Nach [WBP13].

Anwendungen aus unterprivilegierten VMs greifen bei pvFPGA [WBP13] über Event Channels in Xen [Pra+05] über den Hypervisor innerhalb der Dom0 zu. Der Hypervisor wird dabei durch einen sogenannten *Coprovisor* erweitert, welcher Anfragen der VMs in eine Warteschlange einreihet und nacheinander auf dem physischen FPGA ausführt [WBP13]. Durch die Beschränkung der Virtualisierung auf den Zugang und den modifizierten Treiber innerhalb der unterprivilegierten VM handelt es sich daher um eine klassische Paravirtualisierung des Treibers innerhalb des Host-Hypervisors, wie in Abschnitt 2.2.3.3 erläutert.

Ein ähnliches Konzept mit einer seriellen Abarbeitung paralleler Anfragen zur Beschleunigung von HPC-Anwendungen wird von Gonzalez et al. in [Gon+12] beschrieben. Hintergrund ist hier eine typischerweise deutlich geringere Zahl von physischen FPGAs als CPUs. Das Durchreichen an unterschiedliche VMs entfällt, dafür gibt es eine gemeinsame Warteschlange für die Anfragen. Wie auch in VirtualIRC lässt sich beobachten, dass sich das System mit zunehmender *Virtualisierung* entsprechend der Anzahl der Nutzer aus Sicht des jeweiligen einzelnen Nutzers verlangsamt.

Erhöhung der Anzahl der Ressourcen durch Partitionierung in Bereiche

Bedingt durch die wachsende Anzahl an Transistoren in integrierten Schaltkreisen (siehe Abschnitt 2.1.1.1) und den ebenso in der Zahl ihrer Logikelemente wachsenden FPGAs ist seit 2008 der Trend zu beobachten, dass nebenläufige Hardwaredesigns unterschiedlicher Nutzer auf demselben physischen FPGA zur gleichen Zeit ermöglicht werden, um dessen Auslastung zu erhöhen [Bym+14; Che+14; Don+15; EGE08; FVS15; Wee+15].

Ein frühes Beispiel eines FPGAs mit festen Regionen, welche zur Laufzeit dynamisch rekonfiguriert werden können – wenn auch ohne konkreten Cloud-Bezug – zeigen die Arbeiten von El-Araby et al. [EGE08; EGE09]. Bei dem System, welches aus zwei Mikroprozessoren (μ CPU) besteht, ist der physische FPGA mit einem der Prozessoren direkt verbunden³⁴. Durch die Virtualisierung wird somit die Anzahl der vFPGAs erhöht, um jedem Mikroprozessor exakt einen vFPGAs zuzuordnen. Das asymmetrische System wirkt für den Nutzer durch die Virtualisierung symmetrisch, wie in Abbildung 2.22(a) dargestellt. Dadurch gestaltet sich die Programmierung des Gesamtsystems deutlich einfacher. Die auf die vFPGAs ausgelagerten Berechnungen arbeiten nebenläufig, die Kommunikation erfolgt allerdings über einen gemeinsamen Kanal. Die Datenpakete werden in eine Warteschlange eingereiht, sodass die Datenübertragung zwischen Host und FPGA rein sequenziell erfolgt (siehe Abbildung 2.22(b)).

³⁴Die Cray XD1 [Cra16] besteht aus Knoten mit Mikroprozessoren und FPGAs als eng gekoppelte Coprozessoren. Die Hardware wird mittels Mitron-C programmiert, was eine einheitliche systemweite Integration und Programmierung erleichtert.

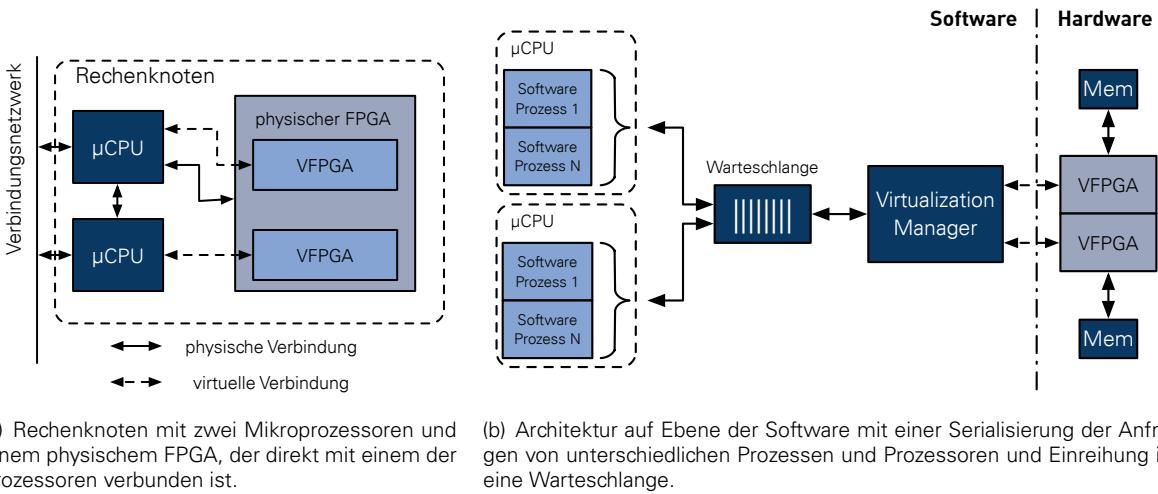


Abbildung 2.22: Virtualisierung des physischen FPGAs durch feste Einteilung in unterschiedliche Bereiche. Die Rechenkerne innerhalb der festgelegten Regionen auf dem physischen FPGA arbeiten nebenläufig. Nach [EGE08].

Die physischen Regionen für die vFPGAs werden mittels eines statischen Teils realisiert, der die Kommunikation und die dynamische partielle Rekonfiguration kapselt. Die vFPGAs selbst sind partiell zur Laufzeit über den ICAP rekonfigurierbar. Im Abschnitt 2.4.4 stellt diese Form der Virtualisierung die typische Methode dar, um die Ressourcen effizient an die Anforderungen der Nutzer anzupassen und die Auslastung der FPGAs zu optimieren.

Ein ähnliches Konzept nutzen auch Dondo Gazzano et al. [Don+15]. Die FPGAs sind allerdings über das Netzwerk lose untereinander sowie mit einem Rechner verbunden, welcher das Management übernimmt. Der FPGA selbst wird in mehrere Bereiche aufgeteilt, welche mittels partieller Rekonfiguration direkt über das Netzwerk konfiguriert werden können. Durch die ausschließliche Kommunikation über das Netzwerk ist der Zugang zu der Ressource von unterschiedlichen VMs deutlich einfacher. Eine Anwendung wird auf diese Weise in kleinere Rechenkerne zerlegt und auf mehrere FPGAs verteilt. Das so entstehende System erscheint dem Nutzer homogen und ist somit einfacher programmierbar. Auf dem System befindet sich allerdings immer nur ein Nutzer, wodurch die Virtualisierung ähnlich wie bei El-Araby et al. dazu dient, eine homogene Sicht von Außen auf Nutzerseite auf die Architektur zu schaffen.

Die Virtualisierung eines eng mit dem Host über PCIe gekoppelten FPGA und einer virtualisierten Schnittstelle für einen Einsatz in einer Cloud wird von Chen et al. in [Che+14] beschrieben. Der Zugriff auf den von unterschiedlichen VMs gemeinsam genutzten physischen FPGA, welcher in gekapselte Bereiche (vFPGAs) mit nebenläufigen Rechenkernen unterteilt ist, erfolgt über virtuelle I/O-Geräte im Hypervisor [Che+14]. Die Anfragen aus den VMs werden in Warteschlangen eingereiht und innerhalb der vFPGAs entsprechend ihrer Prioritäten abgearbeitet. Die Besonderheit in der Arbeit von Chen et al. besteht darin, dass der Speicher eines vFPGAs komplett ausgelesen, gesichert und wieder hergestellt werden kann, was einen kompletten Kontextwechsel ermöglicht, wenn sich in der Warteschlange eine Anfrage mit höherer Priorität befindet. Das Auslesen von internen Registern oder Speichern des FPGAs ist allerdings nicht möglich.

Das Problem der Kommunikation zwischen einer Vielzahl von Nutzerkernen auf demselben physischen FPGA wird in der Arbeit von Kidane et al. [KB16] mit Hilfe eines Network-on-Chip (NoC) gelöst. Die

Kommunikation der Kerne untereinander wird dabei als einer der Hauptkritikpunkte an anderen Arbeiten mit einer Aufteilung eines FPGAs in feste Regionen beschrieben.

2.4.1.4 Hardware-Betriebssysteme mit Threadwechsel

Mehrere Forschungsprojekte beschäftigen sich mit *Betriebssystemen für rekonfigurierbare Hardware*, welche einen wichtigen Schritt darstellen, um einerseits eine Kapselung der Funktionen und andererseits eine Verwaltung der Hardware, analog zu klassischen Betriebssystemen, zu ermöglichen. Die grundlegende Herangehensweise besteht darin, ähnlich wie bei BORPH (siehe Abschnitt 2.4.1.2) einen Rechenkern auf der rekonfigurierbaren Hardware durch einen Softwareprozess oder einen Software-thread zu repräsentieren und somit auch die Schnittstellen zu virtualisieren.

Ein solches System mit der Möglichkeit, mehrere konkurrierende Hardwarekerne auf dem FPGA in rekonfigurierbaren Bereichen abzukapseln, ist beispielsweise ReconOS von Lüppers et al. [LP07; Lüb10]. Das gesamte ReconOS-System-on-Chip (SoC), bestehend aus Prozessor, Speichersubsystem, Peripherie, ICAP zur Rekonfiguration und den rekonfigurierbaren Bereichen fester Größe, ist in Abbildung 2.23 dargestellt. Die Softwarethreads kommunizieren mit den Hardwarebeschleunigern in den rekonfigurierbaren Bereichen durch FIFOs (**O**perating **S**ystem **I**nterface (OSIF)) und haben im Gegensatz zu BORPH eine deutlich komplexere Zustandsverwaltung.

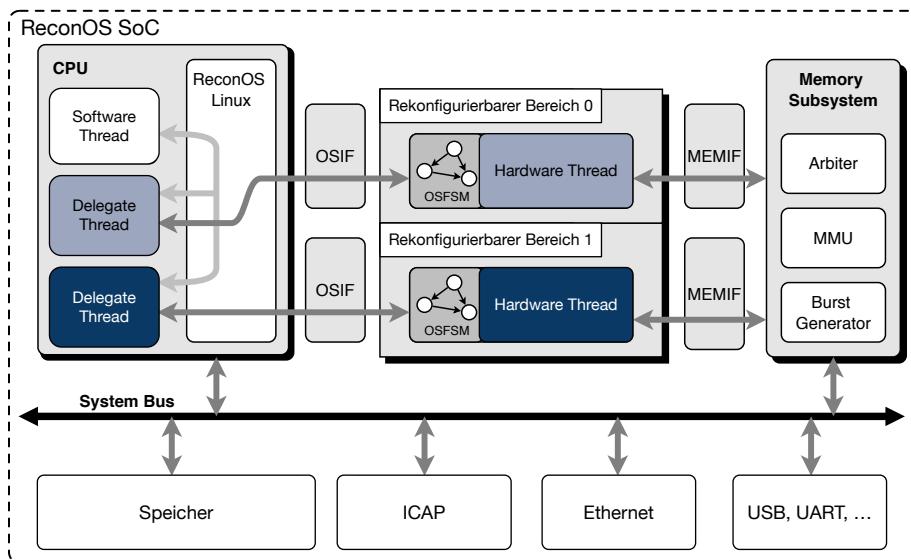


Abbildung 2.23: Aufbau des Hardware Betriebssystems ReconOS. Aus [Agn+14].

Um des Weiteren wie in Betriebssystemen einen Threadwechsel durchführen zu können, ist es notwendig, den kompletten Zustand beziehungsweise Kontext eines Hardwarethreads auszulesen und zu einem späteren Zeitpunkt fortzusetzen, wie in [HTK15] beschrieben (Preemptive Hardware Multitasking). Die Zeit, welche für einen Threadwechsel beziehungsweise für das komplette Auswechseln eines Hardwaredesigns benötigt wird, liegt bei mehreren Millisekunden³⁵, die Ausführung dieses Prozesses erfolgt über den internen ICAP durch Zugriff auf die Register und Speicherinhalte. Das Wiederherstellen eines Hardware-Threads ist allerdings nur in demselben Bereich auf dem FPGA, an der er ausgelesen wurde, möglich. ReconOS ermöglicht es somit, durch eng gekoppelte rekonfigurerbare Bereiche Anwendungsteile zur Beschleunigung in Hardware auszulagern. Hardwarethreads werden dabei in HDL oder durch

³⁵Für einen Bereich mit einer Konfigurationsgröße von 741 kByte werden für das Auslesen und das Wiederherstellen des kompletten Kontextes 25,7 ms benötigt.

HLS beschrieben. Die Kopplung von Prozessor und FPGA über PCI oder Ethernet wird in dem Projekt nicht berücksichtigt.

Verwandte Ansätze stellen HThreads von Peck et al. [Pec+06] oder Self Distributing Virtual Machine (SDVM) von Haase et al. [HHW10], welches entworfen wurde, um parallele und verteilte Berechnungen in Clustern durch das dynamische Austauschen von Rechenressourcen zu optimieren, dar. Ähnlich konzipiert, allerdings nicht auf Prozess-, sondern auf Betriebssystemebene, ist das PRHS-System [Eck14; Mey15; MK11], bei dem Mechanismen der System-Virtualisierung (siehe Abschnitt 2.2.3) untersucht werden, um statische Host-Komponenten (Prozessor, Speicher etc.) mit rekonfigurerbaren Gast-Bereichen zu kombinieren.

2.4.1.5 Zwischenschicht zur Abstraktion von der realen FPGA-Architektur

Um Hardwaredesigns von der physischen FPGA-Architektur zu entkoppeln und so die Flexibilität und die Wiederverwendbarkeit der Entwürfe zu erhöhen, wurden *Overlays*-Architekturen entwickelt. Deren Ziel besteht darin, auf unterschiedlichen FPGAs eine einheitliche Zwischenschicht zu erschaffen, auf welcher wiederum das eigentliche Design implementiert wird. Koch et al. beschäftigen sich in ihrer Arbeit [KBL13] mit einem Overlay für unterschiedliche FPGA-Familien von Xilinx. Basis ist die partielle dynamische Rekonfiguration und die Möglichkeit, in die Konfiguration nach den unterschiedlichen Syntheseschritten eingreifen zu können, wie in [BKT11] beschrieben.

Ähnliche virtuelle FPGA-Architekturen sind der virtuelle FPGA für schnelles Placement und Routing von Coole et al. in [CS10] oder die Realisierung neuartiger FPGA-Architekturen auf einem FPGA wie ZUMA von Brant et al. [BL12]. Durch die Erzeugung einer Zwischenschicht gehen die für diese aufgewendeten Ressourcen verloren, und die Leistungsfähigkeit eines Hardwaredesigns, insbesondere bezüglich der Signallaufzeiten (Timings) und erreichbaren Taktraten, wird geringer [BL12]. In der Arbeit von Figuli et al. [Fig+11] werden neben einer zugrundeliegenden virtualisierten Architektur Möglichkeiten beschrieben, die virtuellen FPGAs über mehrere physische FPGAs auszudehnen, ohne dass die realen Grenzen der Hardware für den Nutzer sichtbar sind. Die Möglichkeiten Entwürfe automatisiert auf mehrere FPGAs zu übertragen, deren Architektur mittels Overlays virtualisiert ist, ist dabei nicht das Ziel dieser Arbeit.

2.4.1.6 Vergleich der Virtualisierungen

Die vorgestellten Arbeiten zur Virtualisierung reichen von einheitlichen Interfaces zur Kopplung von VM und vFPGA, wobei die Ausführung auf dem physischen FPGA mittels Pipelining oder nebenläufig erfolgen kann, bis hin zur Kapselung in Hardwareprozesse und der entsprechenden Integration in ein Betriebssystem. In Tabelle 2.5 wird die Auswahl unterschiedlicher Arbeiten zu aktuellen Forschungsansätzen im Bereich der Virtualisierung rekonfigurerbarer Hardware zusammengefasst.

Im Hinblick auf eine spätere Integration des FPGA als Hardwarebeschleuniger, welcher dynamisch an eine VM gebunden werden kann, sind hierbei die Virtualisierung des Hosts (VMs), die Abstraktion von der physischen Schnittstelle zum FPGA (I/O) und die Virtualisierung des eigentlichen FPGAs mit der Möglichkeit, mehrere Nutzer oder Rechenkerne in Bereiche zu kapseln, Kriterien von besonderer Bedeutung, die in der Spalte *Virtualisierung* in Tabelle 2.5 gekennzeichnet werden. Die genutzten Techniken, wie partielle Rekonfiguration und darauf aufbauende Besonderheiten, wie das Auslesen und Verschieben eines Kontextes, sind in der Spalte *Merkmale* zusammengefasst. Die relevanten Merkmale basieren dabei auf der partiellen Rekonfiguration von (nebenläufigen) Nutzerdesigns (Partial Reconfiguration (PR)) mit Hilfe des ICAP, der darauf aufbauenden Möglichkeit, ein Nutzerdesign in eine andere Region zu verschie-

ben (Design Relocation (DR)) sowie dem Auslesen der aktuellen Register- und Speicherinhalte auf dem FPGA (Hardware Preemption (HP)). Eine Nutzung der Begriffe oder eine Einteilung nach den Kriterien der klassischen Virtualisierung, wie sie in Abschnitt 2.2 vorgestellt wurde, ist aufgrund der im Kontext von FPGAs eher freien Definition des Begriffes häufig nicht möglich. Die vorgestellten Arbeiten lassen sich in vier Gruppen einteilen, welche sich bezüglich der jeweils eingesetzten Technik unterscheiden:

I. Abstraktion von physischen Schnittstellen und Treibern: Die einfachste Abstraktion von den realen Schnittstellen erfolgt durch das jeweils feste Interface sowohl auf Seiten des FPGAs als auch auf dem Host-System, wie bei VirtualRC [Kir+12]. Eine vollständige Virtualisierung von Host, FPGA und Schnittstellen wird in der Arbeit von Chen et al. [Che+14] beschrieben. Der Zugang zu der gemeinsamen Ressource von unterschiedlichen VMs aus wird durch eine Treibervirtualisierung, ähnlich der von pvFPGA [WBP13] oder Nasiri et al. [NG16], realisiert. Zusätzlich ist der FPGA unterteilt in gekapselte Regionen (vFPGAs), welche dynamisch rekonfiguriert werden können. Die Kommunikation mit den Rechenkernen kann nahezu interaktiv erfolgen. Der Zugriff auf die gemeinsamen Ressourcen erfolgt bei mehreren Arbeiten des Weiteren über das Netzwerk [Bym+14; Don+15; Wee+15], was die Schwierigkeit einer Treibervirtualisierung durch die lose Kopplung von vFPGA und VM auf dem Host deutlich vereinfacht.

II. Erhöhung der Anzahl der Ressourcen durch Virtualisierung: Die gängige Form einer Virtualisierung, welche insbesondere bei Arbeiten mit Cloud-Kontext eingesetzt wird, ist die Aufteilung eines physischen FPGAs (und des externen DDR-Speichers) in Regionen fester Größe, welche der Nutzer mit seinem eigenen Hardwaredesign konfigurieren kann [Bym+14; Don+15; FVS15; Wee+15]. Da die Regionen auf den FPGAs fest zugeordnet sind, ist die Dynamik an dieser Stelle gering. Problematisch ist des Weiteren die Reduzierung der Berechnungen auf den FPGAs auf Arbeitspakete mit festem Kommunikationsablauf, was die nebenläufigen Zugriffe unterschiedlicher Nutzer vereinfachte: Die Pakete werden zur Abarbeitung in Warteschlangen eingereiht, wodurch sich für einen Cloud-Einsatz die möglichen Anwendungen aufgrund der fehlenden Interaktion verringern, wie in den Arbeiten von El-Araby et al. [EGE08] und Dondo Gazzano et al. [Don+15] gezeigt. Die Beiträge von Fahmy et al. und Asiatici et al. [Asi+16; FVS15], welche neben der festen Einteilung des FPGAs in Bereiche zusätzlich eine dynamische Partitionierung des externen DDR-Speichers bereitstellen, erreicht mit ihrer Art der Virtualisierung eine hohe Flexibilität für die unterschiedlichen Nutzer und deren Bedürfnisse.

III. Hardware-Betriebssysteme und rekonfigurierbare Prozessoren/Prozesse: Weitere wichtige Ansätze zur Virtualisierung stellen die Arbeiten von Chen et al. [Che+14] und Lübbbers et al. [Agn+14] dar, welche einen vollständigen Wechsel des Kontextes ermöglichen. Insbesondere der Kontextwechsel in ReconOS [Agn+14] ist für den Einsatz in einer Cloud geeignet, da hier auch auf interne Zustände des FPGA zugegriffen werden kann.

IV. Abstraktion von der physischen FPGA-Architektur: Die Nutzung von Overlays bietet den Vorteil, von der physischen FPGA-Familie oder dem Hersteller abstrahieren zu können, was die Flexibilität in einer Cloud-Umgebung erhöht, da selbst unterschiedliche FPGA-Architekturen im System homogen erscheinen und mit derselben Konfigurationsdatei konfiguriert werden können. Diese Art der Virtualisierung ist auch diejenige, welche einer klassischen Virtualisierung wie in Abschnitt 2.2 beschrieben am nächsten kommt. Der Ansatz ist aber aufgrund der Einbußen in der Leistungsfähigkeit auf dem FPGA nur eingeschränkt für die Hintergrundbeschleunigung eines Cloud-Dienstes anwendbar.

Tabelle 2.5: Überblick über relevante Forschungsarbeiten zur Virtualisierung von FPGAs mit Klassifizierung.

Arbeit/System	Virtualisierung			Typ/ Host I/O ^a	Merkmale	Nutzer- Anwendung	Kommunikation	Kurzbeschreibung			
	P	R	D	FPGA	PR	DR	HP	Bereiche	PCIe	Mem	Net
VirtualRC [Kir+12]	X	✓	X	Online Nutzer HDL	X	X	X	1	✓	✓	X
Borph [SB08]	X	✓	X	Online Nutzer HDL	X	X	X	1	✓ ^b	X	X
pvFPGA [WBP13]	✓	✓	X	Offline Feste Kerne	X	X	X	1	✓	X	X
El-Araby et al. [EGE08]	X	✓	✓	Offline ^c Nutzer HDL	✓	X	X	2	✓ ^b	X	X
Byma et al. [Bym+14]	X	✓	✓	Online Nutzer HDL	✓	✓	X	N	X	✓	✓
Weerasinghe et al. [Wee+15]	X	X	✓	Online Nutzer HDL	✓	X	X	2	X	✓	✓
Fahmy et al. [Asi+16; FVS15]	X	✓	✓	Online Nutzer HDL	✓	X	X	N	✓	✓ ^f	X
Dondo Gazzano et al. [Don+15]	X	X	✓	Offline Nutzer HDL	✓	X	X	N	X	✓	✓
Kidane et al. [KB16]	X	✓	✓	Online Nutzer HDL	✓	X	X	N	✓	X	X
Chen et al. [Che+14]	✓	✓	✓	Online Nutzer HDL	✓	X	! ^e	N	✓	X	!
ReconOS [Agn+14; HTK15]	X	✓	✓	Online Nutzer Threads	✓	!	✓	N	✓	✓	✓
PR-HMPSoC [NK14]	X	X	✓	Online Nutzer HDL	✓	!	?	N	X	✓	X
ZUMA [BL12]	X	X	✓	Online Nutzer DSL ^d	✓	X	X	1	X	X	X
Koch et al. [KBL13]	X	X	✓	Online Nutzer DSL ^d	✓	X	X	1	X	X	X

■: Abstraktion von physischen Schnittstellen und Treibern

■: Erhöhung der Ressourcenanzahl durch Virtualisierung

■: Hardware Betriebssysteme und rekonfigurierbare Prozessoren

■: Abstraktion von der physischen FPGA-Architektur

!: eingeschränkt

? : nicht bekannt

✓: vorhanden

X: nicht vorhanden

OV: Overlay

DR: Design Relocation

HP: Hardware Preemption

PR: Partielle Rekonfiguration

Mem: (DDR-)Memory

Net: Netzwerk

^a: Kommunikation mit FPGA von der physischen Schnittstelle abstrahiert.

^b: Protokoll/Schnittstelle ähnlich zu PCIe.

^c: Serialisierung der nebenläufigen Anfragen.

^d: Spezielle domainspezifische Sprache zur Beschreibung des Verhaltens.

^e: Das Auslesen des Kontextes beschränkt sich auf den externen DDR-Speicher.

^f: Externer DDR-Speicher über Seitentabellen virtualisiert.

2.4.2 Beschleunigung von Cloud-Diensten mit rekonfigurierbarer Hardware

Dieser Abschnitt zeigt die bisher erreichten Größenordnungen zur Steigerung der Verarbeitungsgeschwindigkeit und der Energieeinsparung bei der Integration von FPGAs in der Cloud auf und stellt konkrete Einsatzmöglichkeiten vor. Neben den typischen Anwendungen für FPGAs aus Abschnitt 2.1.3 sowie den typischen Cloud-Anwendungen, welche in Abschnitt 2.1.3 vorgestellt wurden, widmet sich dieser Abschnitt der Umsetzung typischer Cloud-Anwendungen auf FPGAs als Hardwarebeschleuniger.

Tabelle 2.6 zeigt eine Auswahl verschiedener Arbeiten, deren Ansätze sich ideal auf rekonfigurierbare Hardware auslagern lassen. Die Anwendungen werden eingeteilt in die Klassen Batch- sowie Stream-Verarbeitung. Die Tabelle zeigt des Weiteren Werte zu Speedup und Energieverbrauch, soweit diese verfügbar sind.

Tabelle 2.6: Rekonfigurierbare Beschleuniger für rechenintensive Cloud-Dienste in Anlehnung an [KS16]. Speedup und Energiersparnis im Vergleich zum Hostsystem.

System	Anwendung/ Technik	Typ	Speedup	Energie- Ersparnis	Integration/ Interface
		Batch	Stream		
Putnam et al. [Put+14] Microsoft Catapult	Suchmaschine Bing		✓	1,95 ×	– Coprozessor/ PCIe
Kachris et al. [Kac+16a]	Map Reduce	✓		4,3 ×	33 × Coprozessor/ AXI4
Blott et al. [Blo+15] Xilinx Research	Memcached		✓	1,35 ×	36 × Autonom/ Ethernet
Kocberber et al. [Koc+13] EcoCloud, HP, Google	Datenbank		✓	3.1 ×	3.7 × Coprozessor/ –
Knodel et al. [KLS16]	Black-Scholes	✓		28 ×	16 × Coprozessor/ PCIe
Choi et al. [CS14]	K-Means	✓		20 ×	– Coprozessor/ PCIe
Eguro et al. [EV12] Microsoft	Sicherheit Anonymisierung		✓	–	– Coprozessor/ PCIe
Gao et al. [GS16]	Sicherheit RSA		✓	–	– Coprozessor/ –

Ideal für FPGAs sind demnach streaming-basierte Anwendungen, wie Memcached von Blott et al. in [Blo+15], oder aus dem Bereich der Batch-Verarbeitung rechenintensive Anwendungen, wie beispielsweise RankBoost von Shan et al. [Sha+10] oder K-Means von Choi et al. [CS14]. Gerade für MapReduce-basierte Beschleuniger wurden in den letzten Jahren eine Reihe von Frameworks wie [Kac+16a; Y+12] entwickelt, um die Umsetzung von Anwendungen dieses Bereichs auf rekonfigurierbare Hardware zu erleichtern. Typische sicherheitsrelevante Anwendungen, welche auf Datenströmen arbeiten, wie der RSA-Algorithmus in der Arbeit von Gao et al. [GS16], wurden zusätzlich in die Tabelle aufgenommen und entsprechend ihrer Verarbeitungsart eingeordnet.

2.4.3 Einsatz von FPGAs zur Erhöhung der Sicherheit

Es existieren mehrere Möglichkeiten, FPGAs mit direktem Zugriff durch die Nutzer in die Cloud einzubringen. Neben dem in Abschnitt 2.1.3 angesprochenen Ansatz als Netzwerkrouter und Firewall sind

insbesondere die Absicherung des Zuganges zur Cloud oder der Einsatz als vertrauenswürdiger Coprozessor für Nutzer, neben der reinen Beschleunigung von Anwendungen, relevant.

2.4.3.1 FPGAs als Zugangshardware zur Cloud

Neben ihrer Funktion als Firewall werden FPGAs in einer Reihe von Arbeiten als kryptographischer Co-processor zur Steigerung der Sicherheit durch Verschlüsselung und Authentifizierung in Cloud-Architekturen eingesetzt. Aufgaben wie das Ver- und Entschlüsseln von Datenströmen sowie die Authentifizierung von Kommunikationspartnern in einer *End-to-End* Absicherung, beispielsweise mittels des Internet Protocol Security (IPSec) Protokolls, können auf FPGA-Hardwarebeschleunigern deutlich effizienter ausgeführt werden [NWZ13; Rao+16; Rou+04] als auf Prozessoren³⁶.

Die Nutzung des FPGAs als sicherer Coprozessor ergibt sich aus der Möglichkeit, die Konfiguration des FPGAs zu verschlüsseln, wie bereits in Abschnitt 2.1.4.4 vorgestellt. Durch die flexiblen Möglichkeiten zur Integration von FPGAs ist auch ein direkter Einsatz im Datenstrom zwischen Netzwerk und Host-System möglich, wie in der Arbeit von Caulfield et al. in [Cau+16] als Ausblick aufgezeigt wurde. Eine derartige Integration kann direkt die ein- und ausgehenden Daten der Cloud absichern, aber ebenso die Daten, welche in der Cloud innerhalb des Network Attached Storage (NAS) oder in einer Datenbank gespeichert werden. Eine andere Möglichkeit besteht im Einsatz eines FPGAs lokal beim Nutzer, um die ausgehenden Daten zu verschlüsseln oder zu anonymisieren [Mon11].

2.4.3.2 FPGAs als vertrauenswürdiger Coprozessor

Neben der einfachen Ver- und Entschlüsselungen des Zuganges zur Cloud ist für die Anwendungen bei nicht vertrauenswürdigen Cloud-Anbietern ebenso eine Anonymisierung von Nutzerdaten möglich, wie in der Arbeit von Eguro et al. [EV12] vorgestellt. Bei dessen Ansatz kann lediglich der FPGA innerhalb der Cloud die Klartextdaten einsehen, und sämtliche Anfragen über den eigentlichen Host erfolgen vollständig anonymisiert. Eine ähnliche Herangehensweise verfolgt die Arbeit von Arasu et al. [Ara+13], die sich mit einem sicheren Coprozessor für Datenbankanfragen auf Basis eines FPGAs beschäftigt. Bei beiden Arbeiten werden die privaten Schlüssel im verschlüsselten Bitstream des FPGAs eingebettet, sodass Angriffe innerhalb des Cloud-Systems nur unter Kenntnis der FPGA-Schlüssel möglich sind. Xu et al. analysieren in [XSS14], wie die klassische MapReduce-Pipeline auf einen FPGA ausgelagert werden kann. Dabei wird zu Beginn der Pipeline auf dem FPGA die Entschlüsselung und am Ende die Verschlüsselung der Daten durchgeführt. Der Ansatz sieht eine ausschließlich aus FPGAs bestehende Cloud ohne weitere Host-Systeme vor. Die Nutzung von sicheren Enklaven innerhalb eines Prozessors wird auch bei modernen Prozessoren angewandt, wie beispielsweise im Fall der Erweiterung Software Guard Extensions (SGX) von Intel [Int17c].

Trusted Computing auf Basis von FPGAs wird in der Arbeit von Eisenbarth et al. in [Eis+07] vorgestellt. Dabei wird eine Region auf dem FPGA für ein Trusted-Platform-Module (TPM)³⁷ sowie für eine Anwendung auf dem FPGA bereitgestellt. Hierbei werden die komplette Anwendung und die Daten vollständig verschlüsselt und sind lediglich für den Nutzer lesbar. Das System ist vollständig gegen Softwareangriffe abgesichert und lediglich für direkte Angriffe auf die spezielle Hardware anfällig, wie sie in [Eis+07]

³⁶Ver- und Entschlüsselung (AES-CBC-128-SHA1) eines Datenstroms von 40 GB/s lastet bis zu 15 Kerne eines modernen Serverprozessors (Intel Haswell) aus [GG13], kann aber effizient auf FPGA-Hardwarebeschleuniger ausgelagert werden [Cau+16].

³⁷TPM-Chips sind von einer Vielzahl von Herstellern verfügbar und die Implementierung innerhalb eines FPGAs benötigt relativ wenig Ressourcen [Eis+07; Gla+08; Tru16].

aufgezeigt werden. Die Erhöhung der Sicherheit durch Auslagerung des TPM auf einen externen Chip, ergänzend zum FPGA, wird von Glas et al. in [Gla+08] untersucht.

Ein weiterer wichtiger Schritt zur Erhöhung der Sicherheit in Cloud-Umgebungen besteht in der homomorphen Berechnung auf verschlüsselten Daten, wie von Atayero et al. in [AF11] oder Tebaa et al. in [TEE12] aufgezeigt wird (siehe Abschnitt 2.3.3.1). Das von Gentry et al. in [Gen+09] entwickelte Verfahren zur homomorphen Verschlüsselung ist für einen Cloud-Einsatz nur bei speziellen Anwendungen einsetzbar, da die Zahl der möglichen Operationen auf den Daten begrenzt ist [NLV11; VJ10]. Mit der Simple Encrypted Arithmetic Library (SEAL) [Mic16a; Mic16d] wurde 2015 eine Bibliothek entwickelt, welche eine Vielzahl von Rechenoperationen auf den verschlüsselten Daten ermöglicht. Die rechenintensive Ver- und Entschlüsselung kann dabei allerdings mit FPGAs deutlich beschleunigt werden, wie von Theoharoulis et al. in [The+11] bereits gezeigt wurde. Die Beschleunigung der Ausführung eines Programmes auf homomorph verschlüsselten Daten mit Hilfe eines FPGAs wurde von Pöppelmann et al. in [Pöp+15] und von Cao et al. in [Cao+14] vorgestellt. Beide Systeme erreichen eine signifikant höhere Verarbeitungsgeschwindigkeit gegenüber einer Ausführung in Software und zeigen, dass der Einsatz von FPGA-Coprozessoren für homomorphe Verschlüsselung eine effiziente Lösung darstellt.

2.4.4 Verwaltung und Bereitstellung von FPGAs in Rechenzentrum und Clouds

Aufbauend und ergänzend zu den in Abschnitt 2.4.1.3 vorgestellten Arbeiten zur Virtualisierung physischer Ressourcen mit dem Ziel, diese unterschiedlichen Nutzern parallel zur Verfügung zu stellen, werden in folgenden Arbeiten vorgestellt, welche sich mit der Bereitstellung und Verwaltung derartiger virtueller FPGA-Ressourcen in Cloud-Architekturen beschäftigen.

Typische heterogene Cloud-Architekturen verfolgen das Ziel, mit Hilfe von speziellen Hardwarebeschleunigern eine Steigerung der Verarbeitungsgeschwindigkeit rechenintensiver Anwendungen zu erzielen und dadurch das SLA zu erhöhen. GPGPUs werden bei kommerziellen Plattformen wie Amazon EC2 dem Nutzer in speziellen Konfigurationen als komplette physische Resource in eine VM durchgereicht. Diese *Accelerated Computing Instances* auf der PaaS-Ebene [Ama16a] werden dabei primär für datenintensive Berechnungen eingesetzt. Wissenschaftliche Beiträge von Crago et al. [Cra+11] und Ravi et al. [Rav+11] versuchen, die Nutzung heterogener Architekturen in OpenStack zu erreichen, wobei allerdings oftmals lediglich ein einfaches Durchreichen der Ressource an eine Virtuelle Maschine ohne Virtualisierung der Hardware erfolgt.

Eine flexiblere Alternative zu GPGPUs und vergleichbaren Beschleunigern mit fester Funktion oder Hardwarearchitektur stellen rekonfigurierbare Architekturen dar. Insbesondere in einem Rechenzentrum, welches flexible Cloud-Architekturen für unterschiedlichste Anwendungsbereiche bereitstellt, bietet die Möglichkeit einer Rekonfiguration der Hardware in Form von FPGAs enormes Potential in Bezug auf statische Hardware. Erste Ansätze zum Einsatz von FPGAs in der Cloud und der Frage nach den sogenannten *Hardware Clouds* veröffentlichten Madhavapeddy et al. in [MS11] im Jahr 2011. In dieser Arbeit werden drei wesentliche Herausforderungen für eine Integration von FPGAs in eine Cloud aufgezeigt. Insbesondere sicherheitsrelevante Aspekte und die Möglichkeiten der Anonymisierung von Nutzerdaten in Cloud-Umgebungen stellten ab 2010 die ersten Einsatzgebiete von FPGAs in der Cloud dar [Mon11]. In den folgenden Jahren wurden diese Aspekte zunehmend thematisiert, wie eine Reihe weiterer Veröffentlichungen in diesem Bereich wie [EV12; GS16; Pöp+15; XSS14] zeigen. Erst ab 2014 gewannen vermehrt Forschungsarbeiten zur Hintergrundbeschleunigung (siehe Abschnitt 2.4.2) und der Verwaltung der teilweise virtualisierten FPGA-Ressourcen an Bedeutung [Asi+16; Bym+14; Cau+16; Che+14; Cho16; Don+15; FVS15; Kac+16b; KLS16; OCC16; Wee+15].

Einen ausführlichen Überblick zu den aktuellen Entwicklungen und der wissenschaftlichen Relevanz von Forschungsarbeiten, welche sich mit der Integration rekonfigurierbarer Hardware in Cloud-Umgebungen auseinandersetzen, bietet die Arbeit von Kachris et al. [KS16]. Die grundlegende Motivation für die Integration von FPGAs in ein Rechenzentrum besteht darin, dass durch den Einsatz von FPGAs an beliebiger Stelle neue Rechenkerne und Architekturen bereitgestellt werden können, welche sich ideal für die wechselnden Problemstellungen der Nutzer eignen und somit die physische Hardware nicht mehr ausgetauscht werden muss. Aufbauend auf derartigen Ansätzen besteht das Ziel zahlreicher Forschungsarbeiten in diesem Feld darin, FPGAs als flexible und universelle Komponente dem Nutzer beziehungsweise Anbieter eines Dienstes bereitstellen zu können [Bym+14; Che+14; FVS15; OCC16; Wee+15].

2.4.4.1 Dedizierte FPGAs als Hardwarebeschleuniger in der Cloud – Amazon EC2 F1

Amazon hat sein Rechenzentrum ab Ende 2016 in einer Testphase mit FPGAs ausgestattet, die vollständig an die EC2-VMs durchgereicht werden können und als *Amazon EC2 F1-Instanzen* zur benutzerdefinierten Hardwarebeschleunigung für beliebige Anwendungen etabliert werden sollen [Ama17a]. Die F1-Instanzen enthalten ein spezielles FPGA-Entwickler Amazon Machine Image (AMI) und ein Hardware Developer Kit (HDK). Wenn das Hardwaredesign fertig gestellt ist, kann es als Amazon FPGA Image (AFI) registriert werden und in beliebigen F1-Instanzen bereitgestellt werden.

Die F1-Instanzen sind in zwei verschiedenen Größen als Vorversion verfügbar. Jede F1-Instanz enthält bis zu acht FPGAs, die der VM fest zugeordnet sind³⁸. Sie werden nicht von verschiedenen VMs oder Benutzern gemeinsam eingesetzt. Dadurch wird sichergestellt, dass die volle Leistung des FPGAs der Instanz jeweils fest zugeordnet ist und die Sicherheit durch die Isolation von Benutzern und Konten erhöht werden kann [Ama17a].

2.4.4.2 Flexible Einsatz von FPGAs in einem Rechenzentrum – Microsoft Catapult

Einen wichtigen Schritt zur Integration von FPGAs in ein produktives Rechenzentrum stellen Putnam et al. in [Put+14] mit dem System *Catapult* vor. Die ursprüngliche Motivation bestand in der Beschleunigung des Rankings mit Hilfe von FPGAs innerhalb des Web-Suchdienstes Bing von Microsoft. Dazu wurden PCIe-Einsteckkarten mit FPGAs bestückt und als Hardwarebeschleuniger für die Prozessoren innerhalb der bestehenden Server-Racks genutzt³⁹. Das grundlegende Hardwaredesign auf dem FPGA enthält einen statischen Teil, welcher die Module zur Kommunikation enthält. Die Anwendung wird hingegen in einem separaten Bereich (*Catapult Shell*) eingekapselt. Entsprechende Bibliotheken zur Kommunikation und zur einfachen Erweiterung des bestehenden MapReduce-Ansatzes werden ebenso bereitgestellt. Neben der engen Kopplung von Prozessor und FPGA über PCIe besteht des Weiteren zwischen den FPGAs untereinander ein 2D-Torusnetzwerk mit 48 FPGAs, um den Datenaustausch zu optimieren. Durch Catapult hat Microsoft den Datendurchsatz um 95 % im Vergleich zur Softwarelösung gesteigert und die Latenz um 29 % reduziert. Der Energieverbrauch der Server wurde durch die zusätzliche Hardware lediglich um 10 % erhöht.

Die Catapult-Architektur ist in Abbildung 2.24 aufgezeigt⁴⁰. Der FPGA leitet den Datenverkehr zur Netzwerkkarte (NIC) einer der beiden Prozessoren des Serverblades, hat aber auch die Möglichkeit, den Datenfluss zu modifizieren oder den Netzwerkanschluss für die eigene Kommunikation zu nutzen. Die

³⁸F1-Instanzen enthalten Einsteckkarten mit Xilinx UltraScale+ FPGAs sowie lokalen 64 GByte DDR4 Error Correction Code (ECC)-Speicher mit einer dedizierten PCIe x16-Verbindung [Ama17a].

³⁹In der Veröffentlichung von 2014 [Put+14] werden insgesamt 1.632 Server mit FPGA-Hardwarebeschleunigern ausgestattet.

⁴⁰Die Catapult-Architektur stellt seit 2016 die Basis von Microsofts neuen Computing-Servern im Rechenzentrum dar [Mic16b].

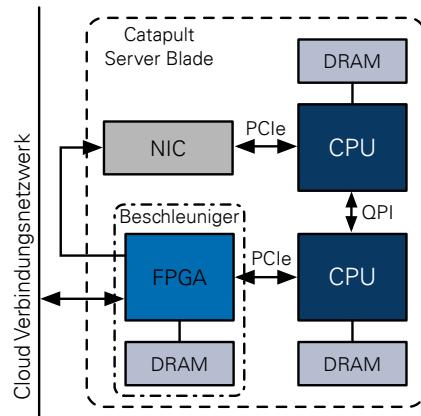


Abbildung 2.24: Hardwarearchitektur eines Knotens von Microsofts Catapult System. Nach [Cau+16].

Auslastungen der CPU- und FPGA-Ressourcen können somit unabhängig voneinander maximiert werden, da ungenutzte FPGA-Ressourcen losgelöst von der CPU genutzt werden können, und umgekehrt. Somit können sowohl reine FPGA-Cluster als auch hybride Anwendungen, welche einen eng mit einem Prozessor gekoppelten FPGA benötigen, realisiert werden. Die Architektur hat allerdings den Nachteil, dass der Netzwerkverkehr zum Prozessor immer über den FPGA geleitet werden muss.

Die Verwaltung der Ressourcen und deren Zuordnung zu einem konkreten Dienst erfolgt mit Hilfe einer zentralen Ressourcenverwaltung. Jeder Dienst verfügt über einen eigenen Knoten zum Management [Ovt+15]. Desgleichen hat jeder FPGA einen lokalen Manager, welcher die Anfragen des Diensts entgegennimmt. Durch diese Hierarchie besteht der Managementaufwand innerhalb der Cloud weitestgehend nur auf lokaler Ebene.

2.4.4.3 FPGAs als eigenständige virtualisierte Rechenressourcen im Rechenzentrum

Die Möglichkeiten und Vorteile, rekonfigurierbare Hardware direkt an die Bedürfnisse unterschiedlichster Nutzer anzupassen und somit eine völlig adaptive und dynamische Architektur eines Rechenzentrums bereitstellen zu können, welches ausschließlich aus FPGAs als Rechenressourcen besteht, werden in der Arbeit von Weerasinghe et al. [Wee+15] erläutert. Die FPGAs werden dem Nutzer dabei wie normale Rechenressourcen in einer Cloud zur Verfügung gestellt. In der angestrebten Architektur wird dabei ein physischer FPGA in mehrere virtuelle, partiell rekonfigurierbare Bereiche unterteilt, welche über das Netzwerk direkt angesprochen und rekonfiguriert werden können. Die FPGAs werden über eine Erweiterung des Cloud-Management-Systems OpenStack als eigenständige Beschleuniger, ähnlich wie Virtuelle Maschinen, verwaltet. Der angestrebte *Accelerator Service* arbeitet dabei direkt mit dem Netzwerkmanager zusammen, welcher das virtuelle Netzwerk für den Zugriff auf die FPGA-Ressourcen bereitstellt. In [Wee+16] wird das Konzept anhand von Messwerten mit klassischen Virtuellen Maschinen und Containern verglichen.

Ein ähnlicher Ansatz wird von Byma et al. in [Bym+14] vorgestellt. Der FPGA wird wie eine traditionelle virtualisierte Ressource behandelt, auf die über das Verbindungsnetzwerk zugegriffen werden kann, und in OpenStack verwaltet. Im Gegensatz zu der Arbeit von Weerasinghe et al. wird die Generierung einer partiellen Konfigurationsdatei für jede der möglichen vFPGA-Regionen beschrieben. Die Ressourcenverwaltung kann entsprechend der später allokierten Region die erforderliche Konfigurationsdatei auswählen, was die Maximierung der Auslastung der physischen FPGAs erleichtert. Der Host verwaltet

mehrere physische FPGAs und ist außerdem für Verwaltung und Rekonfiguration verantwortlich. Ein ähnliches Konzept nutzen auch Dondo Gazzano et al. [Don+15]. Die FPGAs sind in diesem Beitrag allerdings über das Netzwerk lose miteinander und zusätzlich mit einem Rechner verbunden, welcher das Management der Ressourcen mit Hilfe einer Datenbank verwaltet. Die Arbeit von Mishra et al. [MCZ16] widmet sich einem Protokoll zur partiellen Rekonfiguration von eigenständigen FPGA-Knoten.

2.4.4.4 Virtualisierte FPGA-Beschleuniger als eng gekoppelte Coprozessoren in der Cloud

Der grundlegende Gedanke dieses Ansatzes ist die Nutzung der FPGAs als Coprozessoren oder Hintergrundbeschleuniger möglichst verbunden mit der Auslagerung von Funktionsaufrufen eines Programms auf einen verfügbaren FPGA. In Vineyard von Kachris et al. [Kac+16b] werden dafür vorgefertigte Konfigurationsdateien genutzt. Die Auslagerung auf einen FPGA ist dabei dem Nutzer nicht bekannt.

Eine direkte Nutzung des FPGAs steht bei Chen et al. in [Che+14] im Vordergrund. Die FPGAs werden mittels PCIe oder CAPI eng mit einem Prozessor gekoppelt, welcher Arbeitspakete auf den Beschleuniger auslagert. Dabei wird jeweils einem Nutzer eine VM zugeordnet und bei Bedarf das entsprechende Arbeitspaket (*Hardware Modul*) auf einen verfügbaren Beschleuniger ausgelagert. Durch Kontextwechsel und Priorisierung ist sichergestellt, dass Nutzeranfragen zeitnah abgearbeitet werden. Die möglichen Hardwarekonfigurationen sind in einer Datenbank hinterlegt, eigene Designs müssen innerhalb des Systems generiert werden. Die Arbeit von Chen et al. [Che+14], ist ein wesentlicher Bestandteil von IBMs Cloud-Projekt SuperVessel [IBM15b], welches einen Teil der OpenPower Foundation [Ope16b] darstellt.

Eine eigene Ressourcenverwaltung für virtualisierte FPGA-Beschleuniger, welche nebenläufige Rechenkerne auf demselben physischen FPGA ermöglicht, ist in der Arbeit von Fahmy et al. [FVS15] beschrieben. Die Infrastruktur der Verwaltungs-Software ist unterteilt in FPGA Cloud-API (Middleware), Hypervisor und FPGA-Treiber. Das zentrale System des Ressourcenmanagements ordnet einer Nutzeranfrage den jeweils für die angegebene Konfigurationsdatei kleinstmöglichen vFPGA sowie einen Kommunikationsendpunkt in Form eines Threads innerhalb einer VM auf dem Knoten mit dem jeweiligen physischen FPGA zu. Der Hypervisor auf dem entsprechenden Knoten verwaltet die Zugriffe der Nutzer über einen entsprechenden Treiber. Das Nutzerprogramm, welches auf dem externen System des Nutzers ausgeführt wird, greift über das Netzwerk auf den Thread zur Kommunikation mit dem FPGA zu und kann auf diese Weise Daten senden und empfangen oder die Konfiguration ändern.

Die Arbeit von Kidane et al. [KB16] stellt die Kommunikation zwischen vFPGAs in den Vordergrund und optimiert diese mit einem NoC. Der bereitgestellte Dienst bietet neben der Beschleunigung mit vorgegebenen Kernen (Hardware Accelerator as a Service (HAaaS)) auch eine Nutzung eigener Rechenkerne (Reconfigurable Region as a Service (RRaaS)). Die Verwaltung der Ressourcen auf den Knoten wird ähnlich wie in der Arbeit von Fahmy et al. [FVS15] mit einem speziellen Hypervisor ermöglicht.

2.4.4.5 Scheduling von FPGA-Ressourcen in der Cloud

Die Verwaltung heterogener Ressourcen für die Hintergrundbeschleunigung von speziellen Anwendungen bildet einen Schwerpunkt der Arbeiten von Proaño et al. [OCC16; PCC14; PCC16]. Ein Dienst kann dabei entweder komplett in Software oder alternativ auf einem Hardwarebeschleuniger ausgeführt werden, wobei beide Implementierungen in einer Datenbank hinterlegt sein müssen. FPGAs oder GPGPUs werden direkt an die VMs durchgereicht, ohne diese zu virtualisieren. Die Abläufe werden in [PCC14] über die unterschiedlichen Komponenten wie Datenbank, Bitstream, Infrastruktur und Job Controller erläutert. In [OCC16] wird die Verwaltung und insbesondere das Scheduling für das gesamte System

2 Stand der Forschung und Technik

in der Management-, Knoten- und Hypervisor-Ebene aufgezeigt. Ein größeres Paket von Aufgaben wird dabei einem Knoten zugewiesen, welcher wiederum die konkreten Aufgaben an VMs weiterreicht. Um ein entsprechendes SLA einzuhalten, wird ein verfügbarer FPGA des Knotens zur Beschleunigung an die VM durchgereicht.

Ein Scheduling für eine elastische Cloud mit rekonfigurierbaren Ressourcen wird in der Arbeit von Grigoras et al. in [Gri+14] vorgestellt. Das Problem, einen Knoten aufgrund eines lang laufenden Arbeitspaketes, welches diesen nicht vollständig auslastet, aber trotzdem dessen Freigabe verhindert, wurde hierbei durch eine Aufteilung in kleinere Pakete und Ausnutzung der Nebenläufigkeit gelöst. Es wird darauf verwiesen, dass dieses Vorgehen nicht bei jedem Arbeitspaket möglich ist und somit dennoch die Notwendigkeit einer Virtualisierung mit Kontextmigration eines FPGAs besteht. Ebenso stellen Poglia- ni et al. [Pog+16] eine Ressourcenverwaltung mit Scheduling vor, welche dynamisch die Zahl der FPGAs erhöht oder verringert, um systemweit einen vordefinierten QoS zu erzielen. In der Arbeit von Iordache et al. [Ior+16] werden eingehende Arbeitspakete auf Gruppen bestehend aus mehreren physischen FPGAs verteilt, was einer horizontalen Skalierung entspricht.

2.4.4.6 FPGA Cluster und Remote Labs

Die Arbeiten von [Mey12; Sas07; Sch12; Tso10; Waw+07; Yos+10] integrieren FPGAs in Rechencluster, um neue Architekturen oder Protokolle zur Kommunikation oder Skalierbarkeit zu untersuchen. Andere Ansätze entwickeln ein massiv paralleles System für spezielle rechenintensive Problemstellungen [Bak10; Bax07; Men09; Sho09; Tso10]. Aufgrund der fehlenden Virtualisierung und des Fokus auf die Bereitstellung der Ressourcen werden diese Ansätze hier nicht im Detail betrachtet. Des Weiteren existieren Systeme mit stärkerem Augenmerk auf der Bereitstellung vollständiger FPGAs zur Entwicklung [Plu16] und ähnliche Systeme für den Einsatz in der Lehre (*Remote Labs*) [DS07; MN06; Raj+08].

2.4.4.7 Vergleich der Cloud-Integrationen und offene Fragestellungen

Die vorgestellten Arbeiten in diesem Abschnitt beschäftigen sich primär mit der Integration von FPGAs in Cloud-Architekturen und weisen – ebenso wie die Arbeiten zur Virtualisierung – abgesehen davon unterschiedliche Schwerpunkte auf. Da die Virtualisierung für Cloud-Ressourcen eine entscheidende Grundlage darstellt, sind Überlappungen mit den in Abschnitt 2.4.1 diskutierten Arbeiten vorhanden. Tabelle 2.7 gibt einen Überblick über eine Auswahl der zuvor vorgestellten Systeme zur Integration von FPGAs in eine Cloud, sowie über die sicherheitsrelevanten Einsatzmöglichkeiten und ordnet diese entsprechend in die drei Gruppen:

I. FPGAs als Rechenressource in der Cloud: Die Verwaltung von Hardwarebeschleunigern wie FPGAs in einer Cloud-Umgebung wird in einer Vielzahl der vorgestellten Arbeiten entweder vollständig an VMs durchgereicht, wie bei Amazon EC2 F1 [Ama17a], oder in Form von Arbeitspaketen an die gemeinsam genutzte Hardware ausgelagert. Oftmals wird zum einfacheren Zugang von unterschiedlichen VMs auf einen gemeinsamen FPGA (siehe auch Abschnitt 2.4.1) das Netzwerk wie in Byma et al. [Bym+14] oder Weerasinghe et al. [Wee+15] genutzt. Die einzige Integration mit flexiblem Zugang zur Ressource FPGA ist in [Cau+16] von Caulfield et al. beschrieben. Der FPGA kann hierbei entweder als dedizierter Hardwarebeschleuniger oder als rekonfigurierbares Netzwerkinterface eingesetzt werden, um sicherheitsrelevante Anwendungen zu ermöglichen.

Tabelle 2.7: Überblick relevanter Forschungsarbeiten zur Bereitstellung und Verwaltung von FPGAs in der Cloud.

Arbeit/System	Cloud Verwaltung	CPU	Hardware Dienste	Zugang	Virtualisierung	Kurzbeschreibung	
			CPU	FPGA	Host	FPGA	
Microsoft Catapult [Cau+16; Put+14]	?	✓	✓	HaaS	G-N-H-F G-N-F-H F-N-F	✗ !	Flexibler Zugang zu und über FPGAs in einer heterogenen HaaS-Cloud.
IMB SuperVessel Chen et al. [Che+14; IBM15b]	✓ ^a	✓	✓	AaaS	G-N-vF	✓ ✓	Auslagerung von Arbeitspaketen aus einer VM auf einen über PCIe oder CAPI gekoppelten vFPGA.
Amazon EC2 F1 [Ama17a] Nimbix [Nim16]	✓ ^a	✓	✓	AaaS	G-N-vH-F	✓ ✗	Zugang zu VMs mit vollständigen über PCIe durchgereichten FPGAs zur Beschleunigung von Anwendungen.
Byma et al. [Bym+14]	✓ ^a	✓ ^b	✓	BAaaS	G-N-vF vF-vF	— ✓	Verwaltung von vFPGAs über OpenStack, Zugriff über Netzwerk mit komplettem Entwurfsablauf .
Weerasinghe et al. [Wee+15]	✓ ^a	✗	✓	AaaS	G-N-vF vF-N-vF	— ✓	Rechenzentrum auf FPGA-Basis mit OpenStack Erweiterung für AaaS .
Fahmy et al. [FVS15]	✓	✓	✓	AaaS	G-N-H-vF	✓ ✓	Ressourcenverwaltung mit Hypervisor und Middleware zur Bereitstellung von Host-Systemen mit vFPGA.
Kidane et al. [KB16]	✓	✓	✓	HAAas RRaaS	G-N-H-vF vF-vF	✗ ✓	Ressourcenverwaltung mit verteilten vFPGAs und komplettem Entwurfszyklus.
Dondo Gazzano et al. [Don+15]	✓ ^c	—	✓	HPCaaS	G-N-vF	— ✓	Aufteilung von Arbeitspaketen über das Netzwerk auf vFPGAs .
Vineyard Kachris et al. [Kac+16b]	✓	✓	✓	AaaS	G-N-F	✓ !	Auslagern von Funktionsaufrufen auf FPGAs mit bereitgestellten Konfigurationen aus einer Datenbank .
Grigoras et al. [Gri+14]	✓	✓	✓	AaaS	G-N-H-F H-N-F	? !	Unterschiedliche Scheduling-Strategien und elasticsches Management von migrierbaren Beschleunigern.
Proaño et al. [OCC16; PCC16]	✓	✓	✓	BAaaS	G-N-vH-F	✗ ✗	Auslagern von Arbeitspaketen mit Schwerpunkt des Schedulings heterogener Ressourcen in einer Cloud-Architektur.
Iordache et al. [Ior+16]	✓	✓	✓	BaaS	N-H-F	✗ !	Aufteilung von Arbeitspaketen auf skalierbare Gruppen von FPGAs .
Eguro et al. [EV12]	✗	✓	✓	SaaS ^d	G-N-H-F	✗ ✗	Sichere Übertragung zum FPGA und dortige Anonymisierung von Datenbankanfragen.
Pöppelmann et al. [Pöp+15]	✗	✓	✓	SaaS ^d	G-N-H-F	✗ ✗	Homomorphe Ausführung von verschlüsselten Nutzerdaten in der Cloud.
PFC Xue et al. [XSS14]	✗	—	✓	SaaS ^d	G-N-F	— ✗	MapReduce Pipeline vollständig innerhalb des verschlüsselten FPGAs mit zusätzlicher Ent- und Verschlüsselung.

■: FPGAs als Ressource in der Cloud

!: eingeschränkt ?: nicht bekannt

H: Host-System

N: Netzwerk

■: Verwaltung/Scheduling von FPGA-Ressourcen

✓: vorhanden

✗: nicht vorhanden

F: FPGA

■: Sicherheit und Anonymisierung

—: nicht relevant

G: Gateway

^a Erweitertes OpenStack zur Verwaltung

^c Verwaltung von Ressourcen im HPC-Bereich

^b Host nur zur Rekonfiguration und Management der Knoten

^d FPGA nur zur Sicherheit und für Nutzer unsichtbar

II. Verwaltung/Scheduling von FPGA-Ressourcen: Ein oft vernachlässigter Punkt ist die direkte Ver-

waltung spezieller Ressourcen, insbesondere mit einem erweiterten Scheduling, wie in der Arbeit von Grigoras et al. [Gri+14] oder Proaño et al. [OCC16; PCC16] beschrieben. Ein Problem dieser Arbeiten besteht in der fehlenden Virtualisierung der Ressource FPGA, wodurch sich das Scheduling allerdings entsprechend vereinfacht, da ein FPGA immer fest genau einer VM zugeordnet ist oder aber die Abarbeitung einer Batchverarbeitung entspricht.

III. Sicherheit und Anonymisierung: Der Einsatz von FPGAs zur Erhöhung der Sicherheit von flexiblen

Cloud-Architekturen stellt einen weiteren bedeutenden Aspekt dar. Ein Einsatz von FPGAs als rekonfigurierbares Netzwerkinterface mit einer Untersuchung und Klassifikation von Netzwerkpaketen zur Erkennung von Angriffen [SL05] ist ebenso möglich wie Anwendungen zur Ver- und Entschlüsselung von ein- oder ausgehenden Datenpaketen [NWZ13; Rao+16; Rou+04]. Einen ersten Schritt, um zumindest in der Hardware eine abgeschottete Basis zu schaffen, stellt die Arbeit von Caulfield et al. [Cau+16] dar, wobei die Verwaltung der Ressourcen vernachlässigt wird. Die Arbeiten von Eguro et al. [EV12] oder Xu et al. [XSS14] nutzen den FPGA lediglich zur Anonymisierung genau einer Anwendung beziehungsweise eines spezifischen Cloud-Dienstes.

2.5 Einordnung der Arbeit

Die aktuellen Bestrebungen von sowohl Amazon [Ama17a] als auch Microsoft [Put+14], FPGAs in Clouds zu integrieren, zeigen deutlich die Notwendigkeit, FPGAs im Kontext der Cloud näher wissenschaftlich zu betrachten und entsprechende Forschungsschwerpunkte herauszuarbeiten. Die sich aus der in diesem Kapitel vorgestellten Literatur ergebenden offenen Forschungsfragen zu den beiden Teilbereichen der Virtualisierung und der Bereitstellung und Verwaltung von möglichst flexibel einsetzbaren FPGA-Ressourcen werden im Folgenden näher betrachtet. Aus der Vielzahl der Veröffentlichungen in Abbildung B.1 wird deutlich, dass beide Themen in den vergangenen Jahren deutlich an Relevanz gewonnen haben. Nachdem im Folgenden in Abschnitt 2.5.1 offene Forschungsfragen diskutiert werden, widmet sich Abschnitt 2.5.2 der Aufzeigung der Zielsetzung dieser Arbeit und ihres Beitrags zum Wissenschaftsgebiet.

2.5.1 Offene Forschungsfragen

Die Forschungsfragen, welche sich aus den beiden Teilbereichen ergeben, werden im Folgenden getrennt voneinander erläutert, um dann schließlich diese Arbeit und ihre Zielstellung in Abschnitt 2.5.2 besser einordnen und ihren Beitrag abgrenzen zu können.

2.5.1.1 Virtualisierung von Hardwarebeschleunigern

Um FPGAs effizient in eine Cloud einbetten zu können, ist ähnlich wie bei allen anderen Komponenten in der Cloud eine Virtualisierung der Hardware erforderlich. Da gerade in den zukünftigen Cloud-Systemen möglichst große FPGAs eingesetzt werden [Ama17a; Put+14], ist die Virtualisierung der physischen Hardware ähnlich wie bei der in Abschnitt 2.2.3 vorgestellten System-Virtualisierung von großer Bedeutung. Wie in Abschnitt 2.4.1 aufgezeigt wurde diese Fragestellung bisher allerdings nur ansatzweise betrachtet.

Eines der Ziele dieser Arbeit besteht in der Erhöhung der Auslastung der Ressource FPGA indem unterschiedliche Nutzer auf demselben physischen FPGA zugelassen werden, wie in den Arbeiten von Fahmy et al. [FVS15], El-Araby et al. [EGE08], Gazzano et al. [Don+15] oder Weerasinghe et al. [Wee+15] bereits gezeigt. In diesen Arbeiten weisen die virtualisierten FPGAs jedoch eine fixe Größe auf, ebenso unterbleibt eine System-Virtualisierung, sodass das Anhalten und Migrieren von Instanzen nicht möglich ist. Die genannten Arbeiten verzichten ebenso auf die Verknüpfung des FPGAs mit einer Virtuellen Maschine auf dem Host-System, was jedoch für eine Bereitstellung in der Cloud unerlässlich ist. Zum Beispiel setzen die bisherigen in Tabelle 2.5 dargestellten Systeme nur selten VMs mit vFPGAs über PCIe gekoppelt ein. Oftmals erfolgt die Kommunikation stattdessen über das Netzwerk, was den Zugriff zwar deutlich vereinfacht, aber auch zu einer losen Kopplung von Host und FPGA führt. Ebenso existieren derzeit keine Ansätze, neben PCIe auch einen direkten Zugang der FPGAs zum Netzwerk zu etablieren, um somit auch eine Inter-FPGA Kommunikation in einem System mit unterschiedlichen Nutzern auf demselben physischen FPGA zu ermöglichen.

Eine Virtualisierung muss demzufolge zum Ziel haben, den realen FPGA in unterschiedlichen Abstraktionsebenen zunächst hinsichtlich seiner physischen Größe und schließlich vollständig vor dem Nutzer verstecken zu können. Ebenso ist es für eine effektive Virtualisierung eine Migration von Bedeutung, um von der realen Position des FPGAs Rechenzentrum abstrahieren zu können. Die Aspekte sind bisher nur in Teilen Gegenstand der Forschung, wobei der Gesamtkontext einer Integration der FPGAs in eine Cloud oftmals nicht berücksichtigt wird.

2.5.1.2 Bereitstellung spezieller Hardware in einer Cloud-Architektur

Die bisherigen kommerziellen Lösungen beschränken sich auf die Bereitstellung vollständiger physischer FPGAs in einer VM auf dem Gastsystem, ohne jedoch die Ressourcen zu virtualisieren. Dieser Schritt ist aber für ein Cloud-System essenziell, da zur Kosteneinsparung eine hohe Auslastung aller Ressourcen im Rechenzentrum erforderlich ist, wie in Abschnitt 2.3 bereits dargestellt wurde. FPGAs in der Cloud sind derzeit nicht gleichermaßen flexibel und dynamisch einsetzbar wie die anderen (virtualisierten) Komponenten (wie zum Beispiel Prozessor, Speicher, Netzwerk etc.). Die Verwaltung von FPGA-Ressourcen wurde bislang, wie in Abschnitt 2.4 gezeigt, in einer Vielzahl von Arbeiten wie [Cam+16; OCC16; PCC16] untersucht. Diese Arbeiten gehen jedoch nicht auf die Virtualisierung der FPGAs und dementsprechend die Verwaltung virtualisierter FPGA-Ressourcen ein, wie es Ziel der hier vorliegenden Arbeit ist. Da bei aktuellen kommerziellen Ansätzen wie Amazons EC2 F1 [Ama17a] sehr große FPGAs angeboten werden, müssten die Nutzer für eine Maximierung der Effizienz den vollständigen FPGA auslasten, was jedoch nicht immer möglich ist. Ansätze, durch die Virtualisierung von FPGAs die Ressourcenauslastung zu erhöhen, sind daher notwendig.

Darauf aufbauend erfordert eine dynamische Lastverteilung in der Cloud unter Umständen die Migration der VM-Instanz mit der dazugehörigen Hardwarebeschleuniger-Istanz, was bisher nicht ausreichend in der Literatur betrachtet wurde. Um zusätzlich eine möglichst große Dynamik, Adaptivität und universelle Einsatzfähigkeit der Cloud zu ermöglichen, sind ebenso unterschiedliche Cloud-Dienste mit FPGAs erforderlich, die nebenläufig auf demselben System arbeiten und den FPGA universell in der Cloud einsetzbar machen, wie von vielen der Arbeiten in Abschnitt 2.4.4 für einzelne Dienste analysiert wurde. Von Bedeutung ist dabei auch, sowohl für die Beschleunigung als auch für sicherheitsrelevante Anwendungen einen direkten Zugang über den FPGA zu ermöglichen.

2.5.2 Zielsetzung und Beitrag

Die vorliegende Arbeit untersucht, wie rekonfigurierbare Hardware-Architekturen im Kontext von Cloud-Architekturen eingesetzt und genutzt werden können. Die Notwendigkeit der Virtualisierung von FPGAs für den Cloud-Einsatz ergibt sich aufgrund der zunehmenden Verfügbarkeit von physischen FPGAs für die Auslagerung von rechenintensiven Aufgaben [Ama16b; IBM15a; Put+14]. Um die oftmals sehr großen FPGAs effizient auslasten zu können, ist, ähnlich wie bei anderen Komponenten in der Cloud, eine Virtualisierung und Integration der virtualisierten FPGAs (vFPGAs) in eine Verwaltungsstruktur erforderlich. Das zentrale Forschungsinteresse, welches bereits in der Einleitung genannt wurde, lässt sich wie folgt zusammenfassen:

Evaluation eines flexiblen Einsatzes von FPGAs in der Cloud durch Virtualisierung der Ressourcen und Abstraktion von ihrer physischen Position und Größe im Rechenzentrum, zur Bereitstellung einer adaptiven und flexiblen Architektur für unterschiedlichste Anwendungsfälle.

Die Schwerpunkte liegen dabei auf einer flexiblen Verwaltung der FPGAs und deren dynamischer Rekonfiguration sowie in der Virtualisierung der Hardware zur Hintergrundbeschleunigung und der Erhöhung der Sicherheit durch einen direkten Zugang zur dynamisch rekonfigurierbaren Hardware. Eine Virtualisierung der FPGAs, welche eine optimale Auslastung der Ressourcen ermöglicht, bietet dabei durch Nutzung partieller Rekonfiguration und einer Virtualisierung von FPGA und Host sowie der Kommunikation zwischen diesen größtmögliche Flexibilität. Die Virtualisierung erfolgt in dieser Arbeit, nach Analyse unterschiedlicher Ansätze, analog zur System-Virtualisierung mit der Möglichkeit, mehrere Nutzer gleichzeitig auf einer physischen Ressource zuzulassen und voneinander sicher abzukapseln. Des Weiteren werden die FPGAs aus Nutzersicht von ihrer physischen Position im Rechenzentrum losgelöst und erhalten so die Option, unterschiedliche Topologien zu bilden, um unterschiedlichen Anwendungsfällen gerecht zu werden. Auf diese Weise wird es möglich, den Betreiber des Rechenzentrums von den Anbietern eines Dienstes zu entkoppeln. Im Ergebnis können FPGAs so flexibel wie andere virtualisierte Ressourcen in der Cloud eingesetzt werden.

Der Neuigkeitswert der Arbeit besteht in wissenschaftlichen Untersuchungen darüber, wie eine Integration und Verwaltung virtueller FPGAs auf unterschiedlichen Abstraktionsebenen in einem Cloud-System möglich sind. Dabei wird nicht nur der FPGA virtualisiert, sondern – anders als bei vielen anderen Arbeiten – das Gesamtsystem berücksichtigt und der vFPGA dynamisch, je nach Anforderung, virtuell an unterschiedlichen Stellen und Topologien in der Cloud eingesetzt. Im Einzelnen sind die Alleinstellungsmerkmale:

- Grundsätzliche Untersuchungen, wie FPGAs in einer Cloud auf der Anwendungsebene unter Berücksichtigung unterschiedlicher Dienste eingesetzt werden können.
- Konzept zur Virtualisierung der FPGAs analog zur System-Virtualisierung mit Zuständen, der Möglichkeit einer Migration und Kapselung der Nutzer.
- Flexibler Einsatz der vFPGAs innerhalb unterschiedlicher Dienste in der Cloud mit verschiedenen Zugangsmöglichkeiten, um den Anforderungen an eine flexible und universelle Cloud gerecht zu werden.
- Erarbeitung eines entsprechenden FPGA-Hypervisors, um vFPGAs flexibel bereitzustellen zu können, sowie Analysen zu Aufgaben, die an den Host-Hypervisor ausgelagert werden müssen.

- Konzept zur Anpassung der Größe der homogenen vFPGAs an die Ressourcen der Nutzerdesigns sowie Untersuchung der Möglichkeiten der Skalierung von kleinen vFPGAs innerhalb eines physischen FPGAs bis hin zu vFPGAs, welche einen virtuellen Cluster bilden.
- Entwicklung eines Konzeptes zur Einbettung von vFPGAs über PCIe in das Host-System und dessen Verknüpfung mit lokalen VMs.
- Übertragung der Virtualisierung auf die FPGAs in Form einer prototypischen Implementierung mit Abgrenzung und Isolierung der homogenen vFPGAs unterschiedlicher Größe voneinander.
- Prototypische Einbettung von vFPGAs in eine Cloud-Verwaltung mit der Möglichkeit der Migration von VM und dazugehörigen vFPGAs.
- Bewertung und Validierung der prototypischen Implementierung und Auswertung der Ergebnisse sowie der Leistungsfähigkeit des Systems.

Die Abgrenzung zu vergleichbaren Arbeiten besteht hier in Untersuchungen zu unterschiedlichen Ansätzen der Virtualisierung aus den klassischen Bereichen wie Hardware-, System- oder Treiber-Virtualisierung. Das wesentliche Alleinstellungsmerkmal besteht in einer Übertragung der Virtualisierung eines Betriebssystems auf die rekonfigurierbare Hardware. Der Nachweis über die Realisierbarkeit des entwickelten Konzeptes erfolgt durch eine prototypische Implementierung mit Abgrenzung und Isolierung der dynamischen virtuellen FPGAs unterschiedlicher Größe voneinander. Des Weiteren wird ein Konzept zur Verwaltung von zusammengehörigen FPGA/VM-Instanzen in einem dynamischen Cloud-System mit flexilem Zugang zu den vFPGAs entwickelt, um sowohl die Beschleunigung von Cloud-Diensten als auch den Einsatz des FPGAs für sicherheitskritische Anwendungsfälle zu ermöglichen.

3 Anforderungsanalyse und Zielstellung für virtualisierte FPGAs im Cloud-Kontext

In diesem Kapitel werden die Anforderungen an einen universellen FPGA-Einsatz in der Cloud analysiert. Zu diesem Zweck werden diverse Anwendungsfälle und unterschiedliche Sichtweisen auf die FPGAs innerhalb des Cloud-Systems erarbeitet, woraus sich die entsprechenden notwendigen Anforderungen ergeben.

In Abschnitt 3.1 werden die grundlegenden Herangehensweisen und notwendige Dienste für FPGAs in der Cloud herausgestellt. Darauf aufbauend erfolgt in Abschnitt 3.2 der Entwurf einer adaptiven Architektur, welche die FPGAs in unterschiedlichen Konfigurationen zulässt. Abschnitt 3.3 zeigt eine mögliche Einbettung der vFPGAs in eine Systemarchitektur und deren Verwaltungsebene. Abschließend wird in Abschnitt 3.5 ein Systementwurf für die beiden Teilkomponenten der Virtualisierung und der Verwaltung der FPGA-Ressourcen aufgezeigt.

3.1 Abstraktionsschichten zur Etablierung von FPGA-spezifischen Diensten in der Cloud

Um ein System entwerfen zu können, bei dem vFPGAs beliebig mit VMs gekoppelt und in einer Cloud bereitgestellt werden können, wie in Abbildung 3.1 veranschaulicht, ist eine Abstraktion von der physischen Hardware notwendig. Durch angepasste Servicemodelle werden in einem ersten Schritt die unterschiedlichen Hardwareressourcen dynamisch miteinander kombiniert, um diese flexibel nutzen zu können.

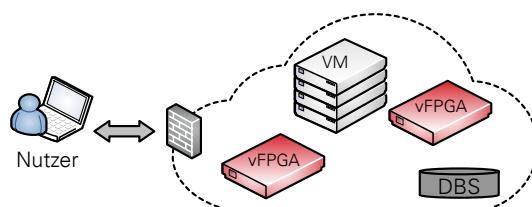


Abbildung 3.1: Cloud mit virtualisierten FPGAs (vFPGAs) und VMs.

Die Bereitstellung von rekonfigurierbarer Hardware in einer Cloud oder einem Mehrbenutzersystem erfordert grundsätzlich auch die Möglichkeit, die FPGAs auf unterschiedliche Arten zu nutzen, sodass in Abschnitt 3.1.1 zunächst eine Analyse unterschiedlicher Nutzertypen und die Festlegung auf entsprechende Servicemodelle in Abschnitt 3.1.2 durchgeführt wird. Abschnitt 3.1.3 zeigt, wie die Servicemodelle aufeinander aufbauen und somit schrittweise eine Abstraktion von der physischen Hardware ermöglichen.

3.1.1 Erweiterung der Beteiligten und Charakterisierung der Rollen

Die in der Literatur etablierten (siehe Abschnitt 2.3.1.5) Rollenverteilungen innerhalb der Cloud müssen für die Bereitstellung von FPGAs um den Betreiber eines Rechenzentrums erweitert werden, da nur dieser uneingeschränkten Zugriff auf die physische Hardware hat. In der Literatur wird häufig nicht zwischen dem Anbieter der Cloud und dem Betreiber des Rechenzentrums (Data Centre Operator) unterschieden. Dieser stellt zwar die physischen Ressourcen für die Cloud bereit, ist aber nicht zwangsläufig auch der Anbieter des Cloud Servicemodells. Insbesondere bei zusätzlicher Spezialhardware wie FPGAs in den Rechenknoten hat der Betreiber des Rechenzentrums eine besondere Bedeutung, da dieser uneingeschränkten Zugriff auf die FPGA-Ressourcen hat.

Bei den zuvor vorgestellten kommerziellen Ansätzen Microsoft Catapult [Cau+16] und Amazon EC2 F1 [Ama17a] werden die FPGAs direkt vom Betreiber der Rechenzentren verwaltet und nicht an die Nutzer für eigene Entwicklungen weitergegeben, was auch einer der Gründe für eine fehlende Virtualisierung der FPGAs in diesen Konzepten ist.

3.1.2 Entwicklung von FPGA-relevanten Servicemodellen/Clouddiensten

Ein entscheidender Aspekt bei der Integration von FPGAs in eine Cloud-Architektur besteht darin, mögliche Anwendungsgebiete und Servicemodelle zu definieren. Der erste Schritt der Abstraktion von physischen FPGAs ist es, eine Einteilung in entsprechende Dienste zu treffen. Eine Anforderung an FPGAs in der Cloud kann zum einen der uneingeschränkte Zugriff auf die Ressourcen sein, um beispielsweise eigene Schaltkreise, I/O-Geräte oder Schnittstellen zu entwickeln (Prototyping). Andererseits kann auch die Entwicklung von FPGA-Hardwarebeschleunigern (Auslagerung von Algorithmen oder sicherheitskritischer Komponenten in Hardware) eine mögliche Zielstellung sein, welche nicht zwangsläufig einen Zugriff auf den kompletten physischen FPGA und die physischen Schnittstellen erfordert. Da die meisten potenziellen Nutzer einer Cloud lediglich eine Anwendung nutzen, benötigen sie keine Kenntnis über den Hardwarebeschleuniger, sondern ausschließlich einen Funktionsaufruf. Die sich nach diesen drei Anforderungen an die FPGAs ergebenen Dienste, welche bereits von Knodel et al. in [KS15c] eingeführt wurden, sind im Einzelnen:

- I. RSaaS – Reconfigurable Silicon as a Service
- II. RAaaS – Reconfigurable Accelerators as a Service
- III. BAaaS – Background Acceleration as a Service

Im Folgenden werden die drei genannten Servicemodelle ausführlich erläutert und mit der NIST-Definition [MG11] von Servicemodellen im Cloud-Computing verglichen. Zusätzlich gibt Abbildung 3.2 eine Übersicht zu den Diensten, Einschränkungen, der Sicherheit und Sichtbarkeit der Hardware. Die Abstufung von Freiheitsgraden durch das vordefinierte Hardwaredesign des FPGAs innerhalb der Modelle wird in Abbildung 3.3 entsprechend veranschaulicht.

3.1.2.1 Reconfigurable Silicon as a Service (RSaaS)

In dem Dienst *Reconfigurable Silicon as a Service (RSaaS)* wird der volle Zugriff auf den kompletten physischen FPGA über eine VM ermöglicht, sodass der Nutzer mit beliebigen Werkzeugen die Hardware seiner Wahl implementieren kann. Für Hardware-Schnittstellen und Treiber-Entwicklung sind Virtuelle Maschinen mit den entsprechend über PCI-Passthrough durchgereichten FPGA-Geräten für die Nutzer

3.1 Abstraktionsschichten zur Etablierung von FPGA-spezifischen Diensten in der Cloud

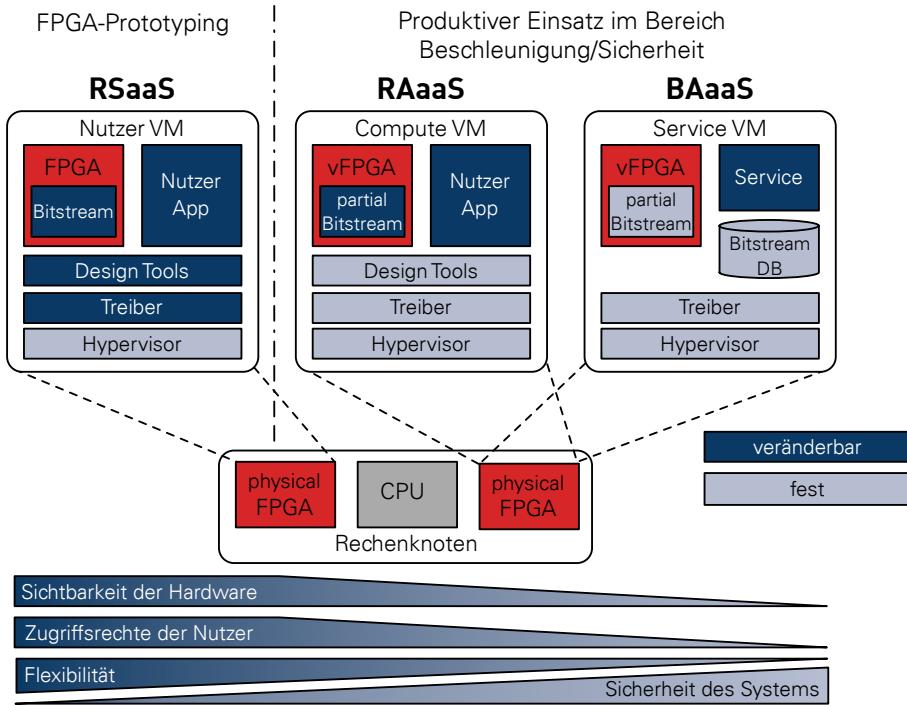


Abbildung 3.2: Übersicht über die drei FPGA-spezifischen Dienste von vollem Zugriff auf die physische Hardware (RSaaS) über Kapselung durch vFPGAs (RAaaS) bis hin zur Hintergrundbeschleunigung ohne direkte Kenntnis der Hardware (BAaaS).

notwendig. Die Zuordnung der FPGAs zu einer VM und entsprechend die Rekonfiguration müssen über einen sicheren Hypervisor erfolgen, um den vollständigen Zugriff auf die physischen FPGAs abzusichern. Der Bitstream für den FPGA kann mit beliebigen Werkzeugen erstellt werden, genauso wie Treiber und Anwendungen innerhalb der VM. Die Flexibilität ist daher in diesem Modell sehr groß, da es keinerlei Einschränkungen für die Nutzer gibt. Die einzige Ausnahme bildet die feste Zuordnung der I/O-Ports aufgrund der angeschlossenen Geräte und um Schäden an der physischen Hardware zu vermeiden. Das vordefinierte Hardwaredesign entspricht Abbildung 3.3(a), wobei die Konfiguration des FPGAs vom Nutzer initial über JTAG erfolgen muss.

Der gesamte Entwicklungsablauf wird somit als Cloud-Service bereitgestellt, ähnlich wie bei Synthesediensten wie Plunify [Plu16], mit dem Ziel, durch hohe Rechenkapazitäten viele unterschiedliche Entwürfe mit verschiedensten Parametern parallel abzuarbeiten. Die Möglichkeit, mehrere Entwurfsabläufe in unterschiedlichen Varianten gleichzeitig auszuführen, kann die Entwurfszeit reduzieren. Parallel zum Software-Flow kann die Implementierung auf realer Hardware einschließlich Validierung und Test auf verschiedenen FPGAs zum Prototyping durchgeführt werden. Da ein Dienst auf dieser Ebene den Nutzern gestattet, den FPGA vollständig neu zu konfigurieren, eröffnet dieses Servicemodell völlig neue Angriffsvektoren, die in aktuellen Cloud-Umgebungen nicht existieren. Die Sicherheit (Zugriff, Ausfall etc.) auf einem Knoten ist geringer als bei anderen Diensten, da die Möglichkeit besteht, das physische System zu zerstören. Das Konzept kann mit den Cloud-Servicemodellen Platform as a Service (PaaS) und Infrastructure as a Service (IaaS) verglichen werden. Der aktuelle Ansatz von Amazon EC2 F1 bietet dem Entwickler vollen Zugriff auf den physischen FPGA mit dem Ziel, leistungsfähige Beschleunigungskerne für weitere Dienste anzubieten [Ama17a].

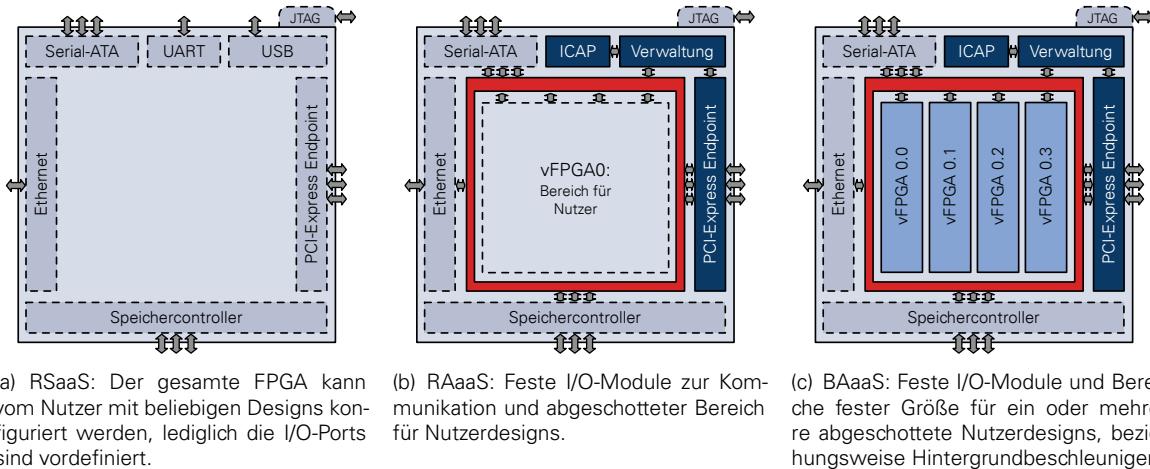


Abbildung 3.3: Hardwaredesigns für die drei Servicemodelle mit sinkender Flexibilität durch vordefinierte Module und Bereiche.

3.1.2.2 Reconfigurable Accelerators as a Service (RAaaS)

Ein weiteres Modell mit allerdings weniger Freiheit für den Nutzer und stärkerem Fokus auf der Beschleunigung von Anwendungen in der Cloud stellt der Dienst *Reconfigurable Accelerators as a Service (RAaaS)* dar, welcher durch das HPCaaS-Konzept [MKH12] inspiriert ist. In diesem Modell wird der FPGA als ausschließlicher Hardwarebeschleuniger verwendet und ist über ein Framework mit API und vordefiniertem Entwurfsprozess nutzbar. Das Framework kapselt die eigentlichen Rechenkerne der Nutzer in einer Virtualisierungs-Schale ein, wie sie in Abbildung 3.3(b) abgebildet ist. vFPGAs unterschiedlicher Größen sind möglich, diese werden dem Nutzer über einen Hypervisor zugewiesen und sind entsprechend der API innerhalb der VM nutzbar. Die partielle Konfiguration erfolgt über den Hypervisor im Host-System und den FPGA-internen ICAP, um höhere Geschwindigkeiten bei der Konfiguration zu erzielen. Der Nutzer muss den Rechenkern für die Anwendung auf dem vFPGA und ein entsprechendes Hostprogramm zur Kommunikation entwerfen. Durch eine derartige Virtualisierungs-Schale wird die Entwicklungszeit erheblich reduziert und der Entwurfsprozess optimiert, da Schnittstellen zur Kommunikation fest vorgegeben sind. Das RAaaS-Modell kann mit dem PaaS-Modell verglichen werden.

Da in dem RAaaS-Modell keine Entwicklung im Bereich der Schnittstellen möglich oder notwendig ist, eignet es sich ideal zur Einbettung in gekapselte vFPGAs. Mittels einer Abstraktionsschicht können vFPGAs in unterschiedlicher Größen angeboten werden, was im Modell RSaaS nicht oder nur mit großem Aufwand möglich ist. Die vom Nutzer entwickelten Beschleuniger können analysiert und getestet werden, um sie später im Modell BAaaS den eigentlichen Anbietern eines Cloud-Dienstes zur Beschleunigung oder Auslagerung ihrer Anwendungen zur Verfügung stellen zu können. Die Sicherheit des Systems wird durch feste Entwurfsabläufe sowie die festen Schnittstellen deutlich erhöht, wodurch das System eine maßgeblich größere Sicherheit aufweist als das RSaaS-Modell. Flexibilität und Sichtbarkeit der physischen Hardware sind jedoch geringer, wie Abbildung 3.2 zeigt.

3.1.2.3 Background Acceleration as a Service (BAaaS)

Die Nutzung der zuvor entwickelten Beschleuniger um Cloud-Dienste im Hintergrund auf die Hardware auszulagern erfolgt innerhalb des Modells *Background Acceleration as a Service (BAaaS)*. Die Beschleuniger, welche zuvor mittels des Dienstes RAaaS entwickelt werden, können als Paket, bestehend aus

3.1 Abstraktionsschichten zur Etablierung von FPGA-spezifischen Diensten in der Cloud

den Bitstreams, für alle möglichen homogenen vFPGAs und die korrespondierende Host-Anwendung in einer Datenbank den Nutzern als Virtual Reconfigurable Acceleration Image (vRAI)¹ bereitgestellt werden. Sämtliche Parameter, wie Anzahl, Ressourcenbedarf und Größe der vFPGAs werden dabei ebenfalls im vRAI hinterlegt. Der Nutzer hat im Modell BAaaS keine direkte Interaktionsmöglichkeit mit dem FPGA, da selbst die Konfiguration im Hintergrund durch das vRAI fest vorgegeben ist.

Die Integration eines Hintergrundbeschleunigers in eigene Anwendungen erfolgt hierbei über eine API, welche im Hintergrund die vFPGAs mit den vorgefertigten Bitstreams (Pre-Built-Bitstreams) konfiguriert und die Kommunikation zwischen Host und vFPGA ermöglicht. Die Konfigurationen und Host-Anwendungen werden vom Cloud-Service-Provider angeboten. Die Zuteilung der erforderlichen VMs zu den vFPGAs auf den physischen Rechenknoten erfolgt im Hintergrund über die Ressourcenverwaltung der Cloud. Entsprechend sind alle wesentlichen Komponenten fest definiert, und der Nutzer kann den ihm bereitgestellten Dienst seinen Bedürfnissen anpassen. Die FPGAs selbst sind nicht sichtbar, jedoch bietet das System eine hohe Sicherheit, da der Nutzer nur noch über Funktionsaufrufe auf die vFPGAs zugreifen kann. Da dieses Modell dem Nutzer konkrete Serviceanwendungen bietet, ähnelt es dem Cloud-Modell SaaS. Anwendungsgebiete sind sicherheitsrelevante Aufgaben [EV12; Mon11] und insbesondere rechenintensive Routinen wie beispielsweise Monte-Carlo-Simulationen aus dem Finanzbereich [KCL08; TBG08].

Die Motivation zur Nutzung des BAaaS-Modells besteht in der Einsparung von VM-Ressourcen durch die Auslagerung auf vFPGAs, welche entsprechend ihrer Energiebilanz für die korrespondierende Anwendung günstiger sind als herkömmliche Rechenressourcen wie Prozessoren oder GPGPUs. Somit können durch eine Hintergrundbeschleunigung oder für spezielle sicherheitskritische Funktionen FPGAs eingesetzt werden, ohne das der Endnutzer Kenntnis davon hat.

3.1.2.4 Abschließende Bewertung und Vergleich mit kommerziellen Ansätzen

Die FPGAs in Amazons EC2 F1-Cloud [Ama17a] sind in der derzeitigen Auslegung mit dem Modell RSaaS vergleichbar, wobei davon auszugehen ist, dass hier langfristig lediglich komplett Pakete zur Auslagerung von rechenintensiven Anwendungen für Service-Anbieter auf der SaaS-Ebene vorgesehen sind, was dem Modell BAaaS entspricht. Dazu ist insbesondere eine Virtualisierung der FPGAs erforderlich, um die Ressourcen ähnlich effizient auszulasten, wie dies durch eine klassische System-Virtualisierung für Rechensysteme erreicht wurden. Die Microsoft-Catapult-Architektur [Put+14] ist hingegen aufgrund der bereitgestellten Schale für ein eigenes Hardwaredesign mit dem Modell RAaaS vergleichbar. Ein direktes Bereitstellen unterschiedlicher FPGAs für das Hardware/ASIC-Prototyping ist nur für spezielle Kunden aus dem Bereich von Forschung und Entwicklung im Schaltkreisentwurf von Interesse und wird daher nicht weiter betrachtet.

Kommerzielle und effiziente Cloud-Lösungen müssen entsprechend große FPGAs bereitstellen, welche durch zusätzliche Virtualisierung optimal ausgelastet werden können. Dadurch können die Ressourcen möglichst flexibel eingesetzt werden und die Nutzer erhalten exakt die Infrastruktur, welche sie für ihre Anwendungen benötigen.

3.1.3 Vertikaler Aufbau der Cloud-Dienste

Für den Anbieter eines Dienstes auf der Ebene SaaS, der diesen Dienst im Hintergrund mittels des BAaaS beschleunigen möchte, um Betriebskosten zu sparen, sind entsprechend die anderen beiden

¹In Anlehnung an Amazons AFI mit dazugehörigen AMI.

3 Anforderungsanalyse und Zielstellung für virtualisierte FPGAs im Cloud-Kontext

Abstufungen oder Dienste erforderlich. Wie in Abschnitt 2.3.1.3 und speziell in Abbildung 2.15 gezeigt, bauen die klassischen Cloud-Dienste ebenfalls vertikal aufeinander auf. Abbildung 3.4 veranschaulicht, wie schrittweise die physischen FPGAs vom Betreiber des Rechenzentrums zunächst vollständig an die Cloud-Provider weitergegeben werden (RSaaS). Die Cloud-Provider entwickeln ihre Anwendungen für die vFPGAs (RAaaS) und stellen schließlich die vorgefertigten vFPGA-Beschleuniger (vRAI) für die eigentlichen Service-Provider im BAaaS-Modell bereit. Der Service-Provider ist der letzte Akteur in der Kette, der sich der Nutzung von rekonfigurierbarer Hardware bewusst ist und schließlich einem Endnutzer einen klassischen Dienst auf oberster Ebene bereitstellt (SaaS), welcher im Hintergrund auf vFPGAs ausgelagert wird.

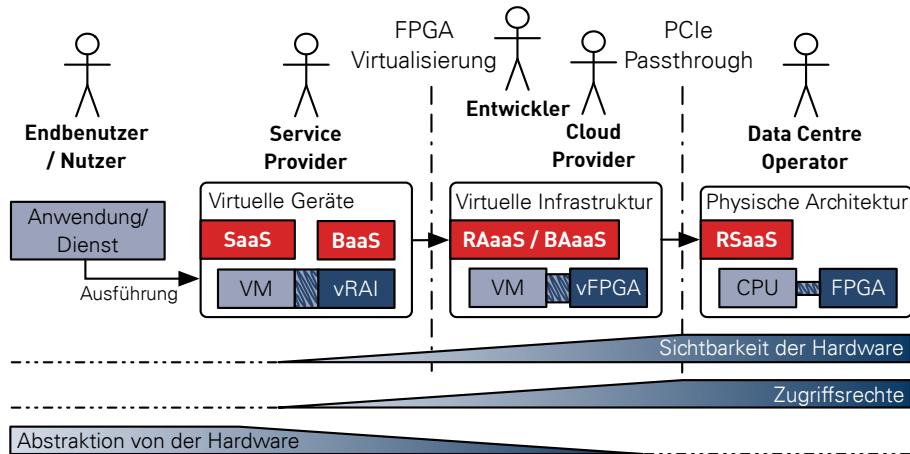


Abbildung 3.4: Verkettung der FPGA-spezifischen Cloud-Dienste mit den beteiligten Akteuren und den Abstraktionsstufen vFPGA für RAaaS und schließlich vRAI für BAaaS, sowie der immer stärkeren Verschmelzung von VM und FPGA.

Die eigentliche Funktion oder Nutzungsart der FPGAs ist aufgrund der nachfolgenden Einbettung in eine Cloud-Architektur mit zusätzlichen Freiheitsgraden deutlich flexibler als ein reiner Einsatz zur Beschleunigung von Anwendungen auf der Ebene eines einfachen Coprozessors.

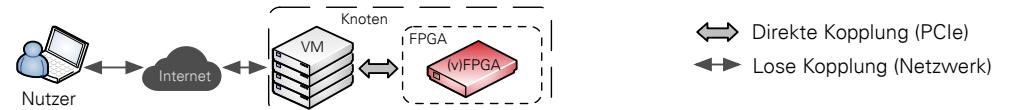
3.2 Entwurf einer adaptiven Architektur mit dynamischem Einsatz von FPGAs

Um die Integration von universell nutzbaren FPGAs in eine Cloud zu erreichen, welche dynamisch eingesetzt werden können, werden zunächst in Abschnitt 3.2.1 unterschiedliche Szenarien und Anwendungsfälle für eine Nutzung aufgezeigt. In Abschnitt 3.2.2 werden schließlich die Einsatzmöglichkeiten skalierbarer vFPGAs erläutert, bevor anschließend in Abschnitt 3.2.3 die entsprechenden Anforderungen an eine Virtualisierung der FPGAs beschrieben werden. Zusammenfassend werden in Abschnitt 3.4 Begriffe im Rahmen der Virtualisierung von FPGAs eingeführt und definiert.

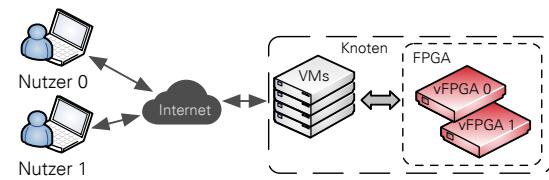
3.2.1 Virtuelle Sicht der Nutzer auf die FPGAs innerhalb der Cloud

Um den adaptiven Einsatz von FPGAs in einer Cloud zu ermöglichen, werden zunächst unterschiedliche Sichtweisen und Anwendungsszenarien für die virtualisierten Ressourcen diskutiert. Abbildung 3.5 stellt die verschiedenen logischen Sichten auf die vFPGAs dar.

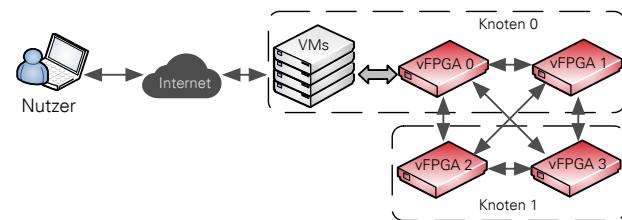
3.2 Entwurf einer adaptiven Architektur mit dynamischem Einsatz von FPGAs



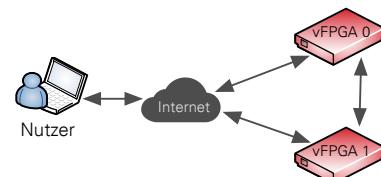
(a) Ein Nutzer hat über eine VM Zugriff auf einen kompletten physischen oder virtualisierten FPGA (vFPGA).



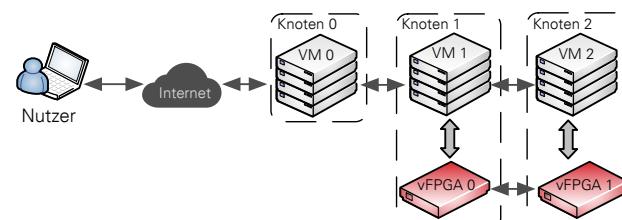
(b) Mehrere Nutzer teilen sich einen physischen FPGA durch direkte Nutzung von vFPGAs oder indirekte durch Hintergrundbeschleunigung.



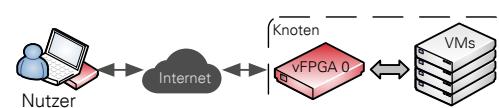
(c) Ein Nutzer hat über eine VM Zugriff auf einen Cluster aus mehreren vFPGAs, welche über das Netzwerk lose gekoppelt sind.



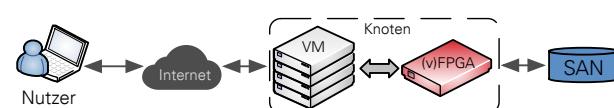
(d) Aufbau eines Systems ausschließlich aus vFPGAs, welche lose über das Netzwerk gekoppelt sind.



(e) Aufbau eines logischen Clusters mit vFPGAs über PCIe, wobei die physischen FPGAs untereinander zusätzlich über das Netzwerk gekoppelt sind.



(f) Direkter Zugang zu einer VM über einen mittels PCIe gekoppelten vFPGA mit direktem Netzwerkgang zur Ver- und Entschlüsselung der Daten und Authentifizierung direkt in Hardware.



(g) Zugang zu einem SAN direkt über den über PCIe gekoppelten FPGA zur Ver- und Entschlüsselung der Daten.

Abbildung 3.5: Logische Sicht auf die virtuellen Ressourcen vFPGA und VM in einer Cloud-Architektur mit Zuordnung zu den physischen Rechenknoten.

Der Einsatz des FPGAs als Hardwarebeschleuniger beziehungsweise Coprozessor wird in Abbildung 3.5(a) gezeigt. Der Nutzer greift über das Internet auf eine ihm zugewiesene VM zu, der ein vollständiger FPGA innerhalb des Knotens zur Verfügung steht. Die bereitgestellte Datenrate ist aufgrund der direkten Kopplung von VM und FPGA relativ hoch. Wird keine vordefinierte Schnittstelle auf dem FPGA verwendet, sondern ein voller Zugriff gewährleistet, entspricht dies dem Modell RSaaS. Im Falle des Einsatzes einer vordefinierten Virtualisierungs-Schale als FPGA-Infrastruktur zur Erhöhung der Sicherheit und Etablierung von vFPGAs ist der Dienst hingegen entweder dem RAaaS- oder dem BAaaS-Modell zuzuordnen. Bei Nutzung der entsprechenden Modelle mit vFPGAs ist die Aufteilung eines physischen FPGAs auf mehrere Nutzer möglich, wie Abbildung 3.5(b) veranschaulicht.

Ebenso ist es notwendig, mehrere vFPGAs zu einem Cluster zusammenzufassen. Dabei können sich die vFPGAs sowohl auf demselben physischen Gerät als auch auf unterschiedlichen FPGAs in unterschiedlichen Knoten befinden. Zur Optimierung der Kommunikation zwischen den vFPGAs ist ein direkter Zugang zu einem Verbindungsnetzwerk erforderlich, um die Host-Systeme der FPGAs nicht zusätzlich zu belasten. Abbildung 3.5(c) zeigt ein System aus mehreren vFPGAs, welche sowohl über unterschiedliche physische FPGAs als auch über verschiedene Rechenknoten hinweg verteilt sind. Der Zugriff erfolgt über eine VM auf einem Host-System zur Vor- und Nachbearbeitung der Daten, ist aber auch direkt möglich, wie in Abbildung 3.5(d) gezeigt. Zur optimalen Anpassung des Systems an unterschiedliche Anforderungen ist es notwendig, auch virtuelle Architekturen mit reinen vFPGA-Hardwarebeschleunigern mit enger Kopplung an eine VM bereitzustellen, wie in Abbildung 3.5(e) dargestellt.

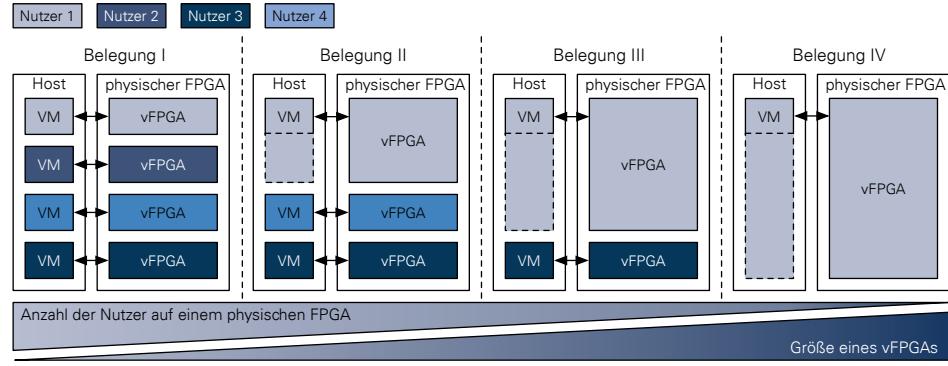
Zudem ist es erforderlich den direkten Zugang zu einer VM über einen FPGA im selben Knoten bereitzustellen, um den ein- und ausgehenden Datenverkehr zu analysieren und zu filtern. Entsprechend kann mit einem zusätzlichem FPGA beim Nutzer eine vollständige Ende-zu-Ende-Verschlüsselung in Hardware aufgebaut werden, wie in Abbildung 3.5(f) skizziert. Abbildung 3.5(g) zeigt zusätzlich, wie der (v)FPGA auch für die Ver- und Entschlüsselung von Daten im SAN eingesetzt werden kann.

3.2.2 Flexibilität im Einsatz von FPGAs in der Cloud

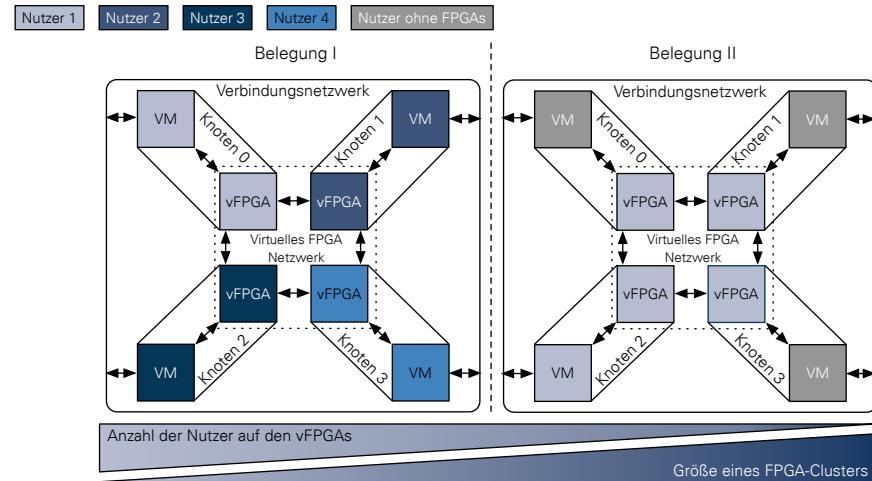
Neben den im vorherigen Abschnitt aufgezeigten Szenarien, mit der dortigen Sichtweise auf die dynamisch und adaptiv einsetzbaren Ressourcen, ist eine Abstraktion von ihrer physischen Größe beziehungsweise der Anzahl der internen Ressourcen (LUTs, DSP, BRAM etc.) notwendig, um sowohl für kleine als auch für große Entwürfe innerhalb eines Clusters von FPGAs eingesetzt werden zu können, wie in Abbildung 3.6 aufgezeigt.

Die Virtualisierung der FPGAs zur besseren Auslastung der Ressourcen stellt dabei vom Ansatz her eine klassische System-Virtualisierung der Hardware dar, mittels derer unterschiedlichste Nutzer in ihrem gekapselten vFPGA arbeiten können, ohne dabei andere Nutzer zu beeinträchtigen. Da die FPGAs als Hardwarebeschleuniger eingesetzt werden, ist des Weiteren ein logischer Kanal zwischen dem vFPGA und einer VM auf dem Host notwendig. Abbildung 3.6(a) zeigt einen physischen FPGA, mit vFPGAs unterschiedlicher Größe und der Zuordnung zu VMs und Nutzern. Somit wird ein in seiner Größe und der Anzahl der Nutzer skalierbares System bereitgestellt. Wenn zusätzliche vFPGAs dem Nutzer zugeordnet werden können, entspricht dies einer horizontalen Skalierung. Wird allerdings die Größe eines vFPGAs verändert, liegt eine vertikale Skalierung vor (siehe Abschnitt 2.3.1.4). Insbesondere bei FPGAs eignet sich die vertikale Skalierung deutlich besser, da auf diese Weise die Ressourcen effizienter ausgelastet werden können. In Kapitel 6 wird anhand eines Szenarios gezeigt, wie und für welche Beispieleanwendungen die vertikale Skalierung effizient eingesetzt werden kann.

3.2 Entwurf einer adaptiven Architektur mit dynamischem Einsatz von FPGAs



(a) Flexible Größe und Nutzeranzahl eines FPGAs durch Virtualisierung. Gezeigt wird das Beispiel eines physischen FPGAs, der in bis zu vier vFPGAs unterschiedlicher Größe unterteilt ist.



(b) Skalierung über die physischen FPGAs hinaus am Beispiel eines Systems mit mehreren Nutzern auf vier Knoten. Jeder vFPGA ist in diesem Beispiel genau einer VM zugeordnet und die vFPGAs belegen einen kompletten physischen FPGA.

Abbildung 3.6: Skalierung der vFPGAs in ihrer Größe von einem kleinen Design innerhalb eines physischen FPGAs bis hin zu einem Cluster aus mehreren FPGAs. Die Farben zeigen die Zuordnung von VM zu vFPGA.

Um eine Skalierbarkeit der Größe der vFPGAs über den physischen FPGA hinaus zu erreichen, ist der Aufbau eines virtuellen FPGA-Clusters erforderlich. Es sollte die Möglichkeit bestehen, ein Nutzerdesign auf mehrere physische FPGAs auszudehnen, wie in Abbildung 3.6(b) aufgezeigt. Dabei wird über eine VM ein vFPGA direkt angesprochen und die direkte Kommunikation zwischen den bereitgestellten vFPGAs erfolgt über ein virtuelles Verbindungsnetzwerk, welches entweder ein dediziertes FPGA-Netz oder das Verbindungsnetzwerk der Cloud sein kann, wie auch in der Arbeit von Yoshimi et al. [Yos10] der Fall. Die automatisierte Übertragung eines einzelnen Hardwaredesigns auf ein FPGA-Cluster ist hingegen Gegenstand der Arbeit von Tang et al. [TMT14].

3.2.3 Anforderungen an eine FPGA-Virtualisierung

Aus den bisher betrachteten Anforderungen der Dienste RAaaS und BAaaS in Abschnitt 3.1.2, der virtuellen Sicht auf unterschiedliche Szenarien in Abschnitt 3.2.1 und der erforderlichen Skalierbarkeit hinsichtlich der Größe der vFPGAs von Teilbereichen auf dem physischen FPGA bis hin zu Clustern aus mehreren FPGAs, wie in Abschnitt 3.2.2 erläutert, ergeben sich eine Reihe von Anforderungen an eine

3 Anforderungsanalyse und Zielstellung für virtualisierte FPGAs im Cloud-Kontext

FPGA-Virtualisierung. Diese werden im Folgenden erläutert. Das Hauptaugenmerk richtet sich dabei auf die Hintergrundbeschleunigung von Anwendungen und die optimale Auslastung der physischen FPGAs durch deren Virtualisierung. Aus den in Abschnitt 3.1 und Abschnitt 3.2 erarbeiteten Anforderungen ergeben sich folgende Hauptaugenmerke und Schlussfolgerungen für eine FPGA-Virtualisierung:

- Virtualisierung der FPGAs analog zur System-Virtualisierung mit System-Hypervisor zur Kapselung und Isolierung der Nutzer voneinander sowie abgesicherte Konfiguration der vFPGAs durch den Host-Hypervisor.
- Interaktion mit einem Host/FPGA-Hypervisor zur Konfiguration des gesamten (RSaaS) und des virtualisierten (RAaaS/BAaaS) FPGAs mit einer Validierung des Bitstreams.
- Abstraktion von der physischen Größe der FPGAs durch vertikale Skalierung und Anpassung an den Ressourcenbedarf der Nutzer:
 - Notwendigkeit, vFPGAs schrittweise mit anderen vFPGAs auf einem physischen FPGA zu kombinieren, um die Anzahl der verfügbaren Ressourcen zu vergrößern.
 - Mehrere unabhängige Nutzer auf einem physischen FPGA, um diesen effizient auszulasten.
 - Konfiguration unterschiedlicher vFPGAs an beliebigen Regionen auf dem physischen FPGA mit dem selben Hardwaredesign.
 - Möglichkeit der Zusammenschaltung mehrerer, dem Nutzer zugehöriger vFPGAs über direkte Kommunikationskanäle zu einem virtuellen FPGA-Cluster.
- Erforderliche Schnittstellen zur Kommunikation gemäß der in Abschnitt 3.2.1 vorgestellten Szenarien:
 - Zum Einsatz als Hardwarebeschleuniger muss eine enge Kopplung mit stream-basierter Kommunikation zwischen vFPGA und VM auf dem Host-System verfügbar sein,
 - Zugang zur Cloud durch direkten Netzwerkzugang über den FPGA und
 - Aufbau von Kommunikationskanälen für einen virtuellen FPGA-Cluster mit direkter Kommunikation zwischen den vFPGAs.
- Bibliotheken und API zur Kommunikation und Konfiguration des vFPGAs für Nutzer des Modells RAaaS.
- Bereitstellung von Beschleuniger-Paketen mit Bitstreams und Host-Anwendung als vRAI zum Einsatz im Servicemodell BAaaS.
- Möglichkeit eines Zugangs zu gemeinsam genutzten Ressourcen auf dem FPGA und insbesondere dem FPGA-Board (zum Beispiel auf dem DDR-Speicher).
- Zustandsverwaltung der vFPGAs mit der Möglichkeit einer Migration des Hardwaredesigns auf andere physische FPGAs beziehungsweise vFPGAs auf dem gleichen FPGA.

Im folgenden Abschnitt wird zunächst eine Systemarchitektur erarbeitet, welche diesen Anforderungen gerecht wird, bevor in Kapitel 4 ein entsprechendes Konzept zur Virtualisierung eines FPGAs vorgestellt wird.

3.3 System- und Softwarearchitektur für die Verwaltung virtualisierter FPGA-Ressourcen

Im Folgenden wird der Entwurfsraum für die System- und Softwarearchitektur zur Verwaltung der virtualisierten FPGA-Ressourcen entsprechend den Schlussfolgerungen aus Abschnitt 3.2 analysiert. Die daraus resultierende Architektur stellt die Basis für die folgende FPGA-Virtualisierung und die übergeordnete Systemarchitektur der Cloud dar. Dazu werden zunächst Knoten entworfen, die neben Prozessoren auch FPGAs enthalten und auf die die in Abschnitt 3.2.1 vorgestellten virtuellen Systeme und Szenarien abgebildet werden können, bevor anschließend die erforderlichen Komponenten zur Verwaltung herausgearbeitet werden.

3.3.1 Analyse einer geeigneten Hardware-Architektur

Bei der Integration von FPGAs in ein Rechenzentrum für eine Cloud sind Aspekte wie Skalierbarkeit, Wartungsintensität und Auslastung zu beachten, wie bereits in [KLS16] erläutert. Ein wesentlicher Schritt zur Entwicklung einer geeigneten Architektur ist daher die Wahl der Kopplung von Prozessor und FPGA, wie sie bereits für FPGA-basierte Hardwarebeschleuniger in Abschnitt 2.1.2.3 erläutert wurde. Anhand von Abbildung 3.7 werden verschiedene Ansätze zur Integration von FPGAs aufgezeigt. Diese Ansätze werden unter dem Gesichtspunkt bewertet, wie darauf eine universell einsatzfähige Architektur aufgebaut werden kann, auf welche sich die Szenarien aus Abschnitt 3.2.1 abbilden lassen. Dafür ist einerseits eine enge Kopplung von Prozessor und FPGA, aber auch der direkte Zugriff auf den FPGA über das Netzwerk erforderlich, um den unterschiedlichen Anforderungen und Szenarien, wie dem Einsatz als Hardwarebeschleuniger oder als sicherer Zugangspunkt, gerecht werden zu können.

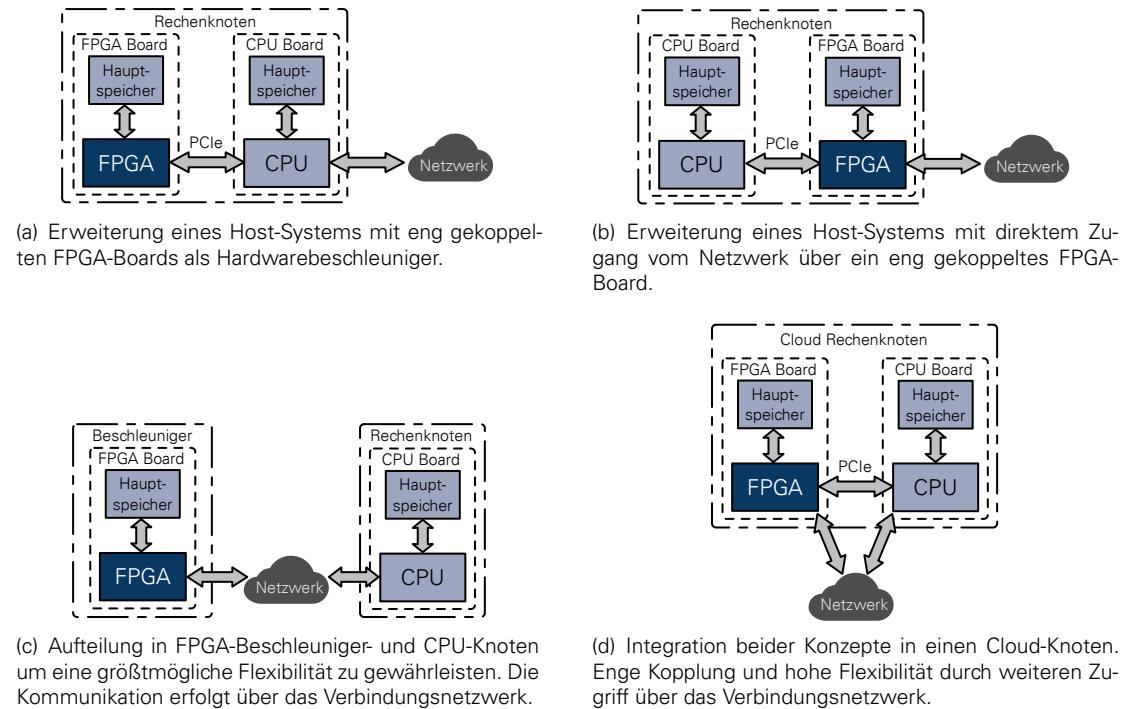


Abbildung 3.7: Unterschiedliche Möglichkeiten der physischen Integration rekonfigurierbarer Hardware in eine Cloud-Architektur. Nach [KLS16].

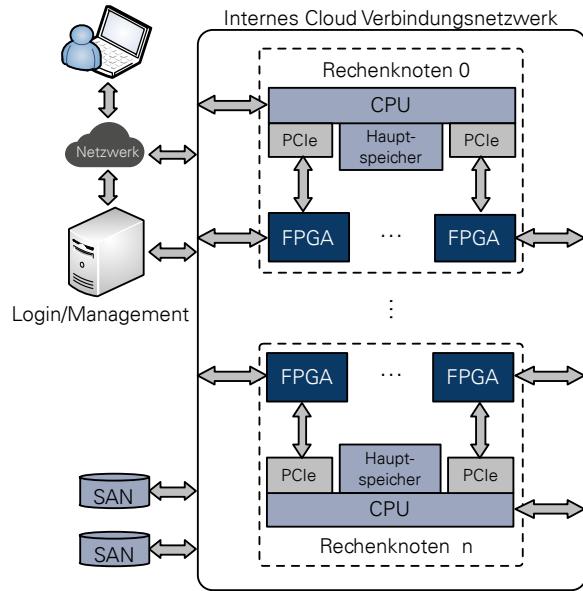


Abbildung 3.8: Physische Zielarchitektur für die unterschiedlichen Szenarien mit Rechenknoten bestehend aus Prozessor und einer beliebigen Zahl von über PCIe gekoppelten FPGAs.

Der in der Vergangenheit am häufigsten genutzte Ansatz für Rechenknoten bestehend aus Prozessor und FPGA ist die in Abschnitt 2.1.2.3 vorgestellte enge Kopplung mit serieller Punkt-zu-Punkt-Verbindung und hoher Datenrate, wie beispielsweise PCIe, QPI oder HyperTransport-Link. Ein solches hybrides System kann für eine Vielzahl von Anwendungen eingesetzt werden und ist die Basis zahlreicher Arbeiten [FVS15; Kac+16b; PCC16; Put+14]. Hochparallele Berechnungen lassen sich auf den FPGA auslagern, sobald eine Anwendung dies erfordert. Eine effiziente Vor- und Nachbearbeitung der Daten in Software ist durch die enge Kopplung ebenfalls einfach durchführbar. Das Problem eines solchen in Abbildung 3.7(a) gezeigten Systemaufbaus besteht jedoch darin, dass der FPGA nur über den Host erreichbar ist und dieser bei jeder Nutzung des FPGAs beteiligt ist. Fällt der Host aus, ist keine Nutzung des FPGAs mehr möglich, was in einem Rechenzentrum aufgrund der Wartungsintervalle zum vollständigen Verlust dieser Ressource über einen längeren Zeitraum führen kann. Eine weitere Möglichkeit besteht darin, den Zugang zum Host über den FPGA bereitzustellen, wie in Abbildung 3.7(b) aufgezeigt. Der FPGA wird dabei als direkte Schnittstelle zum Netzwerk eingesetzt und kann zusätzlich Operationen wie beispielsweise Ver- und Entschlüsselung des Datenverkehrs übernehmen. Fällt der FPGA aus, ist zwangsläufig der gesamte Knoten nicht mehr über das Netzwerk erreichbar.

Den FPGA als alleinige Ressource mit direktem Netzwerzkzugang zu nutzen, wie in Abbildung 3.7(c) gezeigt, führt zu einer eigenständigen und insbesondere weitaus flexibler einsetzbaren Komponente. Weiterhin ist systemweit die einfache Verteilung der Anfragen auf die Ressourcen möglich, da beliebig viele VMs und FPGAs miteinander kommunizieren können, wie eine Reihe von Arbeiten gezeigt haben [Don+15; Wee+15]. Da aber eine Vielzahl der in Abschnitt 2.4.2 vorgestellten Cloud-Anwendungen einen Dienst in Software im Hintergrund beschleunigen, ist die enge Kopplung zwischen Dienst und Beschleuniger notwendig, um ein flexibles System aufzubauen.

Die Verwaltung und insbesondere die Konfiguration des FPGAs ohne Zugriff von einem Rechensystem erschwert zudem die Wartung, da der Zugang zum FPGA-Board ausschließlich über das Netzwerk und ein entsprechendes Hardwaredesign möglich ist. Ein integrierter oder zum FPGA eng gekoppelter Prozessor ist daher eine Voraussetzung für ein flexibles und einfach zu wartendes System.

3.3 System- und Softwarearchitektur für die Verwaltung virtualisierter FPGA-Ressourcen

Ein flexibel einsetzbares und den aufgezeigten Anforderungen gerechtes System stellt somit eine Verknüpfung aus beiden Ansätzen dar, wie sie Abbildung 3.7(d) zeigt. Im Gegensatz zu der Catapult-Architektur [Put+14], welche nur über einen physischen Netzwerkanschluss verfügt und jeglichen Netzwerkverkehr durch den FPGA leiten muss, besitzt ein solches System den Vorteil, bei einem Ausfall einer der beiden Komponenten die verbleibende Ressource weiterhin für einen Teil der Anwendungen nutzen zu können. Ein Prozessor ohne FPGA kann weiterhin VMs bereitstellen und ein FPGA ohne Host kann für den Aufbau eines FPGA-Clusters nachgenutzt werden. So stehen die einzelnen Komponenten auch bei größeren Wartungszyklen zur Verfügung.

Die resultierende Hardwarearchitektur ist in Abbildung 3.8 dargestellt. Im Beispiel sind zwei physische FPGAs in einem Knoten integriert, wobei die Anzahl beliebig ist. Allerdings sollte die Anzahl der FPGAs die Zahl der physischen Prozessorkerne aus Gründen der Leistungsfähigkeit nicht überschreiten. Durch den direkten Zugang zu den FPGAs über das Verbindungsnetzwerk und die eng gekoppelte VM auf dem Host-System, welches ebenfalls über einen direkten Zugang zum Netzwerk verfügt, sowie durch die Option, eine direkte Kommunikation zwischen den FPGAs aufzubauen, können sämtliche Szenarien aus Abschnitt 3.2.1 auf diese physische Architektur abgebildet werden.

3.3.2 Hierarchische Aufteilung der Verwaltungsaufgaben

Die virtuellen Ressourcen, welche dem Nutzer bereitgestellt werden, sind einerseits VMs und andererseits daran gekoppelte FPGAs. Eingehende Anfragen werden somit zunächst an Knoten mit den entsprechenden freien Ressourcen weitergeleitet. Dazu sind zum einen ein Verwaltungsknoten erforderlich, welcher eine Gesamtansicht auf das System hat, und zum anderen die Rechenknoten mit Prozessoren und FPGAs, wie in Abbildung 3.9 gezeigt. Die Verwaltung gliedert sich entsprechend in die Ebenen der Cloud und der einzelnen Knoten, um die Anfragen hierarchisch auf die involvierten Knoten zu verteilen. Wird der Nutzer, der sich über sein lokales System mit der Cloud verbindet, ebenso in den Systementwurf mit einbezogen, ergibt sich die in Abbildung 3.9 dargestellte Systemarchitektur.

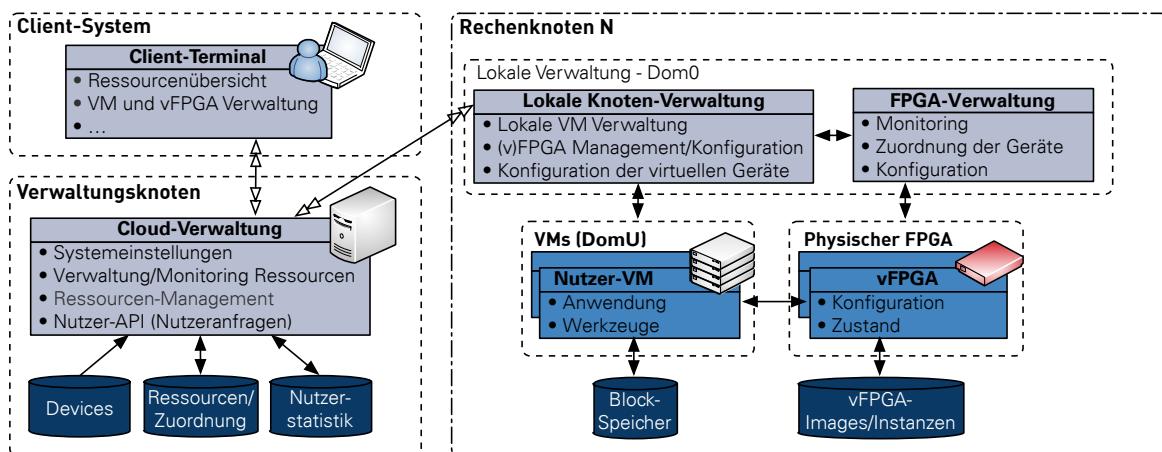


Abbildung 3.9: Logische Systemarchitektur der Verwaltung von FPGA-Ressourcen, bestehend aus einem Knoten zur Verwaltung und den Rechenknoten mit VMs und entsprechenden eng gekoppelten vFPGAs.

Die wesentliche Komponente bildet dabei die Cloud-Verwaltung, welche eine globale Sicht auf das Gesamtsystem ermöglicht, die Belegung verwaltet und die konkreten Anfragen an die Rechenknoten mit den virtualisierten FPGAs delegiert. Die Rechenknoten müssen dazu die Zuweisung der Ressourcen im Detail kennen und deren aktuelle Auslastung an die Cloud-Verwaltung übermitteln. Durch den hierar-

3 Anforderungsanalyse und Zielstellung für virtualisierte FPGAs im Cloud-Kontext

chischen Aufbau wird die Arbeitslast auf das gesamte System verteilt und durch die Auslagerung von Teifunktionen in die Knoten die Kommunikation möglichst lokal gehalten, um Latenzen zu reduzieren.

3.3.2.1 Aufgaben der Cloud-Verwaltung

Der Verwaltungsknoten hat die Aufgabe, den Nutzern eine Schnittstelle zur Administration der eigenen Ressourcen zu bieten. Dabei hat der Nutzer die Möglichkeit, die gewünschten virtuellen Ressourcen zu konfigurieren, zu allokalieren und wieder freizugeben. Eine wichtige Bedingung aus Abschnitt 3.3.3 ist des Weiteren, dass der Nutzer einen der in Abschnitt 3.1.2 eingeführten Dienste auswählen kann. Über den Cloud-Provider, welcher die Rechte innerhalb der Verwaltungsebene administriert, erhält der Nutzer die entsprechenden Rechte für die Nutzung der Dienste oder der Ressourcen. Die Zuweisung zu den Knoten erfolgt in einem zweiten Schritt über einen Scheduler. Die Informationen zu Nutzern, Knoten und FPGAs werden zentral in einer Datenbank verwaltet, wobei ein Parameter die Auslastung eines Knotens und insbesondere der des physischen FPGAs angibt. Die weiteren detaillierten Daten zur Auslastung und den konkreten Zuordnungen von VM zu (v)FPGA erfolgt innerhalb des Rechenknotens mit den konkreten allokierten Ressourcen.

3.3.2.2 Lokale Verwaltung innerhalb des Rechenknotens

Der Rechenknoten mit einem oder mehreren physischen FPGAs, wie in Abbildung 3.9 gezeigt, erhält die Anfragen und ordnet dem Nutzer konkrete VMs und (v)FPGAs der gewünschten Größe und Kopplungsart zu. Die Konfiguration des (v)FPGAs erfolgt auf Basis des gewählten Dienstes und der entsprechenden Konfigurationsart. Eine vollständige Konfiguration, wie sie im Modell RSaaS erforderlich ist, erfolgt über den Zugriff auf den physischen FPGA über dessen JTAG-Interface. Innerhalb der Modelle RAaaS und BAaaS erfolgt die Konfiguration partiell für die einzelnen vFPGAs jeweils über die interne ICAP-Schnittstelle innerhalb eines Hypervisors auf dem FPGA. Die Übertragung der Konfiguration verläuft über PCIe und einen Kanal, auf den nur ein Hypervisor auf dem Host innerhalb der FPGA-Verwaltung zugreifen darf. Die FPGA-Verwaltung erhält infolge dessen eine wesentliche Rolle bei der Verwaltung und Konfiguration der lokalen (v)FPGAs.

Der vom Nutzer stammende partielle Bitstream für einen zugewiesenen vFPGA muss entsprechend auch von der FPGA-Verwaltung auf gültige Regionen innerhalb des FPGAs überprüft werden, um einen Zugriff auf Bereiche fremder vFPGAs und Nutzer zu vermeiden. Eine weitere Aufgabe der FPGA-Verwaltung besteht somit darin, den vFPGA über seine dedizierten Kommunikationskanäle der dem Nutzer zugewiesenen VM zuzuordnen und hierbei wiederum Zugriffe auf Kanäle anderer Nutzer zu verhindern.

3.3.3 Anforderung an die Verwaltung von FPGAs in einem Cloud-System

Um die virtualisierten FPGAs mit den Anforderungen aus Abschnitt 3.2.3 in eine Cloud einzubetten, ergeben sich entsprechend der vorherigen Abschnitte folgende Anforderungen an das Gesamtsystem und speziell an den Knoten mit den Ressourcen, welcher über sowohl Host- als auch FPGA-Hypervisor die Abschottung der Nutzer voneinander und die Stabilität des Systems gewährleistet:

- Einbettung der unterschiedlichen Servicemodelle in die gemeinsame Verwaltungsstruktur.
- Konfigurationsdatei zur Beschreibung einer vFPGA-Ressource und des virtuellen Systems entsprechend eines der in Abschnitt 3.2.1 vorgestellten Szenarien.

- Grobgranulare Verteilung der unterschiedlichen Nutzer und der Arbeitspakete entsprechend ihrer Anforderungen an die Ressourcen.
- Berücksichtigung der Möglichkeiten zur Skalierung (horizontal und vertikal) der vFPGA Ressourcen im Modell BAaaS, um in einer elastischen Cloud die Ressourcen optimal an die Bedürfnisse der Nutzer anzupassen.
- Verwaltung der aktuellen Belegung und Zuweisung zu Nutzern auf Cloud-Ebene für den kompletten Cluster.
- Spezielle Anforderungen an die lokale Verwaltung der Ressourcen innerhalb von Host- oder FPGA-Hypervisor:
 - Konkrete Zuordnung von Nutzern, VMs, vFPGAs und deren Kopplungsart.
 - Interaktion mit den vFPGAs gemäß den Anforderungen aus Abschnitt 3.2.3.
- Umsetzung der Dienst-Modelle mit entsprechenden Sicherheitsstufen und Konfigurationsarten für die FPGAs innerhalb der lokalen Verwaltung:
 - RSaaS: Konfiguration des FPGAs über JTAG für vollen Zugriff (RSaaS) und Gewährleistung einer vollständigen Rekonfiguration des FPGAs.
 - RAaaS: Kapselung des Nutzers in vFPGAs beliebiger Größe, ohne weitere Nutzer auf demselben FPGA und Validierung der Positionen bei der vollständigen partiellen Rekonfiguration mittels eines Frameworks (ICAP).
 - BAaaS: Konfiguration über den Hypervisor mit Bitstreams für jede mögliche Position eines vFPGA auf dem physischen FPGA.
- Überwachung der Ressourcen, um Systemlast und Belegung der Knoten vollständig zu erfassen.
- Möglichkeit für ein Scheduling durch Migration zur Erhöhung der Auslastung und Bereitstellung einer elastischen Cloud.

3.4 Definition innerhalb der FPGA-Virtualisierung im Cloud-Kontext

Um für die folgenden Kapitel eine einheitliche Bezeichnung in Bezug auf die virtualisierten FPGAs zu etablieren, werden die notwendigen Begriffe definiert, um auf Basis der in diesem Kapitel durchgeföhrten Anforderungsanalyse die vFPGAs (siehe Definition 3.1) entsprechend ihres Lebenszyklus besser von einander abgrenzen zu können. Die Begriffe orientieren sich an denen der System-Virtualisierung nach [SN05b].

Definition 3.1: vFPGA Ein vFPGA ist ein virtueller FPGA, welcher sich innerhalb eines physischen FPGAs auf einen oder mehreren vFPGA-Slots (siehe Definition 3.2) befindet. Ein vFPGA wird vom Nutzer als eigenständige Ressource mit dynamischer Anzahl von Hardwareressourcen (Slices, LUTs, Register etc.) wahrgenommen.

Definition 3.2: vFPGA-Slots Ein vFPGA (siehe Definition 3.1) wird abgebildet auf einzelne physische Regionen mit fester Anzahl von Hardwareressourcen und somit fester Größe innerhalb des physischen FPGAs, welche als vFPGA-Slots bezeichnet werden.

Definition 3.3: vFPGA-Design Als vFPGA-Design wird der Hardwareentwurf/das Hardwaredesign eines Nutzers bezeichnet, welches ausgehend von einer Netzliste (RTL-Ebene) innerhalb eines vFPGAs (siehe Definition 3.1) mit seinen Frontend-Schnittstellen platziert und verdrahtet wird.

Definition 3.4: vFPGA-Image Ein partieller Bitstream, welcher die Basis für eine vFPGA-Instanz (siehe Definition 3.5) bildet, wird als vFPGA-Image bezeichnet, das bestimmten vFPGA-Slots (siehe Definition 3.2) zugeordnet wird.

Definition 3.5: vFPGA-Instanz Ein vFPGA-Image (siehe Definition 3.4) innerhalb eines vFPGAs, welches direkt einem Nutzer zugeordnet ist und nutzerspezifische Daten enthalten kann (Kontext), wird als vFPGA-Instanz bezeichnet und kann losgelöst von vFPGA-Slots (siehe Definition 3.2) betrachtet werden.

Neben den soeben vorgenommenen Definitionen, welche sich auf die konkreten vFPGAs und ihren Lebenszyklus beziehen, sind des Weiteren die unterschiedlichen Hypervisoren in dem Gesamtsystem gegeneinander abzugrenzen und zu definieren. Dabei bildet der Begriff Hypervisor, wie er in Abschnitt 2.2.1 als System zur Verwaltung und Zuteilung von Ressourcen des Gast- zum Host-System eingeführt wurde, die Basis für die folgenden Definitionen.

Definition 3.6: System-Hypervisor Der System-Hypervisor entspricht dem klassischen Hypervisor (VMM), welcher die Virtuellen Maschinen (VMs) innerhalb der System-Virtualisierung (siehe Abschnitt 2.2.3) auf dem Host-System bereitstellt.

Definition 3.7: RC2F Host-Hypervisor Die Verwaltungsstruktur für die vFPGAs (siehe Definition 3.1) auf deren Host-System wird als RC2F Host-Hypervisor, beziehungsweise Host-Hypervisor bezeichnet.

Definition 3.8: FPGA-Hypervisor Der FPGA-Hypervisor ist die Verwaltungsstruktur auf dem FPGA, welche die Zugriffe der vFPGAs (siehe Definition 3.1) innerhalb des physischen FPGAs überwacht.

3.5 Systementwurf und Abgrenzung

Die vorherigen Abschnitte haben die zugrundeliegende Systemarchitektur auf Ebene der Hardware und ebenso auf Ebene der Software aufgezeigt, welche für eine flexible Bereitstellung von virtualisierten FPGAs erforderlich ist. Die Anforderungen an die zu virtualisierenden FPGAs wurden in Abschnitt 3.2.3 aufgezeigt, die Anforderungen an das Cloud-System mit dem Schwerpunkt auf der Verwaltung der FPGA-Ressourcen wurden in Abschnitt 3.3.3 dargestellt. Um den zu entwickelnden Prototypen des Gesamtsystems modular aufbauen zu können, wird er in zwei eigenständige Teilkomponenten zerlegt, welche wiederum unabhängig voneinander eingesetzt werden können. Diese zwei wesentlichen Bestandteile oder Teilkomponenten des Gesamtsystems, welche im Verlauf dieser Arbeit entwickelt werden, sind:

- I. RC2F – Reconfigurable Common Computing Frame(work) mit den FPGA-spezifischen Komponenten zur Bereitstellung von virtuellen FPGAs (vFPGAs) und einer entsprechenden API zur Kommunikation zwischen VMs und vFPGAs und
- II. RC3E – Reconfigurable Common Cloud Computing Environment für die Verwaltung und Bereitstellung der virtualisierten FPGA-Ressourcen.

Der Schwerpunkt des RC2F liegt auf der Virtualisierung der FPGAs innerhalb einer Cloud-Architektur, mit dem Ziel, von der physischen Hardware durch Bereitstellung homogener vFPGAs abstrahieren zu können. Ein wichtiger Aspekt ist dabei die flexible Nutzung der FPGAs in unterschiedlichen Szenarien, ihre Interaktion mit einem Host-System und dessen VMs sowie unterschiedlichen Nutzern auf demselben physischen FPGA. Die Cloud-Verwaltung RC3E dagegen ist eine Verwaltungseinheit, deren Aufgabe darin besteht, die Komponenten zu identifizieren, welche für eine Verwaltung und Integration von FPGAs in einer flexiblen Cloud notwendig sind. Entsprechend beinhaltet RC3E lediglich die in Abschnitt 3.3.2 beschriebenen Teilkomponenten.

Bereits bestehende Systeme wie OpenNebula [Ope16a] oder OpenStack [Ope16d] werden nicht eingesetzt, da sie eine sehr hohe Komplexität aufweisen. Die Integration weiterer komplexer Komponenten in die bestehende Infrastruktur der Systeme, welche sehr spezifisch auf klassische System-VMs angepasst sind und andere Komponenten nicht vorsehen, wäre sehr zeitaufwendig und würde aufgrund der festen Strukturen der Systeme eine verminderte Flexibilität zur Folge haben. Bei anderen Arbeiten wie Byma et al. [Bym+14] oder Chen et al. [Che+14], welche auf OpenStack aufbauen und FPGAs wie VMs behandeln, besteht das Problem, dass die Integration nur oberflächlich erfolgt und eine Bereitstellung von flexiblen vFPGAs für unterschiedliche Szenarien und Diensten nicht vorgesehen ist.

In den folgenden beiden Kapiteln werden die Entwurfsräume für eine Virtualisierung und das resultierende RC2F in Kapitel 4 sowie für das Verwaltungssystem RC3E in Kapitel 5 vorgestellt, um schließlich darauf aufbauend mit einer entsprechenden prototypischen Implementierung in Kapitel 6 die Konzepte zu evaluieren.

4 Virtualisierung der FPGAs für den Einsatz in einer dynamischen Cloud-Architektur

Die Notwendigkeit und die entsprechenden Anforderungen an eine Virtualisierung von FPGAs für die Nutzung in einer Cloud-Architektur sowie für die Bereitstellung von vFPGAs (siehe Definition 3.1) wurden in Abschnitt 3.2 diskutiert. Ziel dieses Kapitels ist es zunächst, die Herausforderungen einer Virtualisierung für FPGAs in Abschnitt 4.1 zu erarbeiten. Darauf aufbauend werden die grundlegenden Konzepte für die Virtualisierung in Abschnitt 4.2 aufgezeigt. Der Entwurfsraum für die unterschiedlichen Komponenten wird in Abschnitt 4.3 dargestellt, um die möglichen Virtualisierungskonzepte abschätzen zu können. Abschnitt 4.4 stellt den darauf basierenden Entwurf der FPGA-Virtualisierung RC2F vor.

4.1 Grundbausteine einer Virtualisierung von FPGAs für den Einsatz in Cloud-Architekturen

Um eine Ausgangsbasis für die folgenden Schritte zu etablieren, ist zunächst die Frage der Bedeutung einer *Virtualisierung* im Kontext von rekonfigurierbarer Hardware zu klären. Dabei werden die für diese Arbeit relevanten Gemeinsamkeiten der unterschiedlichen Ansätze aus Abschnitt 2.4.1 diskutiert. Die vorgestellten Systeme, welche rekonfigurierbare Architekturen in Clouds einsetzen, unterscheiden sich dabei stark im grundsätzlichen Ansatz der *Virtualisierung*. Die Abstraktion von den physischen Schnittstellen bildet, gemäß der grundlegenden Definition der klassischen Virtualisierung aus Abschnitt 2.2.1, einen der wichtigsten Bestandteile und auch eine der Gemeinsamkeiten sämtlicher FPGA-Virtualisierungen. Ein weiterer zentraler Aspekt ist die Abstraktion des virtualisierten FPGAs von der Größe der physischen Hardware und weiterführend die Möglichkeit, diese unter mehreren Nutzern aufzuteilen. Daraus ergeben sich weitere Gesichtspunkte, wie der gemeinsame Zugriff unterschiedlicher Nutzer mit ihren VMs auf die gemeinsame physische Ressource FPGA. Zusammenfassend resultieren daraus folgende zu beachtende Punkte und Anforderungen an virtualisierte FPGAs, welche damit auch den Entwurfsraum abstecken:

- Abstraktion und gemeinsame Nutzung physischer Schnittstellen,
- Effiziente Auslastung der Ressourcen und Anpassung an die Bedürfnisse der Nutzer,
- Individuelle Größe der virtuellen FPGAs und unterschiedliche Nutzer auf derselben Hardware,
- Konfiguration der unterschiedlichen vFPGAs über einen FPGA-Hypervisor (siehe Definition 3.8),
- Infrastruktur und API zur Kommunikation zwischen VM auf dem Host und vFPGAs und
- Möglichkeit der Nutzung von FPGAs für sicherheitskritische Anwendungen.

Einen der wichtigsten Aspekte bei der Virtualisierung von Rechnersystemen stellt die Handhabung der ISA, wie in Abschnitt 2.2.1.2 erläutert, dar. Die Abstraktion von der Struktur des realen FPGAs eben-

so wie von den architektspezifischen Komponenten, in denen die Logik realisiert wird (wie LUTs, Slices und dem Verbindungsnetzwerk) kommt eine Virtualisierung mit unterschiedlichen ISAs gleich, welche jedoch mit Leistungseinbußen verbunden ist. Bei einer Auslagerung von Diensten auf die rekonfigurierbare Hardware hat aber die Leistungsfähigkeit eine größere Bedeutung als die Universalität der Architektur selbst. Wird eine Virtualisierung mit gleicher ISA für Host und Gastsystem – also identischer zugrundeliegender Architektur für Host- und Gast-FPGA – bevorzugt, resultiert dies in Einbußen in der Leistungsfähigkeit, wie in Abschnitt 2.4.1.5 gezeigt. In einer produktiven Cloud mit FPGAs (RAaaS und BAaaS) kann von gleichartiger homogener Hardware ausgegangen werden, sodass weder Zwischenschicht noch der Einsatz von Overlays unter dem Gesichtspunkt der Leistungssteigerung sinnvoll erscheinen. Die Anpassung der vFPGAs in ihrer Größe und ihren Ressourcen an die Bedürfnisse der Nutzer mit der Konsequenz, mehrere voneinander abgeschottete Nutzer auf derselben physischen Hardware zuzulassen, ist dabei von großer Bedeutung, um die Auslastung der Hardware zu erhöhen.

4.2 Konzept zur Virtualisierung eines physischen FPGAs

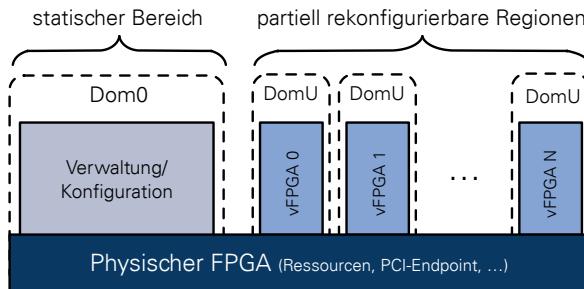
Die Möglichkeit, einen FPGA zu partitionieren und somit nebenläufig Nutzer auf demselben physischen FPGA zu realisieren, stellt die wesentliche Motivation der angestrebten Virtualisierung dar und wird im Folgenden mit Hinblick auf die Grundlagen zur Virtualisierung näher betrachtet und entsprechend konkretisiert.

4.2.1 Aspekte der Übertragung einer klassischen Virtualisierung auf FPGAs

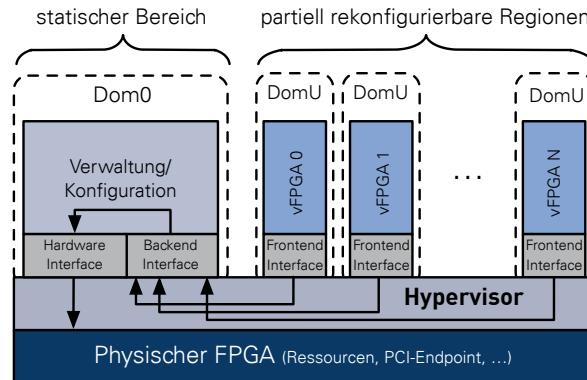
Die Eignung der klassischen Virtualisierungsarten für die angestrebte FPGA-Virtualisierung, wurde bereits in Abschnitt 2.2.5 und speziell in Tabelle 2.4 untersucht. Die Kapselung unterschiedlicher Nutzer auf derselben physischen Hardware und die Freiheit und Flexibilität, die innerhalb der vFPGAs bestehen sollen, lassen jedoch eine Zuordnung zu einer System-Virtualisierung beziehungsweise direkt in die *Bare-Metal-Virtualisierung* zu, welche dann detailliertere Unterscheidungen nach Abschnitt 2.2.3 erlaubt. Aufgrund des eingeschränkten oder komplett fehlenden Betriebssystems sowie der reinen Verwaltung der wesentlichen Zugriffe und der Konfiguration der vFPGAs durch ein festes Grunddesign, welches die Kapselung der Nutzer ausführt und sämtliche Privilegien hält (siehe Abbildung 2.2.3.3), weist die Virtualisierung eines FPGAs die größte Ähnlichkeit mit einer nativen Virtualisierung (Bare-Metal) mit VMM oder Hypervisor auf. Ein weiterer darauf aufsetzender VMM für eine Hosted-Virtualisierung wäre eine weitere Zwischenschicht in Form eines Nutzerdesigns.

Der Entwurf einer nativen System-Virtualisierung für FPGAs ist in Abbildung 4.1(a) dargestellt. Der FPGA wird unterteilt in einen dynamischen und einen statischen Bereich, welcher die Verwaltung des FPGAs sowie die dynamische partielle Rekonfiguration der einzelnen vFPGA-Regionen steuert und der privilegierten Domain (Dom0) gleichzusetzen ist. Die Bereiche, welche die vFPGAs selbst beinhalten, sind entsprechend unterprivilegierte Domains (DomU). Die einzelnen Regionen werden dabei direkt auf der physischen FPGA-Hardware ausgeführt.

Um von den vFPGAs aus und durch das statische Design der privilegierten Domain (Dom0) auf die physischen Schnittstellen des FPGAs zur Kommunikation mit der Außenwelt zugreifen zu können, ist eine Erweiterung mit einem FPGA-Hypervisor zur Administration der Zugriffe entsprechend der Privilegien erforderlich. Ein direkter wechselseitiger Zugriff der vFPGAs auf die physischen Schnittstellen stellt jedoch einen unnötigen Verwaltungs- und Ressourcenaufwand dar, sodass Paravirtualisierung (siehe



(a) Konzept einer FPGA-Virtualisierung analog zu System-VMs mit Einteilung in einen statischen Bereich (Dom0) zur Verwaltung und partiell rekonfigurierbare Bereiche für die vFPGAs (DomU).



(b) Erweiterung mit einem Zugriff auf gemeinsam genutzte Schnittstellen in Form einer Paravirtualisierung im statischen Bereich mit FPGA-Hypervisor und Backend-Interface, sowie Frontend-Interface als Schnittstelle zu den vFPGAs im dynamisch rekonfigurierbaren Bereich.

Abbildung 4.1: Einteilung des physischen FPGAs in einen statischen Bereich zur Verwaltung und mehrere vFPGAs mit partiell rekonfigurierbaren Regionen sowie Übertragung des Konzeptes einer paravirtualisierten System-VM auf FPGAs.

Abschnitt 2.2.3.3) der Hardwareinterfaces innerhalb eines FPGA-Hypervisors eine deutlich effizientere Lösung bietet. Abbildung 4.1(b) zeigt die entsprechende Erweiterung von Abbildung 4.1(a) mit einem zusätzlichen FPGA-Hypervisor, welcher die Kommunikationskanäle zwischen den Frontend-Interfaces innerhalb der vFPGAs und dem Backend-Interface, welches direkten Zugriff auf das physische Interface ermöglicht, gestattet. Die physischen Interfaces können hierbei serielle Protokolle mit hoher Bandbreite wie PCIe oder QPI, aber auch Netzwerkverbindungen wie Ethernet sein.

Auf diese Weise kann über den FPGA ein unmittelbarer Zugang zum Cloud-System geschaffen werden. Der FPGA-Hypervisor befindet sich, wie die Verwaltung in Dom0, im statischen Bereich des FPGAs und kann entsprechend nicht dynamisch rekonfiguriert werden, solange noch vFPGA-Images (siehe Definition 3.4), beziehungsweise vFPGA-Instanzen (siehe Definition 3.5), auf den vFPGAs enthalten sind. Das resultierende Virtualisierungskonzept kann nach Eingrenzung des Entwurfsraumes und in Anlehnung an die Begriffe aus der Betriebssystem-Virtualisierung als

Native Typ 1 System-/Bare-Metal-Virtualisierung bei gleicher Architektur (ISA) mit Zugriff auf externe Geräte mittels Paravirtualisierung innerhalb des FPGA-Hypervisors (VMM)

bezeichnet werden. Für eine möglichst vollständige System-Virtualisierung von FPGAs analog zu den VMs für klassische Betriebssysteme ergeben sich eine Reihe weiterer Anforderungen, wie die Rege-

mentierung der FPGA-Ressourcen, Datenraten und die Speichernutzung, um die Hardware FPGA unterschiedlichen Nutzern gleichzeitig bereitstellen zu können und diese effizient auszulasten. Das Anhalten und Verschieben eines Hardwaredesigns, sowie die horizontale und vertikale Skalierung der Nutzerdesigns für die vFPGAs sind ebenso zu betrachtende Komponenten, für welche im Folgenden der Entwurfsraum aufgezeigt und eingegrenzt wird.

4.2.2 Mehrbenutzerbetrieb auf rekonfigurierbarer Hardware

Ein wesentliches Ziel der FPGA-Virtualisierung besteht darin, unterschiedliche Nutzer auf derselben physischen Hardware zuzulassen. Dieser Aspekt kann auf mehrere Arten umgesetzt werden, wobei die beiden wesentlichen Unterscheidungen bereits in Abschnitt 2.4.1.3 aufgezeigt wurden. Ist lediglich der Zugang zu demselben physischen FPGA durch unterschiedliche Nutzer das Ziel, ist eine Nutzung des vollständigen FPGAs mit Kontextwechsel oder Arbeitspaketen möglich. Bei dieser Form der Virtualisierung, wie sie Wang et al. [WBP13] oder Gonzalez et al. in [Gon+12] einsetzen, werden die Anfragen sequenziell abgearbeitet. Bei längeren Arbeitspaketen wird nach einer bestimmten Zeit ein vollständiger, für den Nutzer intransparenter Kontextwechsel wie bei Happe et al. in [HTK15] durchgeführt. Ein entsprechendes Pausieren des Arbeitspaketes, wie in Abbildung 4.2(a) dargestellt, erfordert somit einen Speicher für Kontexte und Bitstreams.

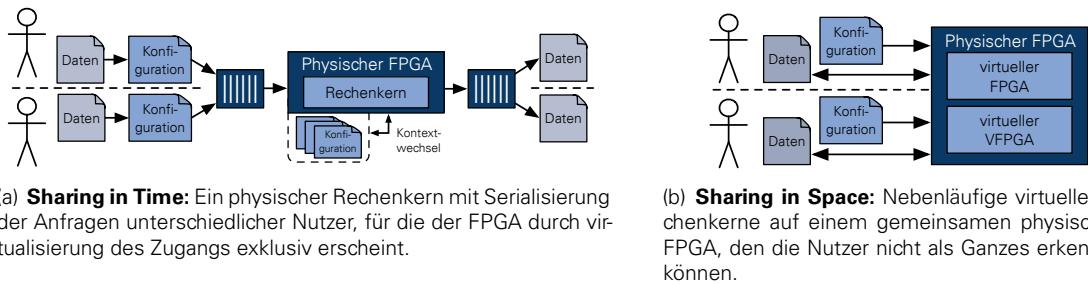


Abbildung 4.2: Unterschiedliche Ansätze zur Virtualisierung rekonfigurierbarer Rechenressourcen.

Eine vollständige Auslastung der Ressourcen eines FPGAs wird durch dieses Verfahren allerdings nicht erreicht. Um durch eine Virtualisierung die Erhöhung der Auslastung durch an die Größe der Nutzerentwürfe angepasste vFPGAs und eine annähernde Echtzeitfähigkeit zu ermöglichen, ist eine Aufteilung des physischen FPGAs wie in Abbildung 4.2(b) erforderlich. Eine solche erfolgt vom grundlegenden Konzept her auch in den in Abschnitt 2.4.1.3 vorgestellten Arbeiten von Fahmy et al. [FVS15], Byma et al. [Bym+14], oder Chen et al. [Che+14]. Eine Partitionierung stellt demnach einen wesentlichen Schritt zur Virtualisierung des FPGAs für eine hohe Ressourcenauslastung und einen flexiblen Einsatz in einer Cloud dar.

Im Gegensatz zu den genannten Arbeiten ist hier der gewählte Ansatz an die etablierten Techniken der Virtualisierung angelehnt. Damit wird eine System-Virtualisierung erreicht, welche einer Partitionierung des physischen FPGAs für unterschiedliche Nutzer, gekapselt in festen, dynamisch partiell rekonfigurierbaren Regionen, entspricht.

4.3 Entwurfsraum einer FPGA-Virtualisierung

Im Folgenden werden die einzelnen in Abschnitt 4.1 und Abschnitt 4.2 skizzierten Varianten innerhalb des Entwurfsraumes weiter ausgearbeitet. Für weitere wesentliche Komponenten, welche sich dabei ergeben haben, wie Schnittstellen, Kommunikationskanäle, Skalierung, Zustandsverwaltung und die Aufgaben des FPGA-Hypervisors, werden ebenso Entwurfsräume aufgezeigt.

4.3.1 Größe, Position und Flexibilität der vFPGAs

Die Anzahl und somit auch die Größe der vFPGAs und daher auch deren Ressourcen sind abhängig von den in ihrer Position festen Schnittstellen und den physischen Regionen, welche in ihrer Zahl theoretisch beliebig, aber aufgrund der physischen hersteller-spezifischen Architektur des FPGAs faktisch begrenzt sind. Die einfachste Möglichkeit der Aufteilung der vFPGAs besteht hierbei in einer festen Anzahl und festen Größe der rekonfigurierbaren vFPGA-Regionen, welche nach Definition 3.2 als *vFPGA-Slots* bezeichnet werden. Diese sind durch die physische Position der Frontends für die vFPGAs fest vorgegeben. Die vFPGAs sind demzufolge virtuelle Instanzen, welche an definierten physischen Positionen beziehungsweise Regionen innerhalb der vFPGA-Slots instanziert werden. Sämtliche vFPGA-Slots haben die gleiche Größe und sind homogen aufgebaut, um vFPGAs beliebig platzieren zu können.

Bei einem größeren Ressourcenbedarf des Nutzers ist demnach eine entsprechende Aufteilung des Hardwaredesigns auf mehrere vFPGAs erforderlich, welche dann ausschließlich über ihre festen Frontend-Interfaces miteinander kommunizieren können. Die Unterteilung eines Hardwaredesigns in unterschiedliche vFPGAs resultiert unter Umständen in Einbußen in der Leistungsfähigkeit durch die Partitionierung des vFPGA-Designs (siehe Definition 3.3). Für einen Einsatz in der Cloud bildet daher eine zusätzliche vertikale Skalierung mit vFPGAs unterschiedlicher Größe über mehrere zusammengefasste vFPGA-Slots eine Möglichkeit, um die Flexibilität zu erhöhen. Die mögliche Größe eines vFPGAs entspricht dabei einem ganzzahligen Vielfachen eines homogenen vFPGA-Slots, um die Anzahl möglicher Varianten zu begrenzen. Eine Skalierung über die Grenzen des physischen FPGAs hinaus erfordert allerdings bei diesem Vorgehen dennoch eine Partitionierung des Hardwaredesigns, welche nicht umgangen werden kann. Die gleiche Herausforderung besteht aber ebenso bei klassischen VMs. Abbildung 3.9 zeigt anhand eines Beispiels mit vier Frontend-Interfaces und entsprechenden vFPGA-Slots, wie unterschiedliche vFPGAs bereitgestellt werden können. Im Beispiel werden ein kleiner vFPGA über einen einzelnen Slot und ein großer vFPGA, welcher sich über drei vFPGA-Slots erstreckt, instanziert. Die Basis der Virtualisierung bildet weiterhin die in Abschnitt 4.2.1 vorgestellte Paravirtualisierung. Eine Nutzung der zusätzlichen Frontends innerhalb eines vFPGAs über mehrere Slots fällt hierbei in den Verantwortungsbereich des Nutzers.

Das Problem dieses Ansatzes besteht lediglich darin, dass die Frontends im statischen Teil des FPGAs angesiedelt sind und dementsprechend ihre Anzahl nicht geändert werden kann, ohne den kompletten physischen FPGA neu zu konfigurieren und die Nutzerdesigns völlig auszulagern. Eine Möglichkeit, diese Situation zu umgehen, ist eine baumartige, dynamisch rekonfigurierbare Struktur aus erweiterbaren Frontends. Eine solche würde allerdings zu unterschiedlichen Kommunikationslatenzen der Schnittstellen führen und die Komplexität des Entwurfes aufgrund von verschiedenen Positionen der Frontends im Baum deutlich steigern. Da davon ausgegangen werden kann, dass die erforderliche Logik für Frontends verhältnismäßig klein sein wird, ist somit eine festgeschriebene Zahl an Frontends, welche sich nach der konkreten FPGA-Architektur richtet, die geeignetste Lösung.

Die Bereitstellung unterschiedlicher vFPGAs wird bei modernen FPGAs über die partielle dynamische Rekonfiguration zur Laufzeit erreicht. Dazu ist lediglich ein festes statisches Hardwaredesign mit fester Position der Frontend-Interfaces erforderlich. Die vFPGA-Images werden auf Basis dessen in den für sie vorgesehenen vFPGA-Slots über einen modifizierten Entwurfsablauf bis hin zum partiellen Bitstream erzeugt. Der Hypervisor auf dem Host-System hat die Aufgabe die vFPGA-Slots zu verwalten, um Überschneidungen zu vermeiden.

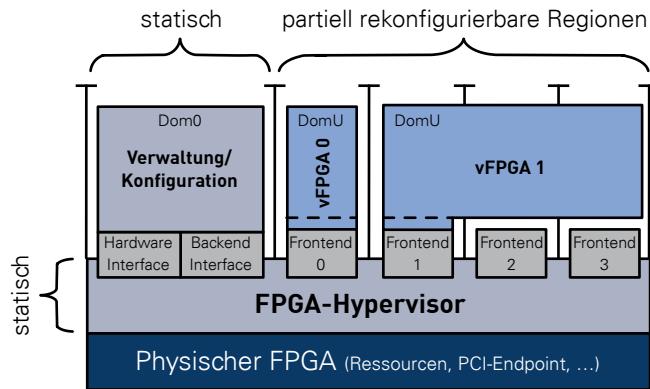


Abbildung 4.3: Paravirtualisierung mit unterschiedlicher Größe der vFPGAs. Beispielsystem mit vier Frontends und vier vFPGA-Slots, belegt mit zwei vFPGAs unterschiedlicher Größe.

Mehrere der zuvor in Abschnitt 2.4 beschriebenen Ansätze setzen eine Einteilung der vFPGAs in eine feste Gitterstruktur voraus, wie Fahmy et al. [FVS15] oder Dondo Gazzano et al. [Don+15], sodass die Größe der vFPGAs immer ein ganzzahliges Vielfaches der Größe eines einzelnen vFPGA-Slots darstellt. Dieses Vorgehen führt nicht zwangsläufig zu einer effizienten Auslastung des physischen FPGAs, ist aber die zu bevorzugende Variante, um den Aufwand der Platzierung beziehungsweise die notwendigen Varianten an vFPGA-Images für unterschiedliche Regionen auf dem physischen FPGA zu minimieren. Der in Abbildung 4.3 skizzierte Ansatz stellt demnach einen ausgewogenen Kompromiss aus Flexibilität und Auslastung von sowohl vFPGA als auch physischem FPGA dar. Um ein vFPGA-Design für alle beliebigen vFPGA-Slots bereitstellen zu können, ist entweder die einmalige Generierung aller Varianten oder aber eine Modifikation der einsatzbereiten Bitstreams wie in den Arbeiten [KBT08; Oom+15; RFG16] erforderlich. Die Modifikation eines bereits platzierten Hardwaredesigns ist aufgrund der in Abschnitt 2.1.2.2 vorgestellten inhomogenen FPGA-Architektur ein komplexes Verfahren, welches die Hersteller von FPGAs nicht oder nur teilweise unterstützen.

4.3.2 Anordnung, Schnittstellen und Architektur der vFPGAs

Neben der Virtualisierung selbst stellt auch die Architektur und Struktur der vFPGAs einen entscheidenden Entwurfsraum dar. Bisher wurde in Abschnitt 4.2.1 lediglich die Notwendigkeit eines FPGA-Hypervisors zur Kapselung der vFPGAs voneinander und zum Zugriff auf die gemeinsamen Hardwareschnittstellen erarbeitet. Im Folgenden werden daher die vFPGAs selbst sowie ihre Schnittstellen betrachtet, ebenso wie deren Anordnung auf dem FPGA. Abbildung 4.4 zeigt exemplarisch verschiedene Varianten der Anordnung von vFPGAs, ausgehend von obligatorischer Nutzung von vFPGAs in den Servicemodellen RAaaS und BAaaS, welche in Abschnitt 3.1.2 eingeführt wurden.

Den Ausgangspunkt für die Konzeption von Anordnung und Architektur der vFPGAs bildet das in Abbildung 3.3(c) in Abschnitt 3.1.2 erläuterte Konzept unterschiedlich gekapselter vFPGAs, welche über einen Switch auf die gemeinsamen Geräte zugreifen können. Die erste Modifikation erfolgt, wie im vorherigen

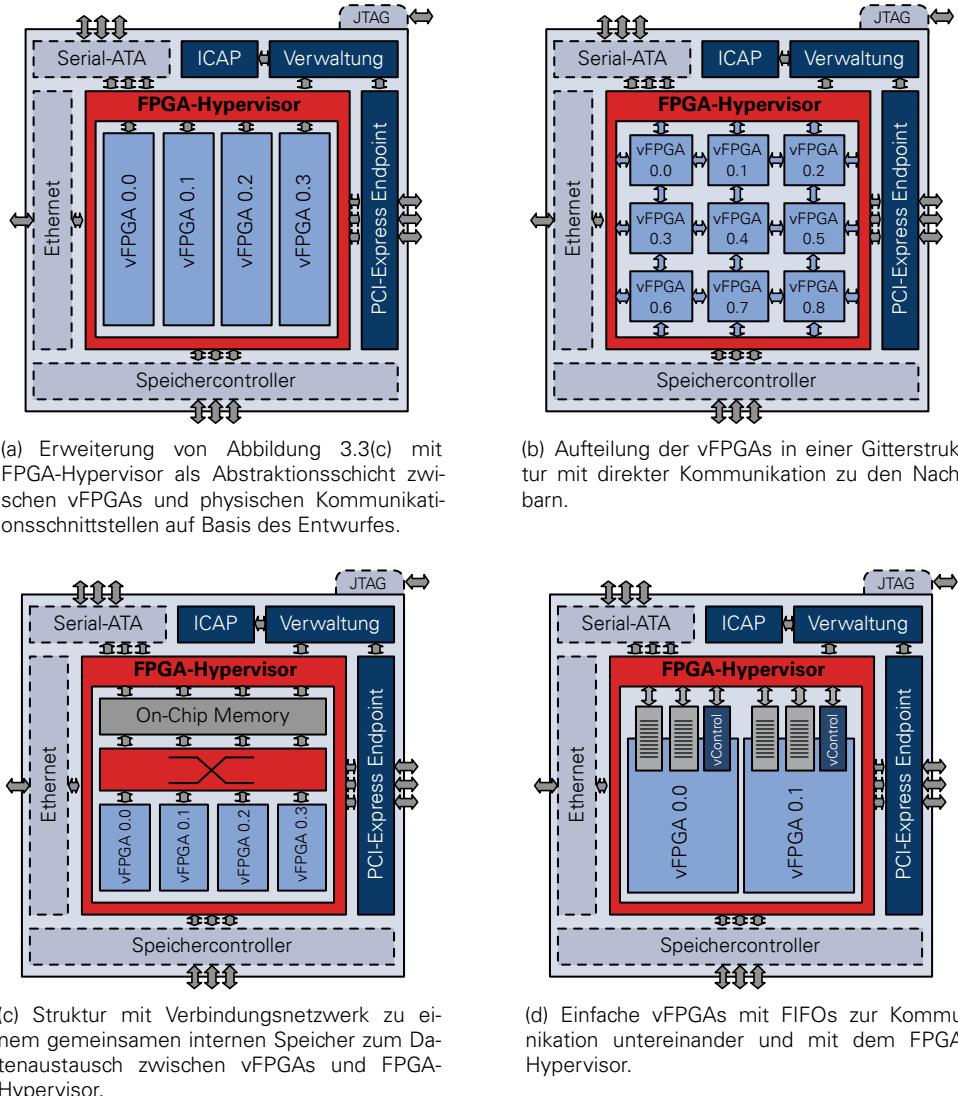


Abbildung 4.4: Entwurfsraum für die Architektur der vFPGAs mit FPGA-Hypervisor, Kommunikationsinterfaces und Anordnung der Schnittstellen.

Abschnitt beschrieben, als Erweiterung des einfachen Switch, welcher die Geräte und vFPGAs über ein Crossbar koppelt und zu einem FPGA-Hypervisor, wie in Abbildung 4.4(a) dargestellt, zusammenführt. Dieser übernimmt auch die Aufgaben der lokalen Ressourcenverwaltung sowie der Paravirtualisierung als Zugriffskonzept für die Hardwareschnittstellen.

Wie in Abbildung 4.4(b) skizziert, kann neben der linearen Anordnung der vFPGAs, wie sie bisher implizit angenommen wurde, beispielsweise auch eine Einteilung in eine Gitterstruktur, ähnlich wie in der Arbeit von Nguyen et al. [NK14], erfolgen. Die vFPGAs haben dann die Möglichkeit, mit ihren Nachbarn zu kommunizieren. Ein Algorithmus kann auf diese Weise parallelisiert werden, jedoch wird dadurch die Dynamik nur vordergründig vergrößert, da ein vFPGA eines Nutzers einen völlig eigenständigen gekapselten Bereich darstellen soll. Bei diesem Ansatz überträgt der Nutzer aber seinen Algorithmus auf mehrere vFPGAs im Gitter, welche dann nicht ideal an die erforderlichen Ressourcen angepasst werden können. Wenn die Möglichkeit besteht, aus Gründen der Effizienz mehrere vFPGAs zu einem größeren vFPGA zusammenzuschalten, ist es nicht notwendig, mit den benachbarten vFPGAs, welche

anderen Nutzern zugeordnet sind, zu kommunizieren. Eine anwendungsspezifische Gitterstruktur ist als Infrastruktur innerhalb eines FPGAs durchaus sinnvoll, aber aus Gründen der Effizienz nicht für eine Virtualisierung zum Einsatz in der Cloud tragbar. Eine weitere Herausforderung besteht in der Kapselung der Nutzer um eine entsprechende Sicherheit von Nutzerdaten und dem Design selbst zu gewährleisten.

Eine vielversprechende Möglichkeit bietet ebenso eine an OpenCL [Sto10] angelehnte Struktur für die vFPGAs, wie sie in Abbildung 4.4(c) aufgezeigt und bereits von FPGA-Herstellern umgesetzt wird [Alt13; Xil17c]. Die Kommunikation mit anderen vFPGAs erfolgt hier über einen gemeinsamen Speicher. Der Datenaustausch kann ebenfalls über diesen Speicher erfolgen, und die dynamische Vergrößerung eines vFPGAs ist ebenso möglich wie eine Kapselung der einzelnen Nutzer, wobei der Speicher entsprechend segmentiert werden muss. Ein wesentlicher Vorteil dieses Ansatzes besteht in einer einfachen Generierung von Nutzerdesigns über HLS, welche für einen Cloud-Einsatz durchaus sinnvoll ist. Ein Nachteil dieses Designs ist dessen fehlende Flexibilität aufgrund der notwendigen Kommunikation über den gemeinsamen Speicher, insbesondere bei Anwendungen, welche ein Streaming von Daten erfordern. Prinzipiell sollte allerdings die Möglichkeit bestehen, auf vFPGAs eine OpenCL-Struktur umzusetzen, um den Cloud-Nutzern die Möglichkeit zu geben, Entwicklungswerzeuge wie OpenCL einzusetzen.

Um eine größtmögliche Flexibilität in Hinblick auf die Szenarien aus Abschnitt 3.2.1 zu gewährleisten, ergibt sich der in Abbildung 4.4(d) aufgezeigte Entwurf. Die vFPGA-Slots sind linear, wie auch schon in Abbildung 4.3, angeordnet. Die Anzahl der Frontend-Interfaces ist hierbei variabel, und die Kommunikation mit den Hardwareschnittstellen und zwischen den vFPGAs erfolgt über mindestens zwei unabhängige FIFO-Interfaces zum Netzwerk innerhalb des FPGA-Hypervisors. Die nebenläufigen Datenkanäle sind zueinander abgegrenzt, um eine Sicherheit der Nutzerdaten zu gewährleisten. Die Verwaltung der vFPGAs kann gesondert über eine Kontrolleinheit erfolgen. Die vFPGAs sind dabei, wie gefordert, in ihrer Größe beliebig. Ein solches Design kann wesentlich einfacher ausgelastet werden als die Gitterstruktur und gestattet dem Nutzer deutlich größere Freiheiten als eine feste OpenCL-Struktur, da dieser sämtliche Ressourcen innerhalb seines vFPGAs nach Belieben einsetzen kann. Des Weiteren ist eine einfachere Realisierung der geforderten Schnittstellen möglich, wie im Folgenden in Abschnitt 4.3.3 gezeigt wird.

4.3.3 Kommunikationskanäle zwischen den vFPGAs, Host und Netzwerk

Hinsichtlich der erforderlichen Datenkanäle ergeben sich unterschiedliche Anforderungen und Kommunikationsschnittstellen innerhalb des FPGA-Hypervisors als Knotenpunkt zwischen vFPGA und dem restlichen Cloud-System. Entsprechend der Integration der FPGAs in die in Abschnitt 3.3.1 vorgestellte Cloud-Architektur sind dabei folgende Kommunikationskanäle innerhalb eines Rechen-Knotens mit FPGA-Beschleuniger erforderlich:

- I. Host-System ↔ vFPGA(s)
- II. Host-VM ↔ Host-Dom0
- III. Netzwerk ↔ vFPGA ↔ Host-VM
- IV. vFPGA ↔ vFPGA
- V. (Host-VM ↔) FPGA ↔ On-Board DDR-RAM

Durch die unterschiedlichen Kommunikationspartner, wie andere vFPGAs oder eine VM auf dem Host-System, ist eine nachrichten- oder paketorientierte Kommunikation oder das Umschalten der Verbindungen über einen Multiplexer erforderlich. An dieser Stelle wird zunächst lediglich ein Knoten mit VMs

und direkt gekoppelten vFPGAs betrachtet. Übersteigt die Zahl der vFPGAs die physischen Kapazitäten im Rechenknoten, ist die Kommunikation auf das Netzwerk auszulagern. Die resultierenden Einbußen in Datenrate und Latenz müssen daher dem Nutzer bekannt sein, damit dieser die Anwendung daran anpassen kann. Im Folgenden werden die einzelnen Kommunikationskanäle näher betrachtet und analysiert.

I. Host-System ↔ vFPGA(s): Der Kommunikationskanal mit der höchsten Priorität, welcher bei einer

Vielzahl von Anwendungen zum Einsatz kommt, verläuft zwischen dem Host-System und den vFPGAs auf dem entsprechend eng gekoppelten physischen FPGA. Da hierbei Latenz und Bandbreite von großer Bedeutung sind, stellt ein dediziertes Interface zwischen VM und den vFPGAs, wie es bei einer Vielzahl von verwandten Arbeiten wie [Che+14; FVS15; V+11] eingesetzt wird, einen vielversprechenden Ausgangspunkt dar. Wird ein dedizierter Kanal zwischen den beiden Kommunikationsendpunkten eingesetzt, kann eine Kommunikation mit einem kontinuierlichen Datenstrom erfolgen, um eine maximale Datenrate zu erzielen. Jeder vFPGA verfügt über einen eigenen dedizierten Kanal zum Host, der somit auf alle, ihm zugewiesenen vFPGAs gleichermaßen zugreifen kann. Die Zahl der vom Host über diesen Kanal erreichbaren vFPGAs muss dem Nutzer und der Ressourcenverwaltung trotz Virtualisierung bekannt sein.

II. Host-VM ↔ Host-Dom0: Einen weiteren Aspekt stellt die Kommunikation zwischen unterschiedlichen VMs auf demselben System und der gemeinsam genutzten physischen Hardware dar, denn

durch diese wird eine Bereitstellung von VM und einem bestimmten vFPGA als virtuelles System ermöglicht. Der Zugriff auf den Hardwarebeschleuniger, welcher eine hohe Datenrate zum Host erfordert, wird in der Regel durch vollständiges Durchreichen des PCIe-Gerätes (PCI-Passthrough) an eine bestimmte VM erreicht. Verbergen sich durch die Virtualisierung des Gerätes hinter dem physischen Gerät mehrere virtuelle Geräte, ist ein solches Durchreichen dagegen nicht möglich. Eine Lösung dafür bietet die Inter-Domain-Kommunikation zwischen den unterprivilegierten VMs und der privilegierten Verwaltungs-Domain, welche Zugriff auf die Geräte des Host-Systems ermöglicht und entsprechend deren Virtualisierung übernimmt.

Die Herausforderung, eine effiziente Inter-Domain-Kommunikation zu entwickeln, ergibt sich nicht allein für die FPGA-Virtualisierung. Einige Arbeiten im Bereich der Virtualisierung mittels des Host-Hypervisors Xen [Bar+03], welche aber einen geringen Datendurchsatz erzielen, da sie auf der Ebene der Netzwerkvirtualisierung arbeiten, sind [Hua+07; Kim+08; WWG08; Zha+07]. Ansätze für die Inter-Domain-Kommunikation auf Basis eines gemeinsamen Speicherbereiches für GPGPU-Virtualisierung auf Basis von Xen sind ebenso in der Literatur zu finden [Gup+09; Shi+12] wie für einen FPGA-Beschleuniger in der Arbeit von Wang et al. [WBP13].

Ein andere Möglichkeit zur Bereitstellung der Hardware ohne eine Inter-Domain-Kommunikation besteht in einer direkten Virtualisierung der Hardwareschnittstelle, wie beispielsweise die Single Root I/O-Virtualisierung für PCIe 3.0-Geräte [San+14a; San+14b]. Hierbei wird mehreren virtualisierten Gastbetriebssystemen ein nativer Zugriff auf ein gemeinsam genutztes PCIe-Gerät gewährt.

III. Netzwerk ↔ vFPGA (↔ Host-VM): Für einen direkten Zugang zu einer VM über den FPGA exis-

tiert für das in dieser Arbeit entwickelte System neben der Kommunikation zwischen vFPGA und VM auf dem Host-System ein weiterer Kanal zum Netzwerk, welcher gleichzeitig genutzt werden kann. Dieses Design bietet beispielsweise die Möglichkeit, eintreffende Netzwerkpakete direkt zu analysieren und an den Host weiterzuleiten. Die beiden hierfür notwendigen Schnittstellen sind in Abbildung 3.8 an den FPGAs dargestellt. Der Datenstrom zwischen vFPGA und VM muss dabei entsprechend um Paketinformationen ergänzt oder anwendungsspezifisch im Nutzerdesign auf

dem vFPGA angepasst werden. Der Zugriff von den vFPGAs auf die gemeinsame Netzwerkschnittstelle auf dem physischen FPGA erfolgt über eine entsprechende Paravirtualisierung innerhalb des FPGA-Hypervisors.

IV. vFPGA ↔ vFPGA: Da bei der Kommunikation zwischen vFPGAs auf unterschiedlichen FPGAs zwangsläufig das Hypervisor-Netzwerk auf dem FPGA involviert ist, kann nicht vollständig auf Pakete verzichtet werden. Entsprechend sind auch für die Kommunikation zwischen vFPGAs auf demselben physischen FPGA Pakete notwendig. Um die Zahl der Kommunikationskanäle möglichst gering zu halten, stellt die Nutzung der Schnittstelle für das Netzwerk die optimale Lösung dar, und es ergeben sich die beiden in Abbildung 4.4(d) gezeigten FIFO-Interfaces zwischen den vFPGAs und dem FPGA-Hypervisor. Hierbei müssen die Informationen zur physischen Position des virtuellen Kommunikationspartners über ein Verzeichnis bereitgestellt werden. Dieses wird benötigt, um entscheiden zu können, ob die Pakete über das externe Cloud-Netzwerk geleitet werden oder lokal vom FPGA-Hypervisor an einen anderen vFPGA gesendet werden müssen. Sämtliche Nachrichten müssen dabei mit virtuellen IP-Adressen für Empfänger und Sender versehen werden. Auf diese Weise kann das sofortige Weiterleiten von über das Netzwerk eingehenden Paketen aufgrund der dedizierten Datenkanäle von den vFPGAs zu den Host-VMs ohne weitere Verzögerung erfolgen.

V. (Host-VM ↔) FPGA ↔ On-Board DDR-RAM: Ein weiterer Datenkanal ist für die Kommunikation mit dem On-Board DDR-RAM erforderlich. Die Anwendungen benötigen einen direkten Zugang zum Speicher, um Daten aus den anderen Kanälen an diesen weiterzuleiten und den Hardwaredesigns der vFPGAs einen schnellen und vor allem direkten Zugang zum Speichercontroller zu gewährleisten.

4.3.4 Skalierung der vFPGAs auf dem physischen FPGA und darüber hinaus

Gemäß der Ausführungen in Abschnitt 3.2.2 soll neben unterschiedlich großen vFPGAs, wie sie in Abschnitt 4.3.1 diskutiert wurden, eine zusätzliche Skalierung der vFPGAs über die Grenzen eines physischen FPGAs hinaus ermöglicht werden, um einen virtuellen FPGA-Clusters zu realisieren. Die Vorgehensweise entspricht dabei im Cloud-Kontext einer vertikalen Skalierung. Ein Hardwaredesign muss in diesem Fall auf unterschiedliche FPGAs automatisiert partitioniert werden. Dies erfolgt, indem interne Kommunikationskanäle angelegt werden, welche dann auf üblicherweise mehrere parallele serielle Kanäle abgebildet werden. Ein derartiger Ansatz ist beispielsweise Gegenstand von Arbeiten im Bereich des Multi-FPGA-Prototyping wie der von Tang et al. [TMT14].

Da für den angestrebten Einsatzzweck im Rechenzentrum Netzwerkverbindungen eingesetzt werden, ist die vertikale Skalierung über mehrere physische FPGAs mit Einbußen in der Verarbeitungsgeschwindigkeit verbunden. Für ein Hardwaredesign, welches entsprechende Mengen an FPGA-Ressourcen benötigt, ist eine manuelle Partitionierung auf einzelne vFPGAs notwendig, welche Nachrichten direkt über das Netzwerk austauschen können. In diesem Modell kann ein vFPGA dementsprechend nur die Nutzerressourcen eines physischen FPGAs umfassen, wie in Abschnitt 4.3.1 beschrieben. Die vFPGAs kommunizieren, wenn sie sich auf demselben physischen FPGA befinden, direkt oder, wenn sie sich in unterschiedlichen Knoten befinden, über das Cloud-Netzwerk. Das grundsätzliche Problem besteht in der Partitionierung des Hardwaredesigns auf die einzelnen vFPGAs. Hierbei handelt es sich um eine klassische horizontale Skalierung, sodass eine nachrichtengekoppelte Programmierung wie mittels Message Passing Interface (MPI) [Sni98] für die Kommunikation der vFPGAs adaptiert werden kann,

wie auch in [Sal+10] beschrieben wird. Die Verteilung von Aufgaben auf einen FPGA-Cluster ist eine Problemstellung, die unter anderem auch in der Arbeit von Kalms et al. [KG16] untersucht wurde.

4.3.5 Zustände und Kontextmigration der vFPGAs

Entscheidend für die Etablierung von virtuellen FPGAs analog zur System-Virtualisierung ist die Fähigkeit des vFPGA, unterschiedliche Zustände annehmen zu können. Das Starten eines vFPGAs entspricht dabei einer Allokation der vFPGA-Slots über den Host-Hypervisor und der anschließenden partiellen Rekonfiguration des korrespondierenden Bereiches auf dem physischen FPGA. Ein Anhalten eines vFPGAs kann ein Stoppen des Taktes innerhalb der dazugehörigen vFPGA-Slots bedeuten. Das Herunterfahren eines vFPGAs wäre ein Entfernen des partiellen Hardwaredesigns innerhalb aller dazugehörigen vFPGA-Slots und des entsprechenden Eintrages innerhalb des Host-Hypervisors.

Insbesondere die Möglichkeit der Migration eines vFPGA ist für einen effizienten Einsatz in der Cloud nach den Anforderungen aus Abschnitt 3.2.3 essenziell. Dazu ist das vollständige Auslesen und Wiederherstellen einen Kontextes mit sämtlichen Speicher- und Registerinhalten (*context-save-and-restore* Mechanismus) des Hardwaredesigns erforderlich. Für eine Migration zwischen unterschiedlichen physischen FPGAs existieren die in Abschnitt 2.2.3.4 aufgezeigten Optionen, wobei eine kalte Migration mit dem vollständigen Herunterfahren des vFPGAs die einfachste Variante darstellt. Dabei wird vom Host-Hypervisor das dem vFPGA zugehörige Nutzerprogramm beendet und dessen aktueller Zustand gesichert. Dem vFPGA wird über den FPGA-Hypervisor ebenfalls signalisiert, dass dieser die Berechnungen stoppen und entsprechende Daten aus Registern und Speichern an den Host-Hypervisor übermitteln muss, wo die Daten ebenfalls gespeichert werden. Der Nutzer muss entsprechende Signale zum Stoppen des vFPGAs in seinem Design und im Host-Programm berücksichtigen und die notwendigen Vorkehrungen zum Ausgeben der Register- und Speicherinhalte treffen. Danach wird die Berechnung auf einem anderen vFPGA weitergeführt, nachdem dieser mit den entsprechenden Register- und Speicherinhalten initialisiert worden ist.

Bei der warmen Migration hingegen muss das Hardwaredesign mit seinen Schnittstellen angehalten oder direkt bei laufendem Betrieb unterbrochen werden. Aufgrund der höheren Geschwindigkeit eignet sich dieses Verfahren ideal für die angestrebte Virtualisierung. Das Auslesen des eingefrorenen Zustandes kann über die Infrastruktur zur Rekonfiguration erfolgen und direkt an einen anderen vFPGA übertragen werden, welcher sich auf einem anderen physischen FPGA befinden kann. Die Migration des unter Umständen genutzten DDR-Speichers des FPGA-Boards ist ebenso erforderlich. Um das Auslesen des Kontextes direkt aus dem FPGA zu ermöglichen, kann ergänzende Logik innerhalb des Entwurfes eingefügt werden, wie beispielsweise zusätzliche Multiplexer und ein aufwändiges Bussystem. Ebenso möglich ist die Nutzung der Carry-Chain Leitungen wie in den Arbeiten von Koch et al. [KHT07] oder Jovanovic et al. [Jov+07] ausgeführt. Derartige Systeme erreichen zwar eine hohe Geschwindigkeit, erfordern aber Modifikationen der Designs auf der RTL- oder der Netzlisten-Ebene und ebenso zusätzliche Hardwareressourcen. Somit würde die Erzeugung des vFPGAs einen Zusatzaufwand für den Nutzer oder eine automatisierte Modifikation mit dem Risiko veränderter Laufzeiten bedeuten. Des Weiteren benötigen die zusätzlichen Komponenten im FPGA-Design weitere Ressourcen.

Um dieses Probleme zu umgehen, kann die Möglichkeit genutzt werden, die aktuelle Konfiguration eines FPGAs mit sämtlichen Registern auszulesen. Dazu können die entsprechenden Bits innerhalb des Bitstreams identifiziert werden, um diesen über die bestehende Infrastruktur zur Konfiguration des FPGAs partiell auszulesen (und wiederherzustellen). Dabei ist keine Zusatzlogik erforderlich und der Kontextwechsel kann an beliebigen Stellen der Berechnung erfolgen. Das grundsätzliche Verfahren wur-

de bereits in Abschnitt 2.4.1.4 Morales-Villanueva et al. [MG13] für einen Xilinx Virtex-5 untersucht und schon in ReconOS von Happe et al. [HTK15] als *Hardware Task Preemption* eingesetzt, um auf einem Virtex-6 die Zustände aller Register und Speicherinhalte zu erfassen und zu einem späteren Zeitpunkt wiederherzustellen, um so Multitasking mit Hardware-Threads zu ermöglichen. Entsprechend erscheint dieses Vorgehen auch für die hier dargestellte Virtualisierung als ein vielversprechender Ansatz.

4.3.6 Virtualisierung des Speichers

Bei der Virtualisierung des Speichers muss zunächst zwischen dem On-Chip Speicher innerhalb des FPGAs und dem Off-Chip Speicher auf dem FPGA-Board unterschieden werden. Der On-Chip Speicher ist bei modernen FPGAs, wie in Abschnitt 2.1.2.2 vorgestellt, über den gesamten FPGA verteilt. Bei einer Zuteilung von vFPGAs auf partielle Regionen resultiert dieser Aufbau in einer von der Region abhängigen Anzahl an Ressourcen und entsprechend auch Größe des Speichers. Der Zugriff auf spezielle Ressourcen anderer Regionen ist mit der partiellen Rekonfiguration nicht ohne Weiteres vereinbar, sodass keine direkte Möglichkeit besteht, den On-Chip Speicher zu virtualisieren, ohne zu Overlays überzugehen. Da der Speicher in vFPGA-Slots eines vFPGAs liegt, der einem Nutzers zugeordnet ist, ist eine Virtualisierung jedoch nicht zwingend erforderlich. Aus der Größe der vFPGAs kann indirekt die Größe des dem Nutzer zur Verfügung stehenden On-Chip Speichers bestimmt werden.

Der On-Board-DDR-Speicher muss hingegen virtualisiert werden, um unterschiedlichen Nutzern bereitgestellt werden zu können und eine dynamische Zuteilung des Speichers zu den Nutzern zu ermöglichen. Ein vergleichbares Vorgehen wird auch bei Betriebssystemen eingesetzt, um den logischen Speicher vom physischen Speicher und seinen Eigenschaften wie der Größe und Verteilung zu abstrahieren.

Die klassischen Ansätze zur Virtualisierung von Arbeitsspeicher in Betriebssystemen ermöglichen eine dynamische Zuordnung von virtuellem Speicher auf physischen Speicher. Häufig genutzte Techniken sind hierbei die statische Einteilung der Bereiche, die Vergabe von Zugriffsrechten, Basis- und Grenzregister oder (mehrstufige) Seitentabellen [Tan06, S. 476].

4.3.7 Konfiguration der FPGAs und Rolle des Hypervisors

Nachdem der Entwurfsraum für die Virtualisierung selbst, aber auch für die Architektur der vFPGAs aufgezeigt wurde, werden im Folgenden die Möglichkeiten zur Kopplung des FPGAs mit dem Host-System analysiert. Dabei steht nicht die Hardwareschnittstelle im Vordergrund, sondern die Softwareebene und die Rolle von sowohl Host- als auch FPGA-Hypervisor selbst. In Abschnitt 4.2.1 wurde bereits die Notwendigkeit des FPGA-Hypervisors für eine Paravirtualisierung innerhalb des FPGAs erläutert, jedoch noch nicht auf dessen Aufgaben zur Verwaltung und Zuordnung der Ressourcen eingegangen. In Anlehnung an die Definition des Aufgabenbereiches für einen Hypervisor/VMM für die System-Virtualisierung ergeben sich folgende Aufgaben:

- I. Definition der Schnittstellen, Abstraktion von der physischen Hardware und Kapselung/Separation der Nutzer voneinander,
- II. Verwaltung der Ressourcenzuteilung und Rekonfiguration der einzelnen vFPGAs,
- III. Überwachung der Zustände der vFPGAs, insbesondere der Migration und
- IV. Steuerung des Zugriffs auf gemeinsame und daher virtualisierte (externe) Geräte (Speicher, Netzwerk etc.).

Tabelle 4.1: Entwurfsraum für die Verwaltung der vFPGAs auf Host- oder FPGA-Hypervisor.

Aufgabe	Host-Hypervisor	FPGA-Hypervisor
Zuordnung von vFPGA-Slots und Nutzern	✓	✗
Kapselung in vFPGAs und Überwachung der Zugriffe	!	✓
Verwaltung des aktuellen Zustandes der vFPGAs	!	✓
Realisierung der Migration und der erforderlichen Zustandsabfolgen	!	✓
Rekonfiguration der vFPGAs	!	✓
Verifikation der partiellen vFPGA-Images	✓	!
Überprüfung der zulässigen Regionen im Bitstream	✓	!
Zuordnung der externen Ressourcen (Netzwerk, Speicher etc.)	✓	✓
Verwaltung der Zugriffe auf externe Ressourcen	✗	✓
Organisation der Seitentabellen für den DDR-Speicher	✓	✗
Übersetzung von virtuellen auf physische Adressen und Absicherung der Zugriffe	✗	✓

✓: optimal !: eingeschränkt oder nur teilweise ✗: problematisch

Die unterschiedlichen Verwaltungsaufgaben und deren Komplexität sowie Eignung für eine vollständige Umsetzung auf Host- oder FPGA-Hypervisor wird in Tabelle 4.1 als Entwurfsraum aufgezeigt und bei der Realisierung der FPGA-Virtualisierung in Abschnitt 4.4 entsprechend berücksichtigt. Die Abstraktion von dem physischen FPGA erfolgt bei der gewählten Virtualisierung durch die Position und die Größe der bereitgestellten vFPGAs. Somit ist lediglich eine Abstraktion von den physischen Schnittstellen notwendig, welche idealerweise mittels der Paravirtualisierung innerhalb des FPGA-Hypervisors erfolgen sollte. Der Zugriff auf die Hardwareschnittstellen und die zugesicherten Datenraten ist dabei ebenfalls vom FPGA-Hypervisor entsprechend der Anforderungen der Nutzer zuzusichern, da hierbei die Realisierung in Hardware effizienter umgesetzt werden kann als in Software auf dem Host, wie auch in Tabelle 4.1 bei der Zuordnung der Aufgaben zu Host- und FPGA-Hypervisor gezeigt.

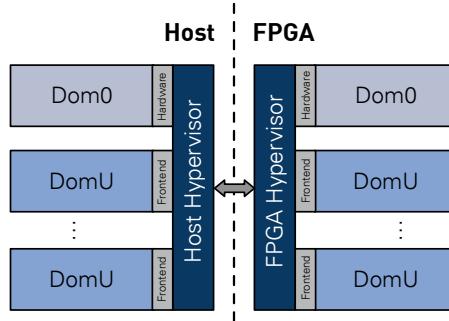


Abbildung 4.5: Aufteilung von Hypervisor auf Host und FPGA.

Die Zuordnung von den konkreten vFPGAs oder die Validierung der partiellen vFPGA-Bitstreams auf die zulässigen Bereiche sind hingegen aufgrund der Komplexität und Dynamik in Software auf dem Host auszulagern, von dem aus auch die eigentliche Konfiguration und Initialisierung der vFPGAs erfolgt. Das Host-System selbst verfügt aufgrund der Kapselung der Nutzer in VMs ebenfalls über einen Host-Hypervisor zur Bereitstellung eigener Betriebssystem-Instanzen und kann dementsprechend in seiner privilegierten Domain (Dom0) die vom FPGA ausgelagerten Verwaltungsaufgaben übernehmen und somit deren Sicherheit gewährleisten. Abbildung 4.5 zeigt das dafür erforderliche System mit den beiden Hypervisoren auf FPGA und Host sowie die VMs und die dazugehörigen vFPGAs.

Für die partielle Rekonfiguration ist ebenso ein Zusammenspiel der Verwaltung auf dem Host (Dom0), dessen Hypervisor für die Validierung und Übertragung eines partiellen vFPGA-Images sowie des FPGA-

Hypervisors zur Durchführung dieser Konfiguration (ICAP) erforderlich. Dabei ist zu gewährleisten, dass sich in dem vFPGA-Image die korrekten Bereiche der vFPGAs befinden beziehungsweise, dass das vFPGA-Image vom Cloud-Anbieter für exakt den zu konfigurerenden vFPGA signiert worden ist. Die Validierung einer Signatur kann sowohl auf dem Host als auch auf dem FPGA erfolgen, wobei bei letzterem unter Umständen das Zwischenspeichern des komplettem vFPGA-Images vor dem Laden in die Konfigurationsinfrastruktur notwendig wird. Die Verwaltung der Zustände der vFPGAs hingegen muss innerhalb beider Hypervisoren erfolgen, um ein reibungsloses und vor allem sicheres Zusammenspiel der beiden Komponenten zu ermöglichen.

Die Zuteilung von externen Ressourcen auf dem FPGA-Board und im Speziellen auf die Zugriffe von den vFPGAs auf diese unter Berücksichtigung der Zugriffsrechte mit zusätzlicher Gewährleistung einer Arbitrierung für die Zusicherung von Datenraten und Zugriffszeiten ist hingegen ausschließlich vom FPGA-Hypervisor aus effizient möglich. Eine vollständige Aufteilung der Zuständigkeiten ist demnach aufgrund der engen Interaktion der beteiligten Systeme ohne Weiteres möglich. Selbst wenn ein vFPGA vom Nutzer ohne VM auf dem Host und nur über das externe Cloud-Netzwerk betrieben wird, ist der Host dennoch für die Verwaltung der Ressource und ebenso für deren Konfiguration verantwortlich.

4.4 Virtualisierung von FPGAs im Cloud-Einsatz – RC2F

In diesem Abschnitt wird das Konzept zur Virtualisierung von FPGAs für deren Integration in Cloud-Architekturen mit der Bezeichnung RC2F entwickelt. Dessen Ziel ist es, nebenläufige vFPGAs unterschiedlicher Größe für verschiedene Nutzeransprüche bereitstellen zu können. Die Aspekte des zuvor besprochenen Entwurfsraums werden berücksichtigt und eine geeignete Variante zur Virtualisierung wird ausgewählt. Die wesentliche Zielsetzung des Einsatzes von FPGAs in der Cloud besteht darin, eine hohe Auslastung der physischen Ressource zu erreichen. Die Verknüpfung von FPGA und VM auf dem Host-System ist dabei ebenso notwendig wie die Möglichkeit unterschiedlicher Zugriffsarten. Einen vFPGA dynamisch zwischen den physischen Host-Systemen zu migrieren, um die Ressource effizient in einer Cloud zu nutzen, ebnet den Weg für die dynamische Integration der adaptiven vFPGAs.

4.4.1 Systemarchitektur für die RC2F-Infrastruktur

Auf Basis der in Abschnitt 4.3.1 aufgezeigten Möglichkeiten erfolgt die Festlegung auf eine Typ 1-System-Virtualisierung mit Paravirtualisierung innerhalb eines Hypervisors auf dem FPGA. Unter Berücksichtigung dieser Aspekte sowie nach Abwägung der in Abschnitt 4.3.2 diskutierten Möglichkeiten ergibt sich das in Abschnitt 4.4.1 aufgezeigte System mit größtmöglicher Flexibilität für die Nutzer. Das System wird in den folgenden Abschnitten in seinen Bestandteilen näher erläutert. Ein für die angestrebte Virtualisierung geeigneter FPGA muss als Voraussetzung die dynamische partielle Rekonfiguration und einen Zugriff auf die interne Konfigurationsinfrastruktur ermöglichen. Für eine Verschlüsselung des FPGA-Bitstreams sind dabei zudem ein Flash-Speicher sowie BBRAM oder eFUSE-Register von Interesse [Xil16b]. Als Architekturen, welche diese Voraussetzungen erfüllen, dienen moderne Xilinx-FPGAs [Xil16c], wobei versucht wird, weitestgehend von den Produkten einzelner Hersteller zu abstrahieren. Denkbar wären beispielsweise auch FPGAs von Intel, welche ähnliche Eigenschaften aufweisen [Int16b].

4.4.1.1 Überblick

Die Systemarchitektur der RC2F-Infrastruktur wird in Abbildung 4.6 aufgezeigt. Das System orientiert sich an dem in Abschnitt 4.3.2 vorgestellten und speziell in Abbildung 4.4(d) illustrierten Systementwurf. Der statische Teil umfasst sämtliche Komponenten zur Kommunikation mit der Außenwelt, den FPGA-Hypervisor mit der Infrastruktur zur partiellen Rekonfiguration und Kontrolleinheiten für FPGA-Hypervisor (Hypervisor Control Unit (HCU)) und den vFPGA-Slots mit ihren Kontrolleinheiten (vFPGA Control Units (vCUs)). Der partiell rekonfigurierbare Bereich besteht aus den vFPGA-Slots, welche für die eigentlichen Nutzerdesigns innerhalb der vFPGAs vorgesehen sind.

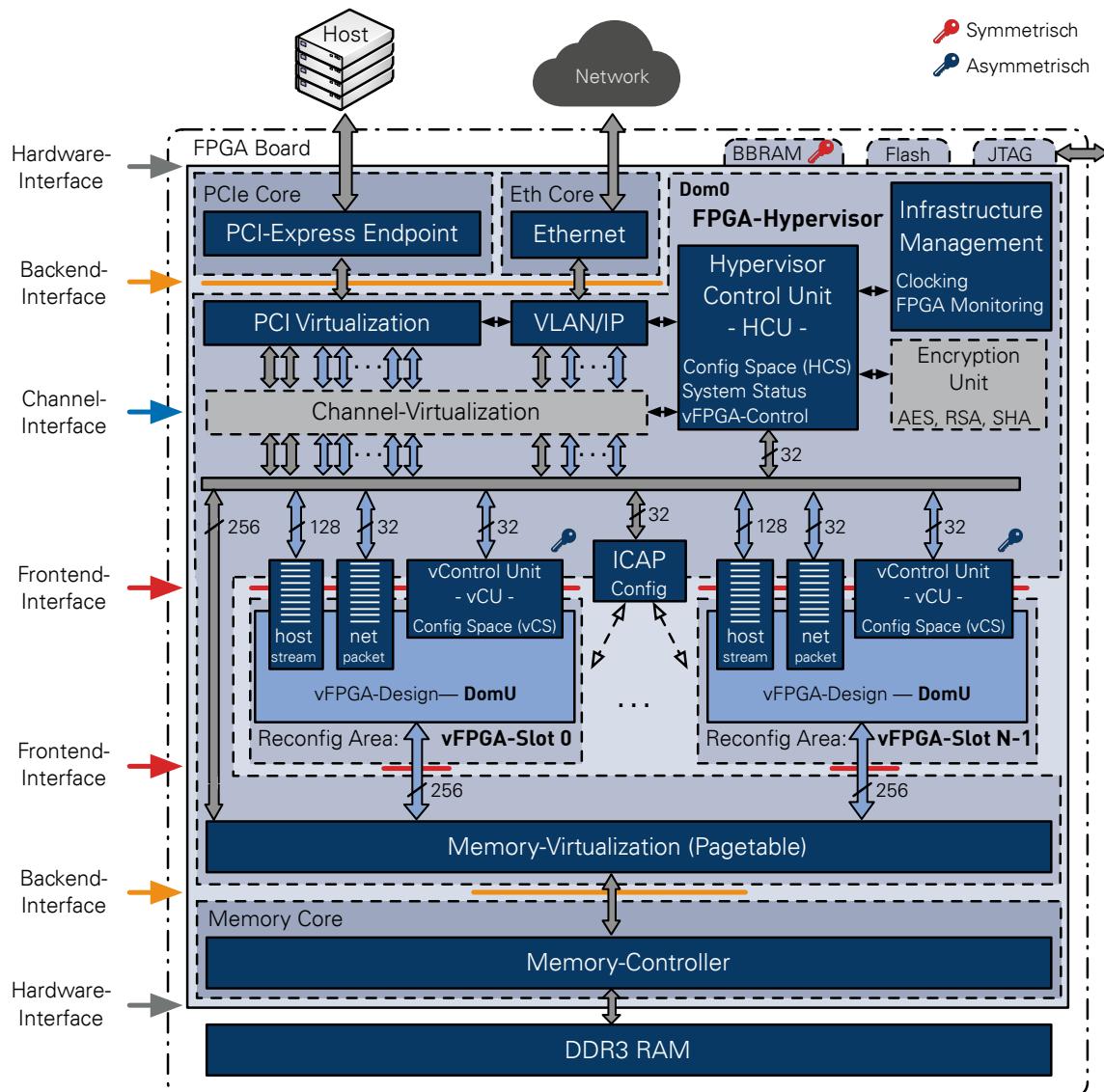


Abbildung 4.6: Systemarchitektur der RC2F-Infrastruktur mit FPGA-Hypervisor, dessen Kontrolleinheit (Hypervisor Control Unit (HCU)) sowie den vFPGA-Slots mit deren Kontrolleinheiten (vFPGA Control Unit (vCU)). Es sind lediglich die Datenleitungen ohne Steuersignale und Adressleitungen dargestellt. Die Datenleitungen für den Speicher und die beiden FIFOs sind je doppelt als Ein- und Ausgabe vorhanden. Die grauen Komponenten sind nicht innerhalb des im Abschnitt 6.1 realisierten RC2F-Prototypen vorhanden.

Als Schnittstellen zur Außenwelt dienen dem System, welches in Kapitel 6 prototypisch entwickelt wird, PCIe, Ethernet sowie eine JTAG-Schnittstelle zur initialen Konfiguration des FPGAs mit der in Abbildung 4.6 abgebildeten RC2F-Infrastruktur. Das statische Gerüst des FPGAs, das die Virtualisierung realisiert, besteht aus dem FPGA-Hypervisor, der sich zwischen den Backend- und Frontend-Interfaces, also zwischen der Schnittstelle zu den Hardware-Controllern und den vFPGAs, befindet. Im statischen Bereich liegen dementsprechend ebenfalls die Hardware-Controller und Steuereinheiten für die externen Komponenten selbst (PCIe-, Ethernet-Core und Speichercontroller), wobei weitestgehend von der realen Schnittstelle abstrahiert wird, um die Möglichkeiten der Erweiterung und Portierung zu vereinfachen.

Dem FPGA-Hypervisor selbst kommt die Aufgabe zu, den FPGA grundlegend zu verwalten und insbesondere die Bestandteile der Paravirtualisierung, also die Front- und Backend-Schnittstellen, zwischen den partiell rekonfigurierbaren und statischen Bereichen auf dem FPGA zu organisieren. Bei der Virtualisierung der Schnittstellen innerhalb des FPGA-Hypervisors wird zwischen dem in Abschnitt 4.4.7 diskutierten externen Speicher und den weiteren Schnittstellen aus Abschnitt 4.4.5 unterschieden. Beim Speicher erfolgt nicht nur eine Virtualisierung der physischen Schnittstelle, sondern auch des Adressraumes des Gerätes in Form von Seitentabellen (Memory-Virtualization). Die Verwaltung des physischen FPGAs wird mittels einer separaten Kontrolleinheit, der *FPGA-HCU*, durchgeführt. Diese Einheit übernimmt des Weiteren die partielle Rekonfiguration der vFPGAs, überwacht diese und verwaltet die unterschiedlichen vFPGA-Slots über deren Kontrolleinheiten, die *vCUs*. Die beiden Einheiten und ihre Konfigurationsspeicher sowie die vFPGA-Schnittstellen werden im Folgenden separat betrachtet.

4.4.1.2 FPGA-Hypervisor – Hypervisor Control Unit (HCU)

Nachdem in Form einer qualitativen Analyse die Aufgaben des FPGA-Hypervisors in Abschnitt 4.3.7 skizziert und dem Host beziehungsweise dem FPGA zugeordnet wurden, wird auf Basis dessen im Folgenden zunächst der Aufbau des FPGA-Hypervisors skizziert sowie die Zuordnung der entsprechenden Aufgaben vorgenommen. Im folgenden Kapitel werden daraufhin die verbleibenden Aufgaben innerhalb des Host-Hypervisor umrissen und in den Gesamtkontext der Cloud eingebettet.

Die Steuereinheit für den FPGA-Hypervisor übernimmt die für die angestrebte Virtualisierung erforderlichen Verwaltungsaufgaben, welche nach Abschnitt 4.3.7 lokal innerhalb des FPGA-Hypervisors auszulagern sind. Die Schnittstelle zur Verwaltung des FPGA-Hypervisors bildet dabei Konfigurationsspeicher, der *Hypervisor Configuration Space (HCS)*, welcher über einen dedizierten Kommunikationskanal über PCIe in den Speicher des Host-Hypervisors eingeblendet wird. Somit sind sämtliche Vorgänge auf dem System sowie der HCS vollständig von den Nutzern abgeschirmt, sodass ein entsprechend hohes Sicherheitslevel gewährleistet werden kann. Der HCS selbst ist in Abbildung 4.7 prototypisch dargestellt und enthält neben den Informationen zur aktuellen RC2F-Infrastruktur auf dem FPGA auch Informationen zum Zustand des FPGAs, wie Temperatur, Takte und Konfiguration. Über den Speicher kann der Host-Hypervisor den vollständigen FPGA oder Teile davon zurücksetzen und die Kommunikationskanäle grundlegend konfigurieren.

Zum Austausch größerer Datenmengen verfügt das System über einen separaten dedizierten Datenkanal, um partielle Bitstreams für die vFPGAs zwischen dem Host-Hypervisor und den Konfigurations-schnittstellen auf dem FPGA (ICAP) oder Datenblöcke in anderen Funktionseinheiten auszutauschen. Das Ziel der Daten wird ebenso über den HCS festgelegt, wie beispielsweise die Adresse für Speicherzugriffe auf den On-Bord Speicher oder für die vollständigen Seitentabellen. Dank der Nutzung nur eines Kanals werden Ressourcen eingespart und aufgrund der ausschließlich sequenziell ablaufenden Konfiguration und Initialisierung der vFPGAs stellt dieser Kanal keinen Engpass dar. Des Weiteren werden

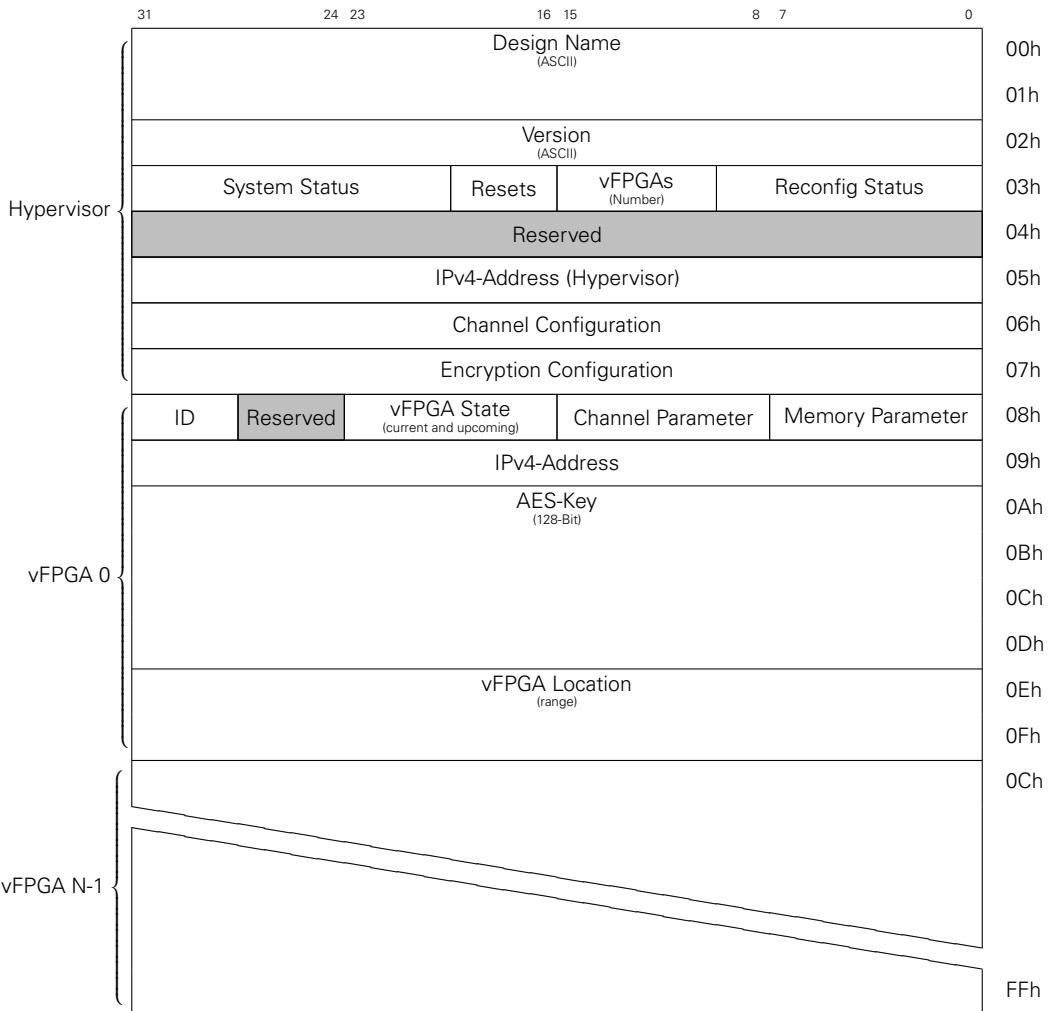


Abbildung 4.7: Prototypischer Aufbau des Hypervisor Configuration Space (HCS), innerhalb der Hypervisor Control Unit (HCU) des FPGA-Hypervisors zur Administration des physischen FPGAs einschließlich der Bereiche für die Verwaltung von bis zu 30 vFPGAs.

über den HCS die Kommunikationskanäle global konfiguriert und entsprechend vom FPGA-Hypervisor zusammen mit der Einheit zur Speicher-Virtualisierung (nachfolgend in Abschnitt 4.4.7 erläutert) gesteuert.

Ein partieller Bitstream kann auf dem FPGA direkt ver- und entschlüsselt sowie verifiziert werden (siehe Abschnitt 2.1.4.3). Dafür wird für jeden vFPGA entsprechend der private AES-Schlüssel des jeweiligen Nutzers hinterlegt (siehe Abschnitt 4.4.8). Die vollständige Überprüfung des Inhalts des Bitstreams erfolgt vor dessen Signatur durch den Anbieter der Cloud. Lediglich die Validierung der Bereiche innerhalb der vFPGA-Images wird vom FPGA-Hypervisor mit den zuvor vom Host-Hypervisor an den HCS übermittelten Daten abgeglichen, um Schäden an vFPGAs anderer Nutzer zu vermeiden.

Neben den Aufgaben der Konfiguration übernimmt der FPGA-Hypervisor ebenfalls die Verwaltung der Zustände der einzelnen vFPGAs (siehe nachfolgend Abschnitt 4.4.2) sowie die Validierung der vom Host-Hypervisor übermittelten Daten zu Regionen und den Datenraten der Nutzerkanäle. Um dies zu erreichen, ist im HCS in Abbildung 4.7 für jeden vFPGA ein separater Bereich reserviert. Eine Zuordnung der vFPGAs zu den realen Nutzern erfolgt dabei lediglich innerhalb des Host-Hypervisors.

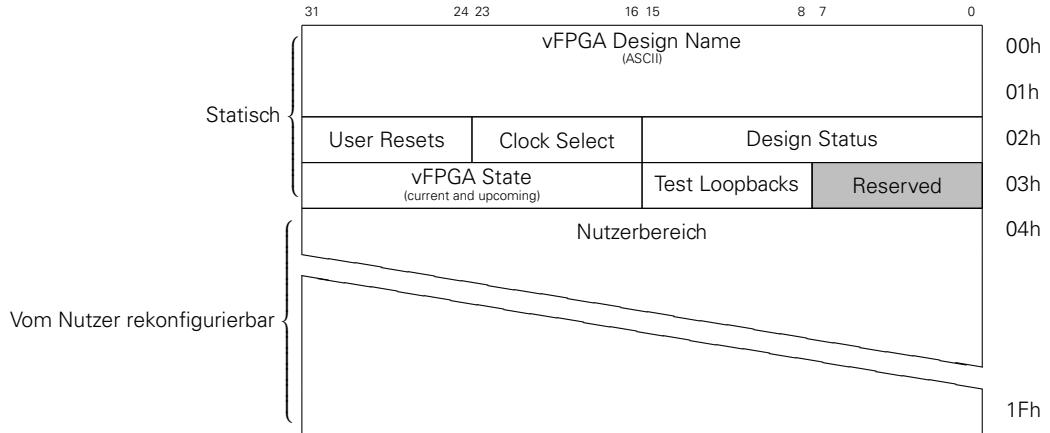


Abbildung 4.8: Virtual Control Space (vCS) innerhalb der vFPGA Control Unit (vCU) für die Verwaltung der vFPGAs mit statischen und vom Nutzer rekonfigurierbaren Bereichen.

4.4.1.3 vFPGA-Steuereinheit – vFPGA Control Unit (vCU)

Die vFPGAs, welche vom Nutzer wie in Abbildung 4.9 dargestellt wahrgenommen werden, stellen ein eigenes (wenn auch virtuelles) System dar und verfügen als solches, wie auch der FPGA-Hypervisor, über eigene Kommunikationskanäle (A und B) und einen eigenen Konfigurationsspeicher (C), den *Virtual Control Space* (vCS), sowie einen Zugang zum externen Speicher (D). Die beiden zum Datenaustausch vorgesehenen separaten Kanäle sind dabei der streaming- (A) und der paketbasierte Kanal (B), welche resultierend aus Abschnitt 4.3.3 als notwendig gelten müssen. Der vCS, welcher in Abbildung 4.8 abgebildet ist, wird in den Speicher der Nutzer-VM eingeblendet und kann mit Hilfe der in Abschnitt 5.1.3 erläuterten RC2F-API konfiguriert werden.

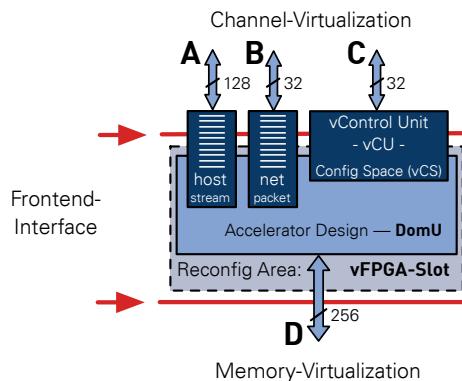


Abbildung 4.9: Architektur eines vFPGAs des RC2F mit der lokalen vFPGA Control Unit (vCU), welche den Virtual Control Space (vCS) beinhaltet. Die Datenleitungen für den Speicher und die beiden FIFOs sind je doppelt als Ein- und Ausgabe vorhanden.

Der vCS besteht aus zwei Teilen. Der statische Bereich wird aus den Bereichen für den jeweiligen vFPGA-Slot aus dem HCS eingeblendet und kann vom Nutzer in seinem Design nicht verändert werden; nur im Inhalt sind Anpassungen möglich, um einen systemweiten eindeutigen Zugriff zu gewährleisten. Ein Nutzer kann daher nur indirekt seine Bereiche innerhalb des HCS modifizieren, und ein Zugriff auf einen vCS von einem anderen vFPGA ist nicht möglich. Der Nutzer kann den Zustand seines vFPGAs einsehen und ändern, wobei der FPGA-Hypervisor eine höhere Priorität hat. Ebenfalls kann der Nutzer auf Basis des Systemtaktes einen eigenen Takt für seinen vFPGA auswählen und den vFPGA ebenso

vollständig zurücksetzen. Die Möglichkeiten entsprechen im Wesentlichen denen eines einfachen vollwertigen physischen FPGAs. Der untere Teil des vCS liegt in der rekonfigurierbaren Region und kann vom Nutzer völlig frei verwendet oder komplett ausgespart werden. Der Bereich kann für Speicher oder als I/O-Ports des vFPGAs eingesetzt werden, wodurch der Nutzer in seinen Gestaltungsmöglichkeiten ähnliche beziehungsweise sogar weitreichendere Freiheiten hat, als dies bei einem physischen FPGA gegeben wäre.

4.4.2 Zustände der vFPGAs und deren Verwaltung

Um die vFPGAs wie virtuelle Maschinen nutzen zu können, sind entsprechende Zustandsübergänge und deren Verwaltung erforderlich. Die Steuerung erfolgt vom Host-Hypervisor aus über den zuvor erläuterten HCS innerhalb des FPGA-Hypervisors. Ein direkter Zugriff der Nutzer auf die Zustände ist ebenso vom vCS der vFPGAs möglich, wobei deren Priorität geringer ist. Die Zustände der einzelnen vFPGAs werden über je eine Zustandsmaschine innerhalb des HCU verwaltet. Um jedoch den kompletten Lebenszyklus eines vFPGAs abzudecken, sind neben den Zuständen auf dem FPGA weitere Zustände auf dem Host-System innerhalb des Host-Hypervisors erforderlich. Diese analog zu Betriebssystemen zu betrachtenden vFPGA-Designs /-Images, welche Nutzern direkt zugeordnet sind und auch nutzerspezifische Daten enthalten können, werden nach Definition 3.5 als *vFPGA-Instanzen* bezeichnet. Dieser Aspekt ist insbesondere für sicherheitsrelevante vFPGA-Designs von Bedeutung, welche sowohl spezielle Anwendungslogik als auch private Schlüssel innerhalb des Bitstream enthalten können. Abbildung 4.10 gibt einen Überblick über die möglichen Zustände und deren Übergänge. Die Zustände werden innerhalb von Host- und FPGA-Hypervisor gleichermaßen verwaltet. Die vFPGA-spezifischen Zustände, welche jeder vFPGA einnehmen kann, sind dabei im Einzelnen:

- **Free (Host / FPGA):** Dem vFPGA-Slot ist keine vFPGA-Instanz zugeordnet und der Slot kann entsprechend vom Host-Hypervisor allokiert werden.
- **Booting (Host / FPGA):** Die Konfiguration des vFPGA-Slots mit einer vFPGA-Instanz erfolgt. Nach der erfolgreichen Konfiguration erfolgt der Übergang in den Zustand *Active*.
- **Active (Host / FPGA):** Der Nutzer kann die vFPGA-Instanz beliebig nutzen und den Zustand ändern.
- **Wait for Idle (Host / FPGA):** Wird der Zustand *Active* aufgrund des Zustandsüberganges *Migrate* oder *Pause* verlassen, werden sowohl vFPGA-Instanz als auch die Host-Anwendung gestoppt. Ist dies erfolgt, wird nach einer definierten Wartezeit in den Zustand *Snapshot* übergegangen.
- **Snapshot (Host / FPGA):** Es erfolgt das Auslesen der vFPGA-Instanz beziehungsweise des partiellen Bitstreams aus den vFPGA-Slots.
- **Paused (Host):** Die vFPGA-Instanz wird auf dem Hostsystem zwischengespeichert und kann fortgesetzt (*Resume*) oder in der Datenbank zwischengelagert werden (*Abort*).
- **Context Relocate (Host):** Der Bitstream der vFPGA-Instanz wird dahingehend modifiziert, dass diese innerhalb eines anderen vFPGA-Slots eingesetzt werden kann.
- **Resuming (Host / FPGA):** Erneute Konfiguration der allokierten vFPGA-Slots mit der vFPGA-Instanz.

Für einen in der späteren Praxis als Hardwarebeschleuniger eingesetzten vFPGA beschränkt sich der Lebenszyklus der vFPGA-Instanzen auf einen inneren Ring, bestehend aus dem Booten des vFPGAs

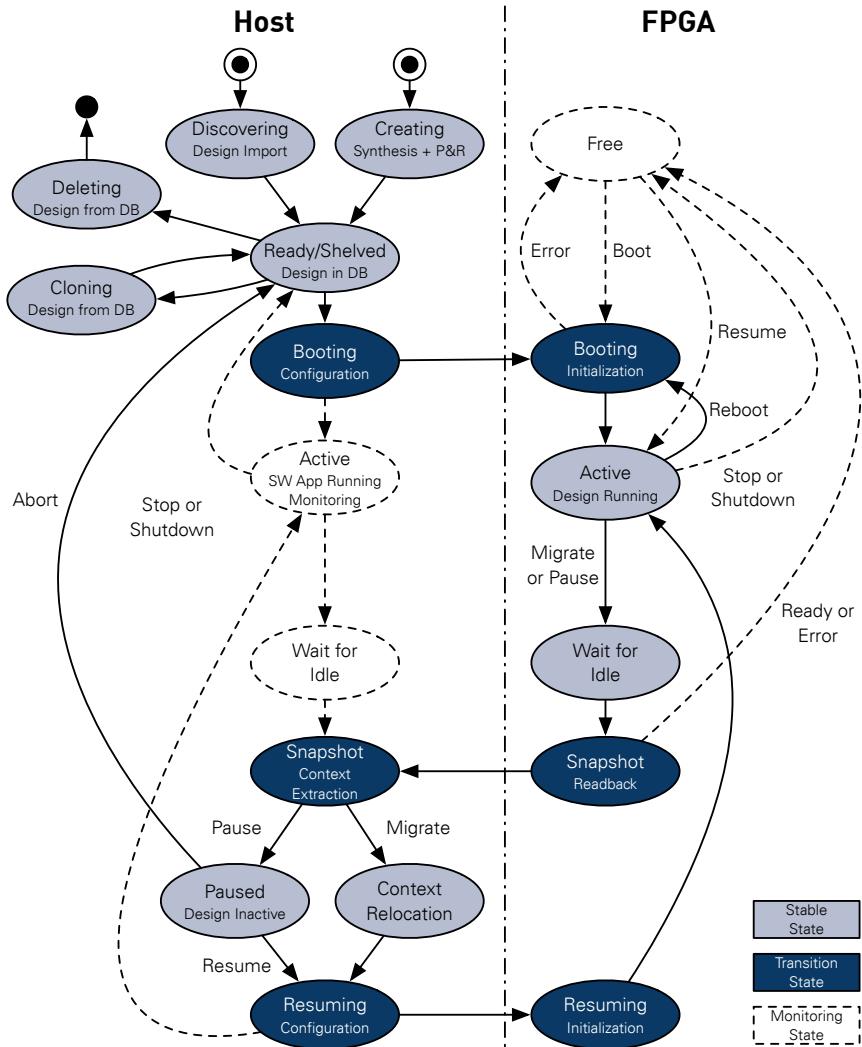


Abbildung 4.10: Zustände und Zustandsübergänge der einzelnen vFPGAs innerhalb der beiden Hypervisoren Host beziehungsweise FPGA. Abgebildet ist der komplette Lebenszyklus eines (nutzerspezifischen) vFPGAs. Die entsprechenden Zustände der Host-VM sind zum besseren Überblick nicht dargestellt, verhalten sich aber analog zum vFPGA, da beide über den Host-Hypervisor ihre Befehle zur Zustandsänderung erhalten.

(Konfiguration), dem Zustandsübergang von Host auf FPGA (Transition State) und der Ausführung (Active) des eigentlichen Hardwaredesigns. Das Stoppen (Herunterfahren) des vFPGAs entspricht dem Entfernen der vFPGA-Instanz. Abbildung 4.10 enthält des Weiteren den Zustand *Snapshot*, welcher dem Auslesen einer vFPGA-Instanz entspricht und damit das Pausieren und eine Kontextmigration ermöglicht. Dazu wird der vFPGA zuvor über den Zustand *Wait for Idle* durch eine vordefinierte Wartezeit in einen Zustand versetzt, in dem sämtliche Daten innerhalb des vFPGAs liegen, und nicht mehr innerhalb eines Hypervisors (Host oder FPGA) oder Systemtreibers. Des Weiteren ist dieser Zustand notwendig, um die DSPs innerhalb des vFPGAs entsprechend leer laufen zu lassen, da die internen Pipeline-Register bei einer Migration auf Basis eines zurückgelesenen Bitstreams einer vFPGA-Instanz nicht ausgelesen werden können. Das Signal muss hier also dementsprechend auch innerhalb der Nutzerdesigns berücksichtigt werden. Abbildung 4.10 zeigt ebenso Zustände, welche für die Verwaltung der vFPGA-Instanzen analog zu VMs in einer Cloud-Umgebung erforderlich sind, wie das Erstellen (*Creating*) oder Entfernen (*Deleting*) und das Bereitstellen in einer Datenbank (*Ready/Shelved*).

Die VM auf dem Host, welche mit dem vFPGA zusammenarbeitet und die Datentransfers steuert, verfügt selbst ebenfalls über einen Zustand. Die beiden Teilkomponenten müssen somit synchronisiert werden, um einen Datenverlust während der Datenübertragung zu vermeiden. Die zentrale Rolle dabei übernimmt der Host-Hypervisor, welcher zunächst das Senden der Daten aus der VM über deren RC2F-API (siehe Abschnitt 5.1.3) stoppt und anschließend ein Timeout abwartet, bevor der vFPGA angehalten wird und nach einem weiteren Timeout die VM entsprechend gestoppt wird. Auf diese Weise wird sichergestellt, dass sich sämtliche Daten zwischen VM und vFPGA jeweils sicher in einem der beiden Systeme befinden und nicht verlorengehen. Die Timeouts entsprechen dabei den Umlaufzeiten der Daten zwischen den beiden Systemen mit einem zusätzlichen Sicherheitsfaktor. Eine ähnliche Vorgehensweise ist beim Netzwerkverkehr notwendig, wobei hier ein Paketverlust durch die Nutzung entsprechender Netzwerkprotokolle vermieden wird. Als letzter Schritt muss schließlich das Auslesen der vFPGA-Instanz mit ihrem vollständigen Kontext erfolgen. Zusätzlich ist das Auslesen des Speicherbereiches, welcher dem vFPGA zugeordnet ist, aus dem externen DDR-Speicher des FPGA-Boards erforderlich.

4.4.3 Konfiguration von FPGA und vFPGAs

Die Konfiguration des FPGAs erfolgt, abhängig vom gewählten Servicemodell und den Rechten, welche dem Nutzer eingeräumt werden, entweder für den gesamten FPGA oder nur für die zur Verfügung stehenden vFPGAs. Die Konfiguration wird ausschließlich vom Host-Hypervisor durchgeführt, um mittels der Cloud-Verwaltung die Nutzerrechte überprüfen zu können. Das RC2F-Grunddesign, welches die vFPGAs bereitstellen, wird automatisch beim Einschalten des Systems aus dem Flash-Speicher des FPGA-Boards geladen, wie für die Servicemodelle in Abschnitt 3.1.2 skizziert wurde. Somit ist sichergestellt, dass sämtliche Servicemodelle zum Systemstart einsatzbereit sind.

4.4.3.1 Unabhängige vFPGAs durch dynamische partielle Rekonfiguration

Bei den Servicemodellen RAaaS und BAaaS wird das RC2F-Grunddesign eingesetzt, wodurch im Folgenden die dynamische partielle Rekonfiguration über die interne Schnittstelle zur FPGA-Infrastruktur (ICAP bei Xilinx FPGAs) erfolgen kann. Dadurch wird die Geschwindigkeit für die Rekonfiguration gegenüber der externen JTAG-Schnittstelle deutlich erhöht und eine schnelle Rekonfiguration für den Cloud-Einsatz wird möglich.

Da die vFPGAs mittels partieller Rekonfiguration dynamisch auf den physischen FPGA geladen werden, ist eine Platzierung auf diesem und damit die Festlegung von sowohl statischem als auch dynamischem Teil erforderlich. Bevor die Topologie der Anordnung festgelegt werden kann, ist eine Analyse der FPGA-Architekturen bezüglich der Anordnung von speziellen Funktionsblöcken wie dem ICAP oder der Kommunikationsschnittstelle zum Host erforderlich. Wie in Abschnitt 2.1.2.2 gezeigt, sind die physischen Positionen der eingebetteten PCIe-Endpunkte auf modernen FPGAs, wie beispielsweise der 7er-Serie von Xilinx, am Rand des FPGAs anzutreffen. Aus der Platzierung des FPGA-Hypervisors und entsprechender Anordnung der vFPGAs resultiert eine architekturabhängige Beschränkung der Entwurfsraumes.

Da des Weiteren die Konfigurations-Frames eine komplette Spalte der zeilen-orientierten Taktregionen ausfüllen [Xil16b], die Taktregionen jedoch von den vFPGA-Slots berücksichtigt werden sollten, ist eine ebenfalls zeilen-orientierte Anordnung des FPGA-Hypervisors über die Taktregionen hinweg zu bevorzugen. Wird der FPGA-Hypervisor so angeordnet, sind folglich die vFPGA-Slots horizontal beziehungsweise spaltenweise über den physischen FPGA innerhalb mehrerer horizontaler Taktregionen verteilt. Die vFPGAs verfügen dabei über eigene Taktbäume in ihrer Taktregion, wodurch benachbarte vFPGAs

nicht beeinflusst werden können und eine sichere Abkapselung der unabhängigen Nutzer voneinander erfolgen kann. Abbildung 4.11 zeigt beispielhaft die zu bevorzugende Anordnung der vFPGAs auf der physischen FPGA-Architektur. Eine Ausnahme bei aktuellen FPGA-Architekturen stellt hierbei lediglich der FPGA-Hypervisor dar, welcher sich über mehrere Taktregionen erstrecken muss, um die entsprechenden Frontends bereitstellen zu können.

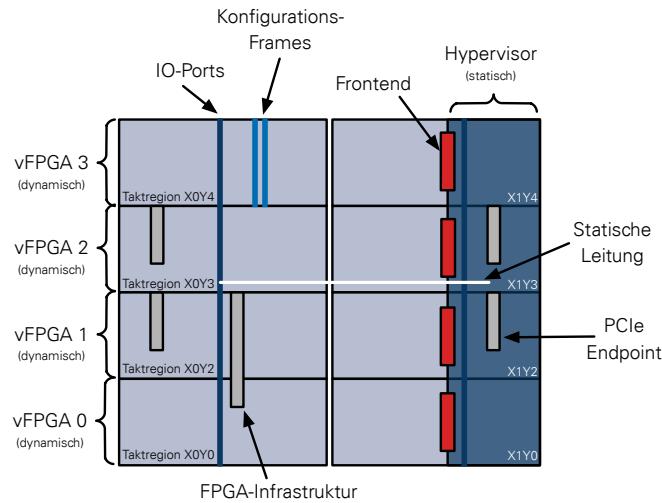


Abbildung 4.11: Mögliche Anordnung der vFPGA-Slots aufgrund der technischen Randbedingungen am Beispiel eines modernen inhomogenen FPGA mit vier übereinanderliegenden Taktregionen und den sich daraus ergebenden vier vFPGA-Slots.

Durch die Einteilung in statischen und dynamischen Bereich sowie eine feste RC2F-Infrastruktur mit den entsprechenden Frontend-Schnittstellen der vFPGAs ergeben sich im Entwurfsablauf automatisch die notwendigen *Partition Pins* (siehe Abschnitt 2.1.4.3). Diese bilden die physische Repräsentation der Frontend Schnittstelle zwischen FPGA-Hypervisor und den vFPGA-Slots. Um die Frontend-Interfaces in ihrer Position innerhalb der vFPGA-Slots einheitlich zu gestalten, werden die Partition Pins einmalig manuell im Entwurfsablauf für die RC2F-Infrastruktur festgelegt.

Zur Gewährleistung einer weitestgehenden Isolation der vFPGAs voneinander, wie es für sicherheitskritische Anwendungen erforderlich ist, können ebenfalls Techniken wie Isolation Design Flow (IDF) [Xil17h] eingesetzt werden. Auf diese Weise wird es möglich, die vFPGAs voneinander zu separieren. Dies wird durch die physische Abschottung der Regionen durch zusätzliche Randbedingungen und freie Bereiche auf dem FPGA (sogenannte Fences) gewährleistet [Xil17h].

4.4.3.2 Verschiebung von vFPGAs zwischen Bereichen

Eine Anordnung der vFPGA-Slots, wie sie in Abbildung 4.11 skizziert wird, ermöglicht das Verschieben der partiellen Bitstreams zwischen den vFPGA-Slots, da die Partition Pins innerhalb der vFPGAs immer an den gleichen Positionen zu finden sind. Einen Nachteil dieser einfachen eindimensionalen Verschiebung stellen lediglich die statischen Leitungen dar, welche jedoch aufgrund der I/O-Pins innerhalb der vFPGA-Slots notwendig sind. Ein weiteres architekturspezifisches Problem besteht darin, dass die spaltenweise übereinanderliegenden Bereiche bei aktuellen FPGAs nicht vollständig homogen sind, sondern immer wieder durch Komponenten wie PCIe-Endpunkte und FPGA-Infrastruktur unterbrochen werden, wie auch in Abbildung 4.11 gezeigt (siehe auch Abschnitt A.2).

Aufgrund dieser Einschränkungen sind zur Erzeugung der vFPGA-Designs verschiedene Entwurfsläufe für die unterschiedlichen Regionen notwendig. Gleichzeitig sollte jedoch nach Möglichkeit die Platzierung von Registern und Speichern an immer identischen Positionen innerhalb der vFPGAs erfolgen, um einen ausgelesenen Kontext innerhalb einer anderen vFPGA-Region wiederherstellen zu können. Für vollkommen homogene Regionen, welche von statischen Leitungen freigehalten werden, können – wie in einer Reihe von Arbeiten [BKT14; DRN13; KBT08; Oom+15] und zuletzt Rettkowski et al. in [RFG16] geschehen – die Bereiche verschoben werden. Auf diese Weise wird eine wesentliche Vereinfachung für den Entwurfsablauf von vFPGA-Designs erreicht, welche in Abschnitt 5.1.5 unter dem Aspekt des Entwurfsablaufs betrachtet wird.

4.4.4 Dynamische Größe der vFPGAs durch partielle Rekonfiguration

Die in Abschnitt 4.3.1 diskutierte dynamische Größe der vFPGAs lässt sich durch partielle Rekonfiguration erreichen, indem die entsprechenden zugrundeliegenden vFPGA-Slots zu einem größeren vFPGA zusammengefasst werden. Dabei ist lediglich darauf zu achten, dass das Hardwaredesign des vFPGAs an die gewünschte Anzahl der Frontend-Interfaces vom Nutzer selbst anzupassen ist. Die Partition Pins liegen dabei in einer speziellen Region, der Partition Pin Region (PPR), welche zusätzlich zum vFPGA hinzugefügt wird. Die FPGA-Ressourcen der vFPGAs sind somit immer ein ganzzahliges Vielfaches derjenigen eines einzelnen vFPGAs, wie bereits in Abbildung 4.3 aufgezeigt. Um die Platzierung zu vereinfachen, sind die vFPGA-Slots entsprechend vordefiniert und somit zwangsläufig vom physischen FPGA abhängig. Im Rahmen dieser Arbeit kann aus Platzgründen auf alle Eventualitäten und Spezialfälle aktueller FPGA-Architekturen unterschiedlicher Hersteller eingegangen werden, und es werden im Rahmen der Ergebnisse in Kapitel 6 lediglich Vorschläge für zukünftige Architekturen aufgezeigt.

4.4.5 Kommunikationsschnittstellen und deren Virtualisierung

Eine Virtualisierung der Schnittstellen ist erforderlich, um den vFPGAs Zugriff auf die nur einmal vorhandenen physischen Schnittstellen zu gewährleisten. Wie in Abschnitt 2.2.3.3 beschrieben, kann diese Virtualisierung mittels einer Paravirtualisierung erfolgen, erfordert dann aber eine entsprechende Koordination und Abstraktion innerhalb der Hypervisoren auf dem Host sowie dem FPGA. In der in Abbildung 4.6 vorgestellten Systemarchitektur sind entsprechend die Backend- und Frontend-Interfaces, zwischen denen die Virtualisierung erfolgt, gekennzeichnet. Die innerhalb des in Abschnitt 4.3.3 aufgezeigten Entwurfsraumes vorgestellten Kommunikationsarten und die entsprechenden Kanäle bilden dabei die Basis der Umsetzung. Im Folgenden werden zunächst Geräte wie PCIe und Ethernet virtualisiert, indem der wechselseitige Zugriff von verschiedenen vFPGAs koordiniert wird, bevor sich Abschnitt 4.4.7 der Virtualisierung des Speichers widmet.

Das Modul zur Hardware-Virtualisierung, welches auf PCIe und Ethernet aufsetzt und sich nach Abbildung 4.6 zwischen dem Backend- und dem Channel-Interface befindet, ist in Abbildung 4.12 aufgezeigt. Ausgehend von den physischen I/O-Ports auf dem FPGA (Hardware-Interface) folgen als nächste Komponenten auf dem Weg zum *Channel-Interface* der vFPGAs, wie es in Abbildung 4.9 aufgezeigt wurde, zunächst die Hardware-Controller, welche die Basisfunktionalität der Geräte (Treiber) bereitstellen und damit dem eigentlichen Backend-Interface einer Paravirtualisierung entsprechen (siehe Abbildung 4.1). Auf diesem aufbauend wird als nächstes der konkurrierende Zugriff der vFPGAs durch einen Arbiter ermöglicht. Anschließend wird über eine weitere Schnittstelle die Möglichkeit geschaffen, vom eigentli-

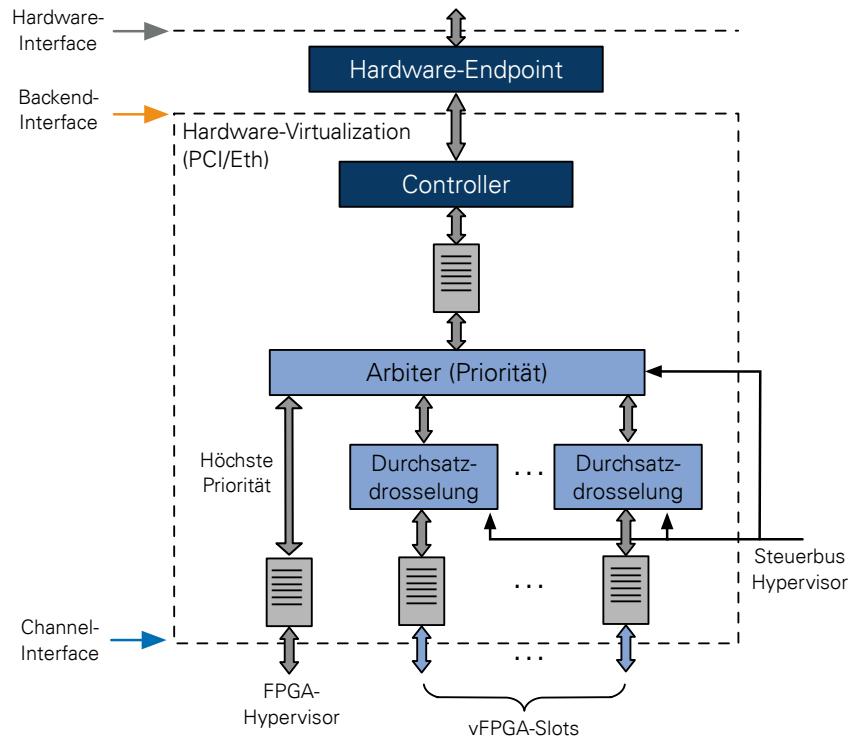


Abbildung 4.12: Prinzipieller Aufbau des Moduls zur Hardware Virtualisierung zwischen dem Hardware-Controller (Backend-Interface) und dem internen Kanal-Interface (Channel-Interface) zu den einzelnen vFPGA-Slots sowie dem FPGA-Hypervisor.

chen Kanal zu abstrahieren (Channel-Interface). Es folgt schließlich das Frontend-Interface, welches die Schnittstelle zu den vFPGA-Slots darstellt.

Die Datenentnahme aus den FIFOs zwischen Backend-Interface und Channel-Interface wird über den FPGA-Hypervisor gesteuert, um den dem Nutzer zugesicherten Datendurchsatz gewährleisten zu können. Die Festlegung dieser Datenrate erfolgt vom Konfigurationsspeicher des FPGA-Hypervisors, der wiederum vom Host-Hypervisor instruiert wird. Der Datenkanal für den Hypervisor (System) hat immer die höchste Priorität, um die vFPGA-Konfiguration oder systemrelevante Aufgaben schnellstmöglich abarbeiten zu können.

Auf das Channel-Interface aufsetzend erfolgt die Channel-Virtualisierung, welche das eigentliche Frontend-Interface zu den vFPGAs bereitstellt. Der wesentliche Grund zur Einführung dieser Schnittstelle besteht in der in Abschnitt 4.3.3 formulierten Forderung nach der Möglichkeit, zwischen vFPGAs eine direkte paketorientierte Kommunikation in Form eines NoCs aufzubauen. Da es sich dabei unter Umständen um Netzwerkverbindungen handelt, werden die vFPGAs über IP-Adressen adressiert. Zu diesem Zweck sind zunächst von der Cloud-Verwaltung die vFPGAs mit entsprechenden IP-Adressen zu versehen. Diese Pakete werden über die Kanal-Virtualisierung geleitet, die einen Switch zum Schalten der Verbindungen beinhaltet und über eine Zuordnung der auf dem FPGA vorhandenen vFPGAs zu ihren IPs verfügt. Dementsprechend können die Pakete direkt über den Switch an die lokal vorhandenen vFPGAs weitergeleitet werden. Zur möglichen Optimierung kann das System dahingehend ausgebaut werden, dass Pakete an vFPGAs auf physicalen FPGAs im selben Rechenknoten über die Schnittstelle zum Host geleitet werden, um eine entsprechend höhere Bandbreite zu erzielen.

4.4.6 Paravirtualisierung auf dem Host zur Inter-Domain-Kommunikation

Um das Konzept der Virtualisierung auch auf Host-Seite umzusetzen und dem Nutzer analog zum vFPGA eine VM bereitzustellen, ist auch hier eine Paravirtualisierung mit Frontend- und Backend-Schnittstelle notwendig. Durch Virtualisierung von FPGA und Host-System ergibt sich nutzerseitig eine einfache Sicht auf das System, wie in Abbildung 4.13 dargestellt. Dabei stehen dem Nutzer auf Seiten des FPGAs die drei unterschiedlichen in Abschnitt 4.4.1.3 aufgezeigten Schnittstellen zur Verfügung, wobei sowohl die streamingbasierte Schnittstelle (A) aus Abbildung 4.9 als auch der Konfigurationsspeicher der vFPGAs (C) in der VM in Form von Gerätedateien eingeblendet werden.

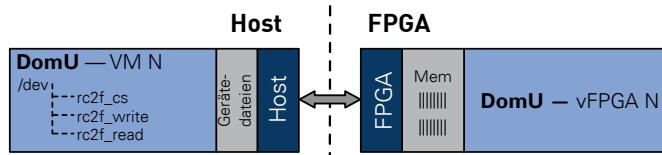


Abbildung 4.13: Logische Sicht der Nutzer auf ein VM/vFPGA-System mit direkter Kopplung und entsprechender Abbildung der Hardware-Interfaces der vFPGAs auf Gerätedateien der unterprivilegierten VM auf dem Host.

In der Analyse des Entwurfsraumes in Abbildung 4.3.3 wurde eine Inter-Domain-Kommunikation zur universellen und vor allem zur Laufzeit veränderbaren Zuordnung von vFPGAs und VMs als bevorzugter umfassender Lösungsansatz erarbeitet. Auf diese Weise besteht keine Abhängigkeit, wie sie bei systemspezifischen Verfahren gegeben ist, beispielsweise bei der I/O-Virtualisierung für PCIe 3.0-Geräte [San+14a; San+14b]. Dieser Ansatz führt vielmehr dazu, dass das physische Gerät primär im Host-Hypervisor beziehungsweise in der privilegierten VM (Dom0) sichtbar ist. Über die entsprechenden Hardwaretreiber werden Speicherbereiche oder Gerätedateien mittels DMA im Speicher eingeblendet.

Die Geräte, auf welche der Host-Hypervisor selbst direkten Zugriff benötigt, sind der Konfigurationsspeicher HCS des FPGA-Hypervisors (`rc2f_hcs`), die Streaming-Kanäle (`rc2f_config_*`) für die partiellen Bitstreams und System-Daten (`rc2f_system_*`) wie Seitentabellen (siehe Abschnitt 4.4.7) oder komplette Datenpakete für spezielle Systemmodule. Innerhalb des Host-Hypervisors ist die Zuteilung der Gerätedateien der einem Nutzer zugewiesenen vFPGAs zu den entsprechenden VMs durch die Verwaltungsebene erforderlich. Für einen vFPGA selbst sind die entsprechenden Geräte, welche in die VM eingeblendet werden, der vFPGA-Konfigurationsspeicher vCS (`rc2f_vcs`) und die Datenkanäle (`rc2f_*`), wie in Abbildung 4.14 dargestellt.

Um ein derartiges Durchreichen der Geräte zu ermöglichen, wird ebenfalls eine klassische Paravirtualisierung eingesetzt, bei welcher der Host-Hypervisor die Geräte über entsprechende Backend-Interfaces einer VM zuordnet. Diese besitzt wiederum ein spezielles Frontend-Interface, welches die Geräte innerhalb der VM bereitstellt beziehungsweise direkt über eine API, welche in Abschnitt 5.1.3 vorgestellt wird, zugänglich macht. Die sich ergebene Architektur mit einer Paravirtualisierung auf sowohl Host als auch FPGA ist in Abbildung 4.14 veranschaulicht.

Dieses Verfahren der Paravirtualisierung wird bei VMs üblicherweise für die Bereitstellung virtueller Netzwerkkarten genutzt, kann aber im Kern auf Kommunikationskanäle erweitert werden. Da zwangsläufig die Daten zwischen verschiedenen Speicherbereichen vom Nutzer in der VM über den Kernel und schließlich den Host-Hypervisor in der Dom0 ausgetauscht werden müssen, ist ein entsprechender Overhead bei der Kommunikation die Folge. Bei einer Virtualisierung mit dem Xen-Hypervisor [Bar+03] besteht die Möglichkeit, beispielsweise über gemeinsame Speicherbereiche innerhalb des Host-Hypervisors eine Kommunikation zu den VMs aufzubauen. Das Verfahren wurde ursprünglich dazu entwickelt,

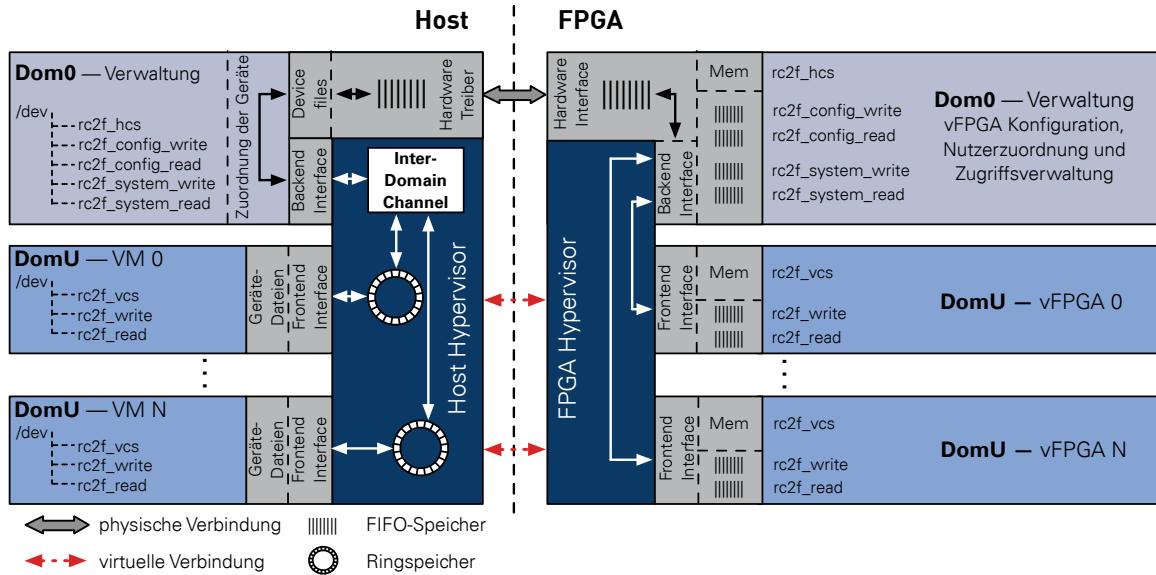


Abbildung 4.14: Hypervisor auf FPGA und Host mit Paravirtualisierung.

um statische Informationen des Host-Hypervisors einer VM zugänglich zu machen. Um eine VM über Daten zu informieren, kann ein spezieller Kanal mit geringem Datendurchsatz, aber auch geringer Latenz aufgebaut werden. Dieses Kommunikationsmodell ist bereits in vielen Arbeiten als Mittel zur Inter-Domain-Kommunikation etabliert. Die Speicherbereiche fungieren dabei als Ringspeicher, und die API bietet dem Nutzer einfache paketbasierte Kommunikationskanäle an [Spr+07].

4.4.7 Speicher-Virtualisierung

Um den reibungslosen Zugriff auf den externen DDR-Speicher des FPGA-Boards durch unterschiedliche Nutzer gewährleisten zu können, ist eine Speicher-Virtualisierung erforderlich. Diese führt im Wesentlichen dazu, dass die Nutzer mit virtuellen Adressen arbeiten, welche auf die physischen Adressen des Speichers übertragen werden. Wie bereits bei der Analyse des Entwurfsraumes in Abschnitt 4.3.6 gezeigt, eignet sich ein an Seitentabellen angelehntes Modell am besten zur Virtualisierung, da die Allokation der Speicherbereiche zwar nur beim Konfigurieren des vFPGAs erfolgt, aber aufgrund der unterschiedlichen Zeitpunkte und Allokationsgrößen eine Fragmentierung des Speichers trotzdem unvermeidbar ist.

Um den Speicher dennoch so effizient wie möglich auszulasten, sind daher Seitentabellen geeignet [Tan06, S. 476], wie auch bereits im Rahmen der Arbeit von Kemnitz [Kem15] gezeigt wurde. Abbildung 4.15 zeigt den Aufbau der Speicher-Virtualisierung mit Seitentabellen. Die globale Verwaltung der vollständigen Seitentabelle, aus der sich die Tabellen für die einzelnen vFPGAs ableiten, erfolgt innerhalb des Host-Hypervisors, der sämtliche Überschreitungen der Speicherbereiche abfängt. Der FPGA-Hypervisor erhält seine Tabellen direkt vom Host-Hypervisor und teilt sie entsprechend den vFPGAs zu. Nutzer haben somit nicht die Möglichkeit, Seitentabellen ohne den Host-Hypervisor zu verändern und auf Speicherbereiche anderer Nutzer zuzugreifen. Durch die Kapselung der Speicherzugriffe über den FPGA-Hypervisor wird sichergestellt, dass die Daten unterschiedlicher Nutzer vollständig voneinander separiert werden. Die Seitentabellen selbst werden vom Host-Hypervisor an die Speicher-Virtualisierung übertragen. Die Datenübertragung vom Host in den Speicherbereich der vFPGAs erfolgt zwangsläufig über die vFPGA-Instanz selbst und fällt somit in den Verantwortungsbereich des Nutzers.

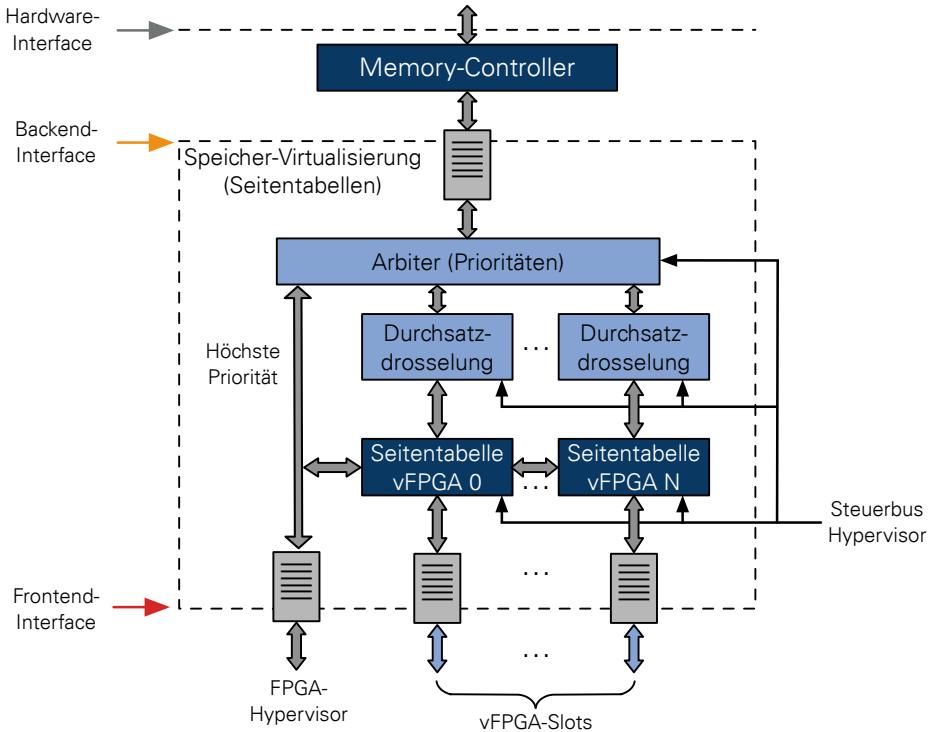


Abbildung 4.15: Speicher-Virtualisierung mit je einer Seitentabelle für jeden Nutzer und einem Kanal für den FPGA-Hypervisor.

Einen kritischen Parameter bei der Speicher-Virtualisierung stellt die Größe der einzelnen Seiten S_{page} dar. In Abhängigkeit der physischen Speichergröße S_{phy} kann entsprechend die Anzahl der sich daraus ergebenden Einträge in einer Seitentabelle N_{table} nach Gleichung 4.1 berechnet werden.

$$N_{table} = \frac{S_{phy}}{S_{page}} \quad (4.1)$$

Der Speicherbedarf einer Seitentabelle S_{table} berechnet sich entsprechend über Gleichung 4.2, wobei in der aktuellen Implementierung keine weiteren Bits für ergänzende Statusinformationen vorgesehen sind. Da jeder vFPGA prinzipiell alle Seiten allokiieren kann, sind Tabellen der Größe S_{table} in allen vFPGAs vorhanden.

$$S_{table} = N_{table} * (2 * \log_2(S_{phy})) \quad (4.2)$$

Für eine Auswahl unterschiedlicher Größen des physischen DDR-Speichers ergibt sich in Abhängigkeit verschiedener Seitengrößen der in Abbildung 4.16 gezeigte Speicherbedarf. Da die für einen vFPGA allokierte Speichermenge in der Regel mehrere MByte umfassen sollte, ist von Seitentabellen im Bereich von wenigen kBytes pro vFPGA auszugehen.

4.4.8 Erweiterung zur sicheren rekonfigurierbaren Hardware in der Cloud

Insbesondere wenn FPGAs für sicherheitsrelevante Anwendungen eingesetzt werden, ist eine Verschlüsselung der Bitstreams ebenso wünschenswert wie eine Überprüfung des Grunddesigns, damit

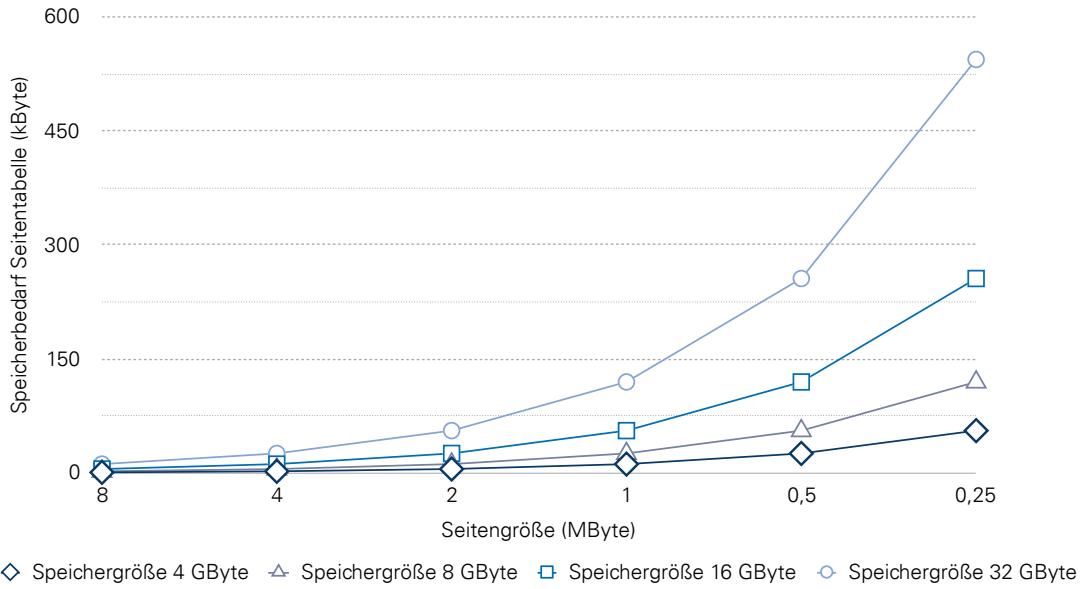


Abbildung 4.16: Größe der Seitentabellen in Abhängigkeit der Seitengrößen.

dieses nicht missbraucht werden kann, um Daten aus den Nutzerdesigns an Dritte zu übertragen. Insbesondere in Cloud-Architekturen existieren aufgrund der unterschiedlichen Akteure und einer entsprechend großen Vielzahl von unterschiedlichsten Angriffsvektoren und Angreifermodellen [Fer+14] zahlreiche Sicherheitsrisiken. Speziell wenn der Anbieter der Cloud oder des Rechenzentrums nicht vollständig vertrauenswürdig ist, wird eine zusätzliche Sicherheitseinrichtung oder Trusted Authority (TA) erforderlich, wie sie in einer Vielzahl von Arbeiten auf dem Gebiet der Kryptographie vorgeschlagen wird [DOW92; Fer+14] und bereits von Kapa et al. [Kep+08], aber auch Devic et al. [DTB10] und Eguero et al. [EV12] für FPGAs evaluiert wurde. Ein Sicherheitskonzept für die Bereitstellung virtualisierter FPGAs unter Einbezug einer TA ist in Abbildung 4.17 aufgezeigt.

Entscheidend bei der Bereitstellung von vFPGAs in nicht vertrauenswürdigen Umgebungen ist, dass ein realer physischer FPGA als Zielarchitektur vorhanden ist und sich dieser auch über einen Schlüssel authentifiziert. Nur so kann darauf aufbauend die Sicherheit der vFPGAs und der darin enthaltenden Schlüssel garantiert werden. Der Schutz des vFPGA-Images erfolgt nach Abschnitt 2.1.4.4 bei aktuellen FPGA-Architekturen in der Regel mit symmetrischen Verfahren. Da dazu der Zugriff auf den physischen FPGA zur initialen Übertragung des Schlüssels in einer vertrauenswürdigen Umgebung erfolgen sollte und der Schlüssel jedem Nutzer bekannt sein muss, ist dies Vorgehen nicht für den Einsatz in der Cloud praktikabel.

Weil der symmetrische Schlüssel des FPGAs lesbar zu diesem übertragen wird, ist zwangsläufig eine TA notwendig, welche diese vertrauenskritische Aufgabe übernimmt. Dies kann entweder der Hersteller des FPGAs oder der Anbieter der virtualisierten FPGAs sein, welcher zu Beginn vollen physischen Zugriff auf die FPGA-Boards haben muss, um seinen Schlüssel zu hinterlegen. Darauf aufbauend kann die TA mit symmetrischen Verfahren wie AES den authentifizierten FPGA mit einer verifizierten RC2F-Infrastruktur konfigurieren, und ein Sicherheitskonzept für vFPGAs in der Cloud kann, wie in Abbildung 4.17 aufgezeigt, etabliert werden. Weiterführend wird mittels asymmetrischer Verfahren (elliptische Kurven) eine zusätzliche Infrastruktur für die vFPGAs bereitgestellt und somit die Übertragung eines vFPGA-Image (siehe Definition 3.4) vom Entwickler oder Nutzer an den authentifizierten RC2F-FPGA abgesichert.

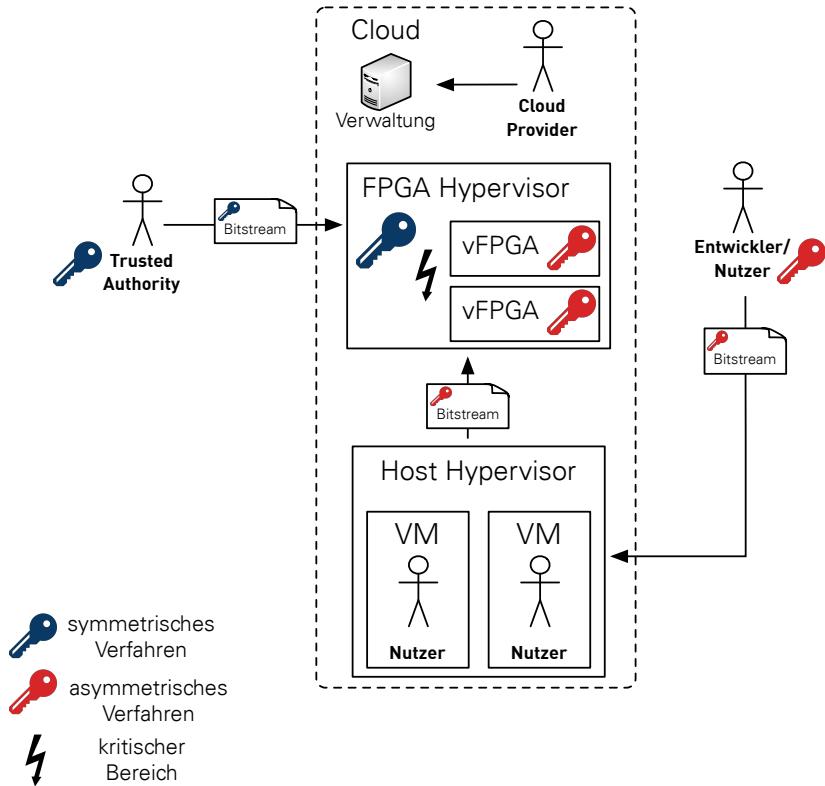


Abbildung 4.17: Sicherheitskonzept für FPGAs im Cloud-Einsatz. Die RC2F-Infrastruktur wird von der TA mittels symmetrischer, die vFPGA-Designs mittels asymmetrischer Verschlüsselung geschützt. Der kritische Bereich besteht innerhalb des FPGA-Hypervisors, da dieser vollen Zugriff auf die vFPGAs hat.

Zu diesem Zweck ist in der RC2F-Infrastruktur eine separate Encryption-Unit vorgesehen, welche vom FPGA-Hypervisor aus über dessen Konfigurationsspeicher HCS initialisiert werden kann. Eine Inspektion des partiellen Bitstreams (vFPGA-Image) ist innerhalb des FPGA-Hypervisors nicht möglich, sodass derartige sicherheitsrelevante vFPGA-Designs nur in vFPGA-Slots ohne statische Leitungen eingesetzt werden sollten, oder auf speziellen FPGA-Architekturen (siehe Abschnitt 6.4.1). Ein Nachteil dieses Ansatzes besteht darin, dass die TA, welche die RC2F-Infrastruktur liefert, über diese die vFPGA-Images der Nutzer auslesen kann. Dabei wird die Schwachstelle auf die TA als einzigen kritischen Bereich reduziert und alle anderen Beteiligten (siehe Abschnitt 2.3.1.5) werden ausgeblendet.

Indem lediglich nur ein Nutzer innerhalb der FPGA-Virtualisierung zugelassen wird, wie im Modell Reconfigurable Accelerators as a Service (RAaaS), ist allerdings eine weitere Steigerung der Sicherheit möglich, da ein möglicher Einfluss anderer Nutzer auf dem selben FPGA verhindert wird. Sämtliche Vorteile der in diesem Kapitel aufgezeigten Virtualisierung können dementsprechend mit einem hohen Grad an Sicherheit genutzt werden. Vollständig abgesicherte FPGAs, welche der Nutzer ohne Vertrauen in andere Beteiligte nutzen kann, sind in der Cloud mit aktuellen FPGA-Architekturen nur schwer möglich. Das Konzept wurde im Rahmen weiterer Arbeiten zur Erweiterung des RC2F zum *Secured FPGA (SecFPGA)* für Anwendungen mit hohen Sicherheitsanforderungen von Genssler in [Gen17] und Genssler et al. in [GKS17] untersucht.

5 Entwurfsprozess und Verwaltungshierarchie der Cloud

Nachdem in Kapitel 4 eine System-Virtualisierung für FPGAs vorgestellt wurde, widmet sich dieses Kapitel der notwendigen Software-Architektur für den Entwurfsprozess des Hardwaredesigns (vFPGA-Image) für die zuvor entwickelten vFPGAs. Ebenso wird die Verwaltungshierarchie für eine Integration der virtualisierten FPGAs in eine Cloud-Architektur aufgezeigt. Hierbei steht nicht der Aufbau eines vollständigen Entwurfsprozesses oder einer komplexen Cloud-Architektur im Vordergrund, sondern die Untersuchung ausgewählter Aspekte im Hinblick auf die Integration der zuvor virtualisierten FPGAs. Insbesondere sind dabei die Komponenten mit direkter Interaktion und Verwaltung beziehungsweise Bereitstellung der Ressource FPGA unter Verwendung der in Abschnitt 3.1.2 vorgestellten Servicemodelle von Bedeutung.

In Abschnitt 5.1 wird der Entwurfsprozess für die Hardwaredesigns erläutert, welcher den Entwicklern bereitgestellt werden muss. Nachdem die Interaktionsmöglichkeiten betrachtet wurden, widmet sich Abschnitt 5.2 dem konzeptionellen Aufbau einer Architektur mit den wesentlichen zuvor identifizierten FPGA-spezifischen Teilkomponenten. Die Beschreibungsmöglichkeiten der nutzerspezifischen vFPGAs werden in Abschnitt 5.3 in Form von drei unterschiedlichen Anwendungsszenarien aufgezeigt.

5.1 Entwurfsprozess und Verwendung der vFPGAs

Nachdem die Architektur des RC2F-Hardwaredesigns im vorherigen Kapitel erläutert wurde, wird in diesem Abschnitt die Erzeugung eines Hardwaredesigns für vFPGAs untersucht. Abschnitt 5.1.1 zeigt zunächst einen Entwurfsraum zur Erzeugung der Hardwaredesigns für vFPGAs, bevor anschließend in Abschnitt 5.1.2 ein angepasster Entwurfsablauf für vFPGAs innerhalb des Cloud-Dienstes RAaaS vorgestellt wird. Vervollständigt wird dieser Entwurfsablauf durch eine entsprechende API zur Interaktion mit den vFPGAs in Abschnitt 5.1.3 und der detaillierten Beschreibung einer Systemumgebung für vFPGA-Instanzen in Abschnitt 5.1.4. Der erläuterte Entwurfsprozess wird anschließend für eine Bereitstellung von Beschleunigern im Modell BAaaS in Abschnitt 5.1.5 erweitert. Dabei stehen insbesondere sowohl die Migration von vFPGA-Images als auch die in Abschnitt 5.1.6 aufgezeigte Einbettung sämtlicher relevanter Daten für den einfachen Einsatz als Hintergrundbeschleuniger für typische Cloud-Anwendungen in Form eines einfachen Paketes (vRAI) im Vordergrund. Mit einer Diskussion der Einsatzmöglichkeiten für sicherheitsrelevante Anwendungen in Abschnitt 5.1.7 schließt der Abschnitt.

5.1.1 Entwurfsablauf innerhalb der unterschiedlichen Servicemodelle

Bei einer Betrachtung des Entwurfsablaufs ist aufgrund der in Abschnitt 3.1.1 eingeführten Servicemodelle eine genauere Abstufung der Modelle erforderlich. So unterliegt der Entwickler im Modell RSaaS keinerlei Einschränkungen bezüglich seines Hardwaredesigns und ist dementsprechend nicht an einen

speziellen Entwurfsablauf gebunden. Der erzeugte Bitstream stellt insofern ein Sicherheitsrisiko dar, als dass keine Kontrolle bezüglich der Hardwareschnittstellen stattfindet und entsprechend das gesamte System, einschließlich dem Host, einem Risiko ausgesetzt wird. Aufgrunddessen ist das Modell RSaaS nur Nutzern in einem abgeschotteten Teil-System bereitzustellen.

Innerhalb des Servicemodells RAaaS besteht hingegen lediglich die Möglichkeit, eigene Hardwaredesigns innerhalb der vorgegebenen vFPGAs zu realisieren. Auch in diesem Modell werden dem Entwickler jegliche Freiheiten, abgesehen von einer Änderung der Schnittstellen (Frontend-Interfaces) gewährt, um unterschiedlichste Anwendungen entwickeln zu können. Das Ergebnis ist ein partieller Bitstream für einen vFPGA, welcher die Basis für eine vFPGA-Instanz bildet und nach Definition 3.4 als *vFPGA-Image* bezeichnet wird. Die vFPGA-Images können dann innerhalb des Servicemodells BAaaS als verifiziertes vRAI-Paket zur Hintergrundbeschleunigung von Cloud-Diensten angeboten werden.

5.1.2 Modifizierter Entwurfsablauf für Entwickler im Modell RAaaS

Im Modell RAaaS verfügt der Entwickler über weitestgehende Freiheiten in Bezug auf das Hardwaredesign innerhalb des gekapselten vFPGAs. Des Weiteren steht bei der Entwicklung der physische FPGA exklusiv zur Verfügung, um das entwickelte Hardwaredesign für unterschiedliche Größen der vFPGAs und verschiedene vFPGA-Slots zu evaluieren. Zu diesem Zwecke werden entsprechende Debugging- und Tracing-Werkzeuge der FPGA-Hersteller bereitgestellt. Der Entwurfsprozess kann sowohl lokal, als auch innerhalb der Cloud erfolgen. Für den Beschleuniger selbst wird die entsprechende RC2F-Infrastruktur zur Verfügung gestellt, welche das komplette statische Design als Netzliste enthält. Der grundlegende Aufbau eines vFPGAs, für den ein Design innerhalb des Modells RAaaS entwickelt werden soll, wurde in Abschnitt 4.4.1 und speziell in Abbildung 4.9 vorgestellt. Da dem Nutzer demnach die Schnittstellen und die möglichen Größen des vFPGAs sowie die verfügbaren Ressourcen innerhalb der vFPGAs bekannt sind, ist für ein eigenes vFPGA-Design (siehe Definition 3.3) lediglich dieses einfache Gerüst als Basis nötig. Dadurch ist es möglich, den kompletten Prozess nicht zwangsläufig in der Cloud auszuführen. Die Ressourcen können nach einer Synthese ermittelt werden und bieten einen ersten Anhaltspunkt zur Wahl einer geeigneten vFPGA-Größe. Nach dem Syntheseschritt kann die Zuweisung eines vFPGAs mit ausreichenden Ressourcen automatisch innerhalb des Entwurfsprozesses erfolgen, wobei der Nutzer entsprechende Informationen über Ressourcenverbrauch und Zeitverhalten erhält und iterativ sein Hardwaredesign optimieren kann.

Der Entwurfsprozess bietet ebenfalls die Möglichkeit, mittels HLS einen Beschleuniger zu entwickeln. Mithilfe entsprechender Zusatzbibliotheken können die Schnittstellen beschrieben werden, um eine einfache Integration in sowohl einen vFPGA auf Seiten der Hardware als auch in eine entsprechende API auf Seiten des Hosts (siehe Abschnitt 5.1.3) zu gewährleisten. Der zuvor erläuterte modifizierte Entwicklungsprozess zur Erzeugung eines Bitstreams im Modell RAaaS ist in Abbildung 5.1 exemplarisch aufgezeigt. Das statische RC2F-Design sowie Regionen der vFPGA-Slots und Frontend-Interfaces müssen dem Entwickler bereitgestellt werden.

Ergebnis des Entwurfsablaufes sind einerseits ein ausführbares Host-Programm sowie andererseits ein vFPGA-Image für spezielle vFPGA-Slots. Das erzeugte vFPGA-Image muss über die Ressourcenverwaltung der Cloud auf Kompatibilität hinsichtlich des zu konfigurierenden Bereiches auf dem physischen FPGA validiert werden, um entsprechende Schäden am Hypervisor, den Schnittstellen und damit dem Host-System zu vermeiden.

Der Entwurfsablauf innerhalb des Modells BAaaS ist aufgrund der angestrebten Migration der vFPGA-Instanzen deutlich komplexer und wird im Nachfolgenden in Abschnitt 5.1.5 aufgezeigt.

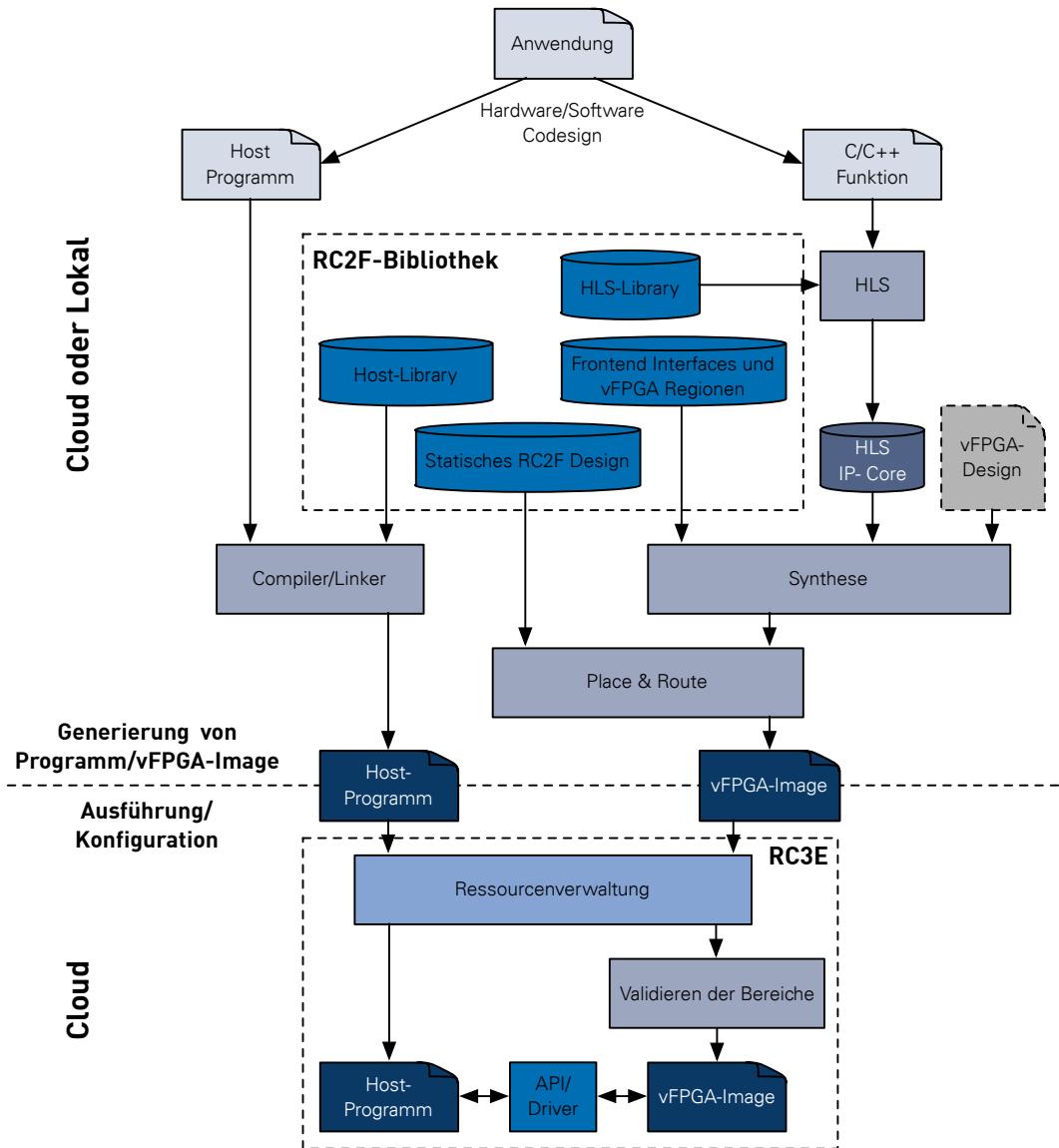


Abbildung 5.1: Entwurfsablauf zur Erzeugung eines vFPGA-Image auf Basis des RC2F im Cloud-Dienst RAaaS.

5.1.3 Bereitstellung von Host-API und Bibliotheken zur Integration der vFPGAs

Die Kommunikation zwischen vFPGA und VM ist aufgrund der statischen Schnittstellen zu den vFPGAs und den bereitzustellenden Treibern auf dem Host-System vordefiniert. Daher ist zur Interaktion mit den vFPGAs eine grundlegende API erforderlich, um späteren Cloud-Diensten auf dem Host eine einfache Schnittstelle zum Hardwarebeschleuniger innerhalb der vFPGAs zur Verfügung zu stellen. Neben der Initialisierung des vFPGAs muss die API insbesondere eine Konfiguration der innerhalb der VM verfügbaren vFPGAs über deren Konfigurationsspeicher (vCS) bieten und ebenso die Datentransfers realisieren. Tabelle C.1 gibt einen Überblick der wichtigsten Funktionen der API und ihrer Auswirkungen auf den FPGA beziehungsweise auf die vFPGAs. Neben der direkten Interaktion mit den vFPGAs ist des Weiteren über den Host-Hypervisor ein Zugriff auf den FPGA-Hypervisor (siehe Definition 3.8) möglich, um beispielsweise über die RC2F-API die Konfiguration eines vFPGAs zur Laufzeit zu ändern. Der daraus entstehende Entwurfsablauf auf Seiten der Software auf dem Host ist in Abbildung 5.1 durch die Host-Bibliothek innerhalb des RC2F gekennzeichnet.

Für den flexiblen Einsatz der vFPGAs sind ebenso eine stärkere Integration von HLS-Systemen in den Entwurfsprozess und eine engere anwendungsspezifischere Kopplung von FPGA und API erforderlich, um die Entwicklung von Anwendungen zu vereinfachen. Zu diesem Zweck ist die direkte Integration von HLS-Werkzeugen in den Entwurfsablauf möglich, indem über die RC2F-Bibliothek sowohl die Schnittstellen der Hardware auf Seiten des HLS-Rechenkerns als auch auf Seiten des Hosts eingebunden werden. Das Host-Programm muss entsprechend an den Schnittstellen vom Entwickler durch die bereitgestellten Funktionen ergänzt werden. Die daraus hervorgehende Umgebung ist vergleichbar mit OpenCL oder CUDA, wobei die API entsprechend eng mit dem Host-Hypervisor und somit der Cloud-Verwaltung zusammenarbeiten muss. Da die Beschreibung der vFPGAs selbst mehrere Parameter erfordert, um sowohl den FPGA-Hypervisor (HCS) als auch vFPGA-Konfigurationsspeicher (vCS) zu initialisieren, wird die Beschreibung der vFPGA-Instanzen in separate Konfigurationsdateien ausgelagert, welche im folgenden Abschnitt näher beschrieben werden.

Eine besondere Aufgabe kommt der API zuteil, wenn eine Migration von VM und vFPGA erfolgen soll, da hierbei die Datentransfers über die API vom Hypervisor schrittweise gestoppt werden müssen, um einen Datenverlust zu vermeiden, wie bereits in Abschnitt 4.4.2 erläutert. Die API greift dabei intern innerhalb des Treiber auf Signale zu, welche vom Host-Hypervisor generiert werden. Der Entwurfsprozess zur Bereitstellung von vFPGA-Designs, die an beliebigen vFPGA-Positionen instanziert und ausgelesen werden können, wird in Abschnitt 6.1.4.7 erläutert.

5.1.4 Beschreibung einer FPGA-Ressource als RCFG-Datei

Zur Bereitstellung der nutzerspezifischen FPGA-Ressourcen innerhalb der unterschiedlichen Servicemodelle ist eine zusätzliche Beschreibung des gewünschten Systems beziehungsweise der vFPGAs erforderlich. Diese Beschreibung der FPGA-Ressourcen mit Informationen über den Aufbau des Systems wird im Folgenden als **Reconfigurable (Device) Configuration (RCFG)** bezeichnet. Da bei der Beschreibung einer vFPGA-Instanz in der Regel eine VM auf dem Host involviert ist (siehe Abschnitt 4.3.7), orientiert sich die Beschreibung an derjenigen der VMs¹. Die Beschreibung der Ressourcen innerhalb der RCFG erfolgt aus zwei Gründen: Zum einen werden die Ressourcen, FPGAs oder vFPGAs entsprechend des gewählten Servicemodells von der Ressourcenverwaltung dem Nutzer zugeordnet, zum anderen werden die Einträge im Host-Hypervisor und ebenso innerhalb des FPGA-Hypervisors angelegt.

Da in Abhängigkeit der verschiedenen Servicemodelle unterschiedliche RCFGs erforderlich sind, werden diese im Folgenden skizziert und entsprechend erläutert. Listing 5.1 zeigt eine exemplarische RCFG, welche im Modell RSaaS `service='rs'` einen vollständigen physischen FPGA beschreibt, der als Instanz den Namen `name='fpga0'` erhält und in der VM-Instanz `vm='vm1-hvm'` mittels Hardware-Virtualisierung und PCI-Passthrough an einer bestimmten Adresse im PCI-Baum `pci='01:00.0'` eingeblendet wird. Die virtuelle Netzwerkadresse wird für das System über `vif='10.0.0.43'` eingerichtet. Für die VM muss eine eigene Konfigurationsdatei in Abhängigkeit der Virtualisierung vorliegen, wobei die Einbettung der VM-Konfiguration in die RCFG für den FPGA ebenso möglich ist. Da in diesem Modell unterschiedliche FPGAs bereitgestellt werden sollen, ist ein entsprechender Eintrag mit dem Namen des FPGA-Boards `board='vc707'` vorhanden. Die Konfiguration des FPGAs erfolgt mittels der JTAG-Schnittstelle `config='jtag'`, wobei ein initiales Design zusätzlich angegeben ist: `design='led.bit'`. Über eine RCFG-Datei kann im Modell RSaaS nur ein FPGA mit dazugehöriger VM allokiert werden, da das angeforderte System ansonsten zu komplex werden kann und eine Abbildung auf die physischen Hardwareressourcen nicht zwangsläufig möglich ist.

¹Die Basis für die vFPGA Konfigurationsdatei bildet eine Systembeschreibung, wie sie unter anderem durch den Xen-Hypervisor nach [Pic09, S. 122] genutzt wird.

Listings 5.1: Beispiel einer RCFG für einen vollständigen FPGA im Modell RSaaS.

<pre>service = 'rs' name = 'fpga0' vm = 'vml-hvm'</pre>	<pre>#Service Model RSaaS #FPGA-Instance Name #VM-Instance Name</pre>
<pre>board = 'vc707' vif = 'ip=10.0.0.43' vpci = '01:00.0' design = 'led.bit' config = 'jtag'</pre>	<pre>#FPGA-Board #FPGA-IP #PCI Node in VM-Instance #Initial Design #Configuration Method</pre>

Komplexer wird die Beschreibung der vFPGAs im Modell RAaaS `service=ra`, wie in Listing 5.2 gezeigt. Zusätzlich zu den bereits bekannten Parametern werden in diesem Beispiel über `vfpag=[2]` zwei vFPGAs mit unterschiedlicher Anzahl von vFPGA-Slots `size=[2, 1]` allokiert, wobei beide an dieselbe VM `vm=['vml-pvm']` durchgereicht werden. Die Anzahl der Frontend-Interfaces wird über `frontends=[2, 1]` für jeden vFPGA festgelegt werden, wobei die Anzahl kleiner oder gleich der Anzahl der vFPGA-Slots sein muss. An dieser Stelle muss die Cloud-Verwaltung versuchen, das gewünschte virtuelle System auf ein physisches System abzubilden, wobei im Modell RAaaS zusätzlich die Position des vFPGAs auf dem physischen FPGA durch das Feld `loc=[0,2]` angegeben werden kann. Über `debug=['csp']` wird in dem Modell RAaaS der Ressourcenverwaltung mitgeteilt, welche Debug-/Tracing-Schnittstelle zusätzlich zu instanzieren ist. Die Kapazität des externen DDR-Speichers kann ebenso angegeben werden (`memory=[2000,1000]`), wie auch der gewünschte Zustand der vFPGAs `boot=['paused']`. Die Position der Werte innerhalb der Listen, wie beispielsweise `size`, `loc`, `vif` oder `design`, ist entscheidet für die Zuordnung der Einträge untereinander. Ist des Weiteren in einer Liste wie zum Beispiel `key` nur ein Eintrag vorhanden, wird dieser für alle vFPGAs gleichermaßen übernommen. Ist keine eindeutige Zuordnung möglich oder ist diese nicht zulässig, wird eine Fehlermeldung ausgegeben.

Listings 5.2: Beispiel einer RCFG für einen vFPGA im Modell RAaaS.

<pre>service = 'ra' name = ['vfpag-bsmc'] vm = ['vml-pvm']</pre>	<pre>#Service Model RAaaS #vFPGA/User Design Name #VM-Instance Name</pre>
<pre>vfpag = [2] size = [2, 1] frontends = [2, 1] loc = [0,2] memory = [2000,1000] vif = ['ip=10.0.0.42','ip=10.0.0.43']</pre>	<pre>#Number of vFPGAs #vFPGA-Slots #Frontend-Interfaces #vFPGA location on physical device #DDR-Memory Size in MByte #vFPGA-IPs</pre>
<pre>debug = ['csp'] key = ['AAAABClyc2 ... Buay74HNE'] boot= ['paused'] design = ['bsmc-large.bit','bsmc-small.bit']</pre>	<pre>#Instantiate ChipScope Server in Dom0 #User AES-Key for vFPGA authentication #Initial vFPGA-State #Initial Design</pre>

Anhand der vorhandenen Daten erfolgt durch die Cloud-Verwaltung die Auswahl eines FPGAs, welcher die entsprechenden Kriterien erfüllt. Sind die Forderungen nicht auf das physische System abbildbar, muss das System eine entsprechende Fehlermeldung generieren. Der vFPGA kann beispielsweise nicht größer sein als ein physischer FPGA, und an eine VM können nicht mehr physische FPGAs direkt durch-

gereicht werden, als sich im Knoten befinden². Die RCFGs werden von der Cloud-Verwaltung nach der Überprüfung direkt vom Host-Hypervisor an den FPGA-Hypervisor übergeben, und die entsprechenden Daten werden in den GCS (siehe Abbildung 4.7) eingetragen.

Im Modell BAaaS hat der eigentliche Nutzer keine Kenntnis von den vFPGA-Ressourcen und die RCFG-Datei, welche zusammen mit allen erforderlichen vFPGA-Images in der vRAI hinterlegt wird, ist deutlich einfacher. Wie in Listing 5.3 gezeigt, sind beispielsweise keine Positionen der vFPGA-Slots (`loc`) oder Informationen zur Debug-/Tracing-Schnittstelle (`debug`) erforderlich.

Listings 5.3: Beispiel einer RCFG für einen vFPGA im Modell BAaaS.

```

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

vfgpa = [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 = ['AAAAABC1yc2 ... Buay74HNE']          #User AES-Key
boot= ['booting']                             #Initial vFPGA-State
design = ['kmeans-quad.vrai']                  #Initial Design

```

Die RCFGs müssen von der Ressourcenverwaltung in Hinblick auf die Rechte der Nutzer innerhalb des Servicemodells überprüft werden, bevor zunächst die globale Allokation der passenden Ressource erfolgt und dem Nutzer zugewiesen wird. Die konkrete Verarbeitung des Inhaltes geschieht anschließend innerhalb der Dom0 des zugewiesenen Knotens. Die Auswertung der RCFG und der Aufbau größerer und speziellerer FPGA-Anordnungen in Anlehnung an die Motivation in Abschnitt 3.2.1 wird für unterschiedliche Szenarien nachfolgend in Abschnitt 5.3 diskutiert.

5.1.5 Erweiterung des Entwurfsprozesses für das Modell BAaaS

Die auf dem in Abschnitt 5.1.2 innerhalb des Servicemodells RAaaS beschriebenen Weg erzeugten vFPGA-Images müssen für den produktiven Einsatz im Modell BAaaS zunächst für unterschiedliche, möglichst homogene vFPGA-Slots generiert werden. Aufgrund der Abhängigkeit von den FPGA-Herstellern wird dieser Aspekt im Rahmen der prototypischen RC2F-Virtualisierung nachfolgend in Kapitel 6 genauer betrachtet. Um die vFPGA-Designs auf FPGAs mit unterschiedlichen Nutzern einzusetzen, muss des Weiteren sichergestellt werden, dass innerhalb der vFPGA-Images die Bereiche mit denen der vorgesehenen vFPGA-Slots übereinstimmen und weder die RC2F-Infrastruktur im statischen Bereich noch andere FPGAs beschädigt werden können.

Nach einer Analyse des vFPGA-Images auf zulässige vFPGA-Slots werden weitere Kontextinformationen, wie zum Beispiel die ID des entsprechenden Frontends sowie die genutzten vFPGA-Slots, am Anfang des vFPGA-Images als zusätzliche Kommentarzeile hinzugefügt. Anschließend werden sämtliche Informationen mit dem geheimen Schlüssel des Cloud-Anbieters von diesem signiert und in einer Beschleuniger-Datenbank bereitgestellt. Eine Verifikation der Signatur ist vor jeder Rekonfiguration des

²Durch weiterführende Virtualisierung und Abstraktionsschichten kann ebenfalls dieses Problem gelöst werden, was allerdings aufgrund der Einbußen in Datenrate und Latenz beim Einsatz der FPGAs als Hardwarebeschleuniger eine weitere Herausforderung darstellt.

FPGAs innerhalb des Host-Hypervisors notwendig, um Schäden am System und an den vFPGAs anderer Nutzer zu vermeiden und damit die Sicherheit zu erhöhen. Die signierten vFPGA-Images können zusammen mit der erforderlichen Softwareanwendung für den Einsatz als vRAI, wie nachfolgend in Abschnitt 5.1.6 gezeigt, zusammengefasst und innerhalb des produktiven Dienstes BAaaS weiteren Nutzern zur Verfügung gestellt werden. Die Nutzer können so ihren Cloud-Dienst auf die Spezialhardware auslagern, ohne Kenntnis von den FPGAs im Hintergrund zu haben.

Für das Modell BAaaS stellt die Erzeugung von vFPGA-Images, welche innerhalb beliebiger vFPGA-Slots eingesetzt werden können, einen wesentlichen Aspekt dar. Besonders anspruchsvoll ist das Wiederherstellen eines vFPGA-Images innerhalb eines anderen vFPGA-Slots, da in diesem Fall sämtliche Komponenten an exakt den gleichen relativen Positionen innerhalb der homogenen vFPGA-Slots ange-siedelt sein müssen. Dieses kann mittels *Module Relocation* (siehe Abschnitt 2.1.4.3) erfolgen, indem die Frameadressen innerhalb der vFPGA-Images modifiziert werden. Die vFPGA-Images können dann aufgrund der homogenen vFPGA-Slots durch einfaches Verschieben an unterschiedlichen vFPGA-Slots instanziert werden. Für den Sonderfall, dass zusätzlich statische Leitungen innerhalb der vFPGA-Slots vorliegen, ist ein wiederholtes Verdrahten der Leitungen für die unterschiedlichen vFPGA-Slots erforderlich. Die höhere Laufzeit für die dafür erforderlichen Entwurfsläufe ist vertretbar, da die entstehenden vFPGA-Images mehrfach eingesetzt werden können und somit eine relativ lange Lebensspanne besitzen. Sämtliche erzeugten vFPGA-Images, welche erforderlich sind, um eine vFPGA-Instanz innerhalb beliebiger vFPGA-Slots zu instanziieren, werden in ein vRAI-Paket gekapselt, welches nachfolgend in Abschnitt 5.1.6 vorgestellt wird.

Die Migration einer vFPGA-Instanz erfolgt, wie bereits in Abschnitt 4.3.5 erläutert, ohne Eingriff in das eigentliche Hardwaredesign und ist fast ausschließlich Aufgabe der Software. Es sind somit keine zusätzlichen Hardwareressourcen innerhalb der vFPGAs erforderlich, welche einen Eingriff in die Nutzerdesigns darstellen und neben der Erhöhung der benötigten Menge an Ressourcen auch das Zeitverhalten der vFPGA-Images negativ beeinflussen würden. Das vollständige Auslesen einer kompletten vFPGA-Instanz ist ebenso über die Konfigurationsinfrastruktur des FPGAs möglich, sodass keine zusätzlichen Ressourcen auf dem FPGA beansprucht werden. Damit wird auch die Migration vollständig auf den Host-Hypervisor ausgelagert, der, wie bereits in Abschnitt 4.4.2 gezeigt, die Zustände von VM/vFPGA administriert.

Werden die verschiedenen vFPGA-Images nicht durch Module Relocation erzeugt, befinden sich Register oder Speicher an unterschiedlichen physischen Positionen innerhalb der vFPGA-Slots und ein Auslesen und Wiederherstellen eines Kontextes an unterschiedlichen vFPGA-Slots ist nicht ohne Weiteres möglich. Um dennoch eine Migration zu ermöglichen, ist die Abbildung von Quell- auf Ziel-Bitstream erforderlich. Dazu werden sämtliche Informationen der Komponenten innerhalb aller vFPGA-Slots in einer Datenbank gespeichert und im Fall einer Migration entsprechend übertragen. Ein solcher Ansatz hat den Nachteil, dass unter Umständen große Datenmengen anfallen und ebenso ein zusätzlicher Rechenaufwand für die Migration erforderlich ist.

5.1.6 Kapselung vollständiger Beschleuniger-Umgebungen in vRAI-Pakete

Sämtliche für die Ausführung von Beschleunigern innerhalb des Modells BAaaS erforderlichen Dateien werden in Virtual Reconfigurable Acceleration Image (vRAI) als Pakete gebündelt, um sie als vollständig gekapselte Beschleuniger an die Entwickler eines Cloud-Dienstes auf den höheren Ebenen zu übergeben. Aus Sicht der Anbieter eines Dienstes besteht dabei die Anforderung, eine Anfrage in Form eines

5 Entwurfsprozess und Verwaltungshierarchie der Cloud

Funktionsaufrufes möglichst kompakt, sicher oder energieeffizient abzuarbeiten, ohne dabei Kenntnis von der physischen Hardware im Hintergrund zu besitzen.

Um einen Beschleuniger aus einer VM heraus zu allokiert und auszuführen, sind mehrere Bestandteile innerhalb des vRAI-Paketes (siehe Abbildung 5.2) erforderlich:

- Die erforderlichen vFPGA-Images für alle möglichen vFPGA-Slots mit zusätzlicher Kennzeichnung der FAR (siehe Abschnitt 2.1.4.3) zur Verifikation des zu rekonfigurierenden Bereiches (vFPGA-Slot).
- Die RCFG-Datei zur Beschreibung des erforderlichen vFPGAs, wie er für das vFPGA-Image notwendig ist (siehe Abschnitt 5.1.4).
- Die Host-Anwendung zur Initialisierung und Interaktion mit der vFPGA-Instanz, welche direkt in den auszulagernden Dienst des Nutzers eingebettet wird (siehe Abschnitt 5.1.3).
- Optional (bei statischen Leitungen): VCBM zum Auslesen der relevanten Bits innerhalb der vFPGA-Instanzen, welche den aktuellen Zustand (Kontext) kennzeichnen.

Eine vRAI selbst kann von Entwicklern in einen eigenen Dienst über die in der Host-Anwendung innerhalb der vRAI definierten Funktion eingebunden werden. Sämtliche Arbeitsschritte von der Allokation der Ressourcen über die Konfiguration der vFPGAs bis hin zur Initialisierung und eigentlichen Ausführung der vFPGA-Instanz wird von der Host-Anwendung innerhalb des Paketes übernommen.

Um die Validierung der Regionen innerhalb des Host-Hypervisors zu optimieren, werden die partiellen vFPGA-Images innerhalb der vRAI-Pakete zusätzlich mit den IDs und den FAR-Adressen der vFPGA-Slots versehen. Diese zusätzliche Kontrolle der Positionen verringert das Risiko, den falschen vFPGA-Slot zu konfigurieren. Des Weiteren enthalten die vRAIs die für die Migration erforderlichen Masken zum Auslesen und Wiederherstellen des Zustandes (Kontextes) für den Sonderfall, dass die vFPGA-Slots statische Leitungen enthalten, welche innerhalb der unterschiedlichen vFPGA-Images für verschiedene vFPGA-Slots berücksichtigt werden müssen.

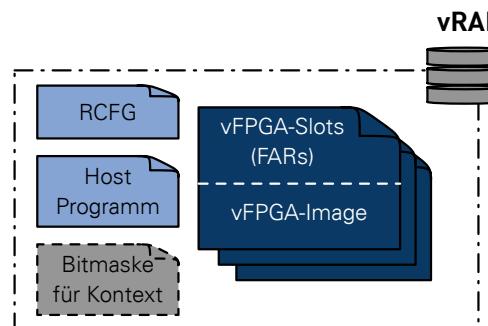


Abbildung 5.2: Virtual Reconfigurable Acceleration Image (vRAI)-Paket mit sämtlichen zum Ausführen einer vFPGA-Instanz erforderlichen Dateien wie vFPGA-Image, RCFG, Host-Programm und optionale Masken.

Bei Ausführung einer vRAI wird zunächst eine erforderliche vFPGA-Instanz, falls diese noch nicht vom Nutzer allokiert wurde, entsprechend der RCFG über die Ressourcenverwaltung oder indirekt über den lokalen Host-Hypervisor angefordert. Komplexere Systemkonfigurationen sind, wie in Abschnitt 5.3 aufgezeigt, dabei ebenso denkbar. Anschließend wird mittels des Host-Programms über die RC2F-API der zugewiesene vFPGA konfiguriert, wobei zunächst die vFPGA-Slots im Header des vFPGA-Images zur Validierung ausgelesen werden. Danach erfolgt die Initialisierung des vFPGAs über den HCS innerhalb des FPGA-Hypervisors, bevor ein fester Programmablauf oder auch eine interaktive Ausführung erfolgen kann.

5.1.7 Einsatz der vFPGAs und vRAIs für sicherheitsrelevante Anwendungen

Auf Basis der vFPGAs und der vRAIs können durch eine zusätzliche Verschlüsselung die vRAI-Pakete für sicherheitskritische Anwendungen eingesetzt werden. Dabei ist zu bedenken, dass hierbei, wie bereits in Abschnitt 4.4.8 erläutert, unter Umständen eine Trusted Authority mit einbezogen werden sollte, um die Authentizität der initialen statischen RC2F-Infrastruktur zu überprüfen. Ebenso muss die TA das verschlüsselte vFPGA-Image entsprechend in einer vertrauenswürdigen Umgebung innerhalb des Host-/FPGA-Hypervisors auf Konformität mit der statischen RC2F-Infrastruktur und den vFPGA-Slots überprüfen. Ein Zugriff auf das vFPGA-Image ermöglicht insbesondere die erforderliche Analyse der zu rekonfigurierenden vFPGA-Slots, wie sie bei statischen Leitungen innerhalb der vFPGA-Slots erforderlich ist. Die Arbeit mit verschlüsselten vFPGA-Images oder vFPGA-Instanzen ist folglich nur möglich, wenn entweder eine TA involviert ist oder die erforderlichen vFPGA-Slots komplett von statischen Leitungen freigehalten werden können.

Für eine größtmögliche Sicherheit muss nicht zwangsläufig auf die FPGA-Virtualisierung innerhalb des Dienstes BAaaS verzichtet werden. Wird nur ein einziger großer vFPGA auf dem physischen FPGA instanziert, oder gehören alle vFPGAs zum selben Nutzer, dann können die vFPGAs für sicherheitskritische Anwendungen eingesetzt werden. Um dem Nutzer des Weiteren eine exklusive Umgebung bereitstellen zu können, in der keine anderen Nutzer involviert sind, können ebenfalls die Dienste RSaaS und RAaaS angeboten werden. Wird RAaaS genutzt, ist zwangsläufig eine Konfiguration des gesamten FPGAs durch den Nutzer erforderlich.

5.2 Erweiterung einer System-Architektur zur Verwaltung von virtualisierten FPGAs

Der prinzipielle Aufbau des vollständigen Systems zur Integration von vFPGAs in eine Cloud wurde in seinen Grundzügen bereits in Abschnitt 3.3.2 aufgezeigt. Im folgenden Abschnitt 5.2.1 wird an einem Entwurf ausgewählter FPGA-spezifischer Komponenten der RC3E-Architektur gezeigt, welche Komponenten in einer Cloud-Verwaltung entsprechend angepasst beziehungsweise ergänzt werden müssen. Eine hierarchische Aufteilung der Ressourcenverwaltung wird in Abschnitt 5.2.2 skizziert. Abschnitt 5.2.3 widmet sich nochmals der Rolle des Host-Hypervisors, nachdem die wesentlichen Aufgaben und zeitlichen Abläufe skizziert wurden. Wie eine Zuordnung möglichst effizient mit einer Aufteilung der Arbeitslast auf die Hierarchieebenen erfolgen kann, wird schließlich in Abschnitt 5.2.4 diskutiert.

5.2.1 Aufbau und Verwaltungsebenen

Die grundlegende Systemarchitektur der rekonfigurierbaren Cloud-Verwaltung RC3E orientiert sich an den in Abschnitt 2.3.2 und Abschnitt A.3.1 gezeigten Komponenten sowie den etablierten Verwaltungssystemen OpenNebula [Ope16a] oder OpenStack [Ope16d]. Die wesentliche Besonderheit der zu entwickelnden Architektur stellt die Kopplung von VM und virtueller Hardwareressource – dem vFPGA – dar, welcher, anders als in typischen Architekturen, eine feingranulare Sicht auf die Ressourcen erfordert.

Abbildung 5.3 zeigt den Aufbau einer Cloud-Verwaltung mit den typischen Elementen, angepasst für das angestrebte RC3E-System zur Evaluation der Einbettung von FPGAs in eine Cloud, wie es bereits von Knodel et al. in [KLS16] beschrieben wurde. Für die Integration von vFPGAs sind *Lastverteilung* und *Scheduling* auf der Cloud-Verwaltung und eine erweiterte lokale Verwaltungsebene (*Lokale*

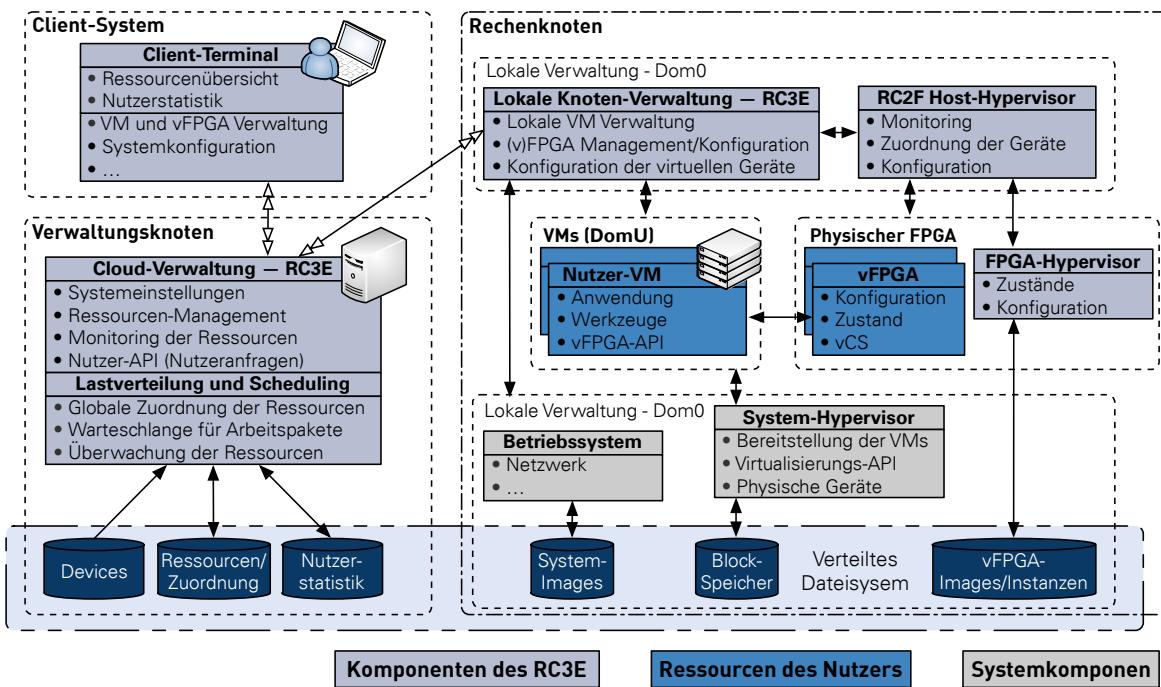


Abbildung 5.3: Architektur des Cloud-Ressourcenverwaltung RC3E zur Evaluation der Einbettung von FPGAs in eine Cloud-Architektur.

Knoten-Verwaltung) innerhalb der Dom0 auf dem Rechenknoten sowie dessen erweiterter RC2F Host-Hypervisor (siehe Definition 3.7) zur Integration der FPGAs von Bedeutung. Die VMs für die Nutzer werden über einen klassischen System-Hypervisor (siehe Definition 3.6), wie beispielsweise XEN (siehe auch Abschnitt 2.2.3.1) bereitgestellt. Die Ressourcenverwaltung RC3E ist eine typische Drei-Schichten-Architektur (3-Tier-Architektur) mit einer Trennung von Client-System, Cloud-Verwaltungsebene und untergeordneten Rechenknoten, welche die eigentlichen Aufgaben abarbeiten. Die Systeme bauen dabei auf ein gemeinsames Dateisystem für VM- und vFPGA-Images beziehungsweise komplett Beschleuniger in Form von vRAIs sowie Datenbanken zur Verwaltung des Systems und der Ressourcen auf. Die für eine Integration von FPGAs erforderlichen beziehungsweise anzupassenden Komponenten sind:

Cloud-Verwaltung: Auf oberster Verwaltungsebene der Cloud bilden die Etablierung der unterschiedlichen Servicemodelle aus Abschnitt 3.1.2 und das Durchreichen an die ausführenden Knoten die wesentliche Erweiterung, zusammen mit einer Lastverteilung und einem Monitoringsystem für die vFPGAs. Das System sollte möglichst an die unterschiedlichen in Abschnitt 3.1.2 vorgestellten Servicemodelle anpassbar sein und die speziellen Sicherheitsebenen in Bezug auf die FPGAs durchsetzen, welche für die jeweiligen Servicemodelle definiert wurden.

Lastverteilung und Scheduling: Die eingehenden Anfragen, Arbeitspakete oder Systemallokationen werden an die verfügbaren Knoten weitergereicht. Dazu ist eine Lastverteilung erforderlich, die Zugriff auf die relevanten Daten zur Auslastung und Verfügbarkeit der einzelnen vFPGAs hat. Der Aufwand der Zuteilung eines vFPGAs zu einem konkreten vFPGA-Slot auf dem physischen FPGA sollte möglichst an den eigentlichen Rechenknoten delegiert werden, um die Arbeitslast der Cloud-Verwaltungsebene innerhalb der Cloud-Architektur besser zu verteilen. Das dynamische Hinzufügen von weiteren Rechenknoten und Ressourcen zu den Nutzer-Ressourcen wird ebenfalls zentral von dieser Ebene in Form des *Load Schedulers* gesteuert (siehe Abschnitt 5.2.4), um der Anforderung einer elastischen Cloud möglichst gerecht zu werden.

Lokale Knoten-Verwaltung: Der Knoten ist zuständig für die feingranularen Aufgaben der Verwaltung und ordnet die vFPGAs den konkreten Bereichen auf dem FPGA und den eigentlichen VMs zu. Der interne Aufbau basiert auf dem Konzept der FPGA-Virtualisierung RC2F, welche in Abschnitt 4.4 eingeführt wurde. Die enge Kopplung zwischen Host- und FPGA-Hypervisor erfolgt dabei wie in Abschnitt 4.4.1.2 beschrieben, wobei der RC2F Host-Hypervisor des Weiteren die Aufgabe übernehmen muss, die einzelnen vFPGAs für die Ressourcenverwaltung zu überwachen.

Die notwendige Interaktion zwischen den Komponenten bei unterschiedlichen FPGA-spezifischen Szenarien wird im folgenden Abschnitt näher betrachtet.

5.2.2 Hierarchische Verteilung der Anfragen auf die verfügbaren Ressourcen

Die Cloud-Ressourcenverwaltung RC3E hat die Aufgabe, die vom Nutzer beschriebenen Ressourcen bereitzustellen, wobei die aktuelle Systemauslastung bekannt sein muss. Im ersten Schritt muss lediglich ein verfügbarer Rechenknoten identifiziert werden, welcher den Anforderungen genügt, also vFPGAs in ausreichender Zahl und insbesondere Größe bereitstellt. Um in möglichst kurzer Zeit eine entsprechende VM zu starten, muss die Entscheidung so schnell wie möglichst getroffen werden.

Steht der Rechenknoten fest, kann nach Starten der VM der entsprechende vFPGA vom lokalen Host-Hypervisor mit der entsprechenden zugrundeliegenden RCFG (siehe Abschnitt 5.1.4) vorbereitet werden. Diese Vorbereitung umfasst im Wesentlichen die konkrete Zuordnung eines vFPGAs zu den vFPGA-Slots auf einem physischen FPGA. Die Zuordnung erfolgt demnach in zwei Schritten, um nicht alle Entscheidungen innerhalb der Cloud-Verwaltung zu treffen, sondern die Details auf den Rechenknoten mit den Ressourcen auszulagern. Die Aufgaben, Abläufe und involvierten Ebenen der Cloud sowie der speziellen Teilkomponenten werden in Abbildung 5.4 für die drei Szenarien (I) der Allokation einer vFPGA-Ressource mit VM, (II) dem Starten eines vRAI-Paketes und (III) der Freigabe einer vFPGA-Ressource veranschaulicht. Die Ebenen der 3-Tier-Architektur sind dabei (A) der Cloud-Verwaltungsknoten, (B) der Rechenknoten mit einer (freien) vFPGA-Ressource und (C) der physischen FPGA-Ressource in diesem Rechenknoten. Anhand dieser drei nachfolgend erläuterten Szenarien werden die wesentlichen Schritte, welche den Lebenszyklus eines vFPGAs in der Cloud abdecken, skizziert:

(I) Allokation einer vFPGA-Ressource mit VM: Ausgehend von der Anfrage, welche auf der obersten Ebene im Cloud Verwaltungsknoten (A) eingeht, wird zunächst aufgrund einer grobgranularen Systemauslastung (CPU und FPGA) ein Rechenknoten mit ausreichenden freien Ressourcen in ausreichender Größe (B) ermittelt. Nach einer unverbindlichen Reservierung in der globalen Datenbank (A) erfolgt eine Zuordnung innerhalb des Rechenknotens (B) auf Basis der RCFG für die vFPGA-Instanz (siehe Abschnitt 5.1.4). Nach erfolgreicher Zuordnung erfolgt der bindende Eintrag in der Datenbank, und sowohl VM als auch vFPGA werden gestartet. Innerhalb der lokalen Knoten-Verwaltung und dem RC2F Host-Hypervisor werden die erforderlichen Kommunikationskanäle zum vFPGA an die VM entsprechend Abschnitt 4.4.6 durchgereicht und die Initialisierung des vFPGAs kann über den Konfigurationsspeicher des FPGA-Hypervisors (HCS) erfolgen, ebenso wie die Rekonfiguration der vFPGA-Slots (C).

(II) Starten eines vRAI-Paketes: Das in Abbildung 5.4 skizzierte Starten eines vRAI-Pakets kann direkt aus einer VM erfolgen, wobei in der skizzierten Sequenz bereits ein vFPGA allokiert wurde. Über die Anwendung des Nutzers innerhalb einer VM (B) kann der vFPGA mit einem vFPGA-Image für die entsprechenden vFPGA-Slots konfiguriert werden, und die Interaktion über Datenkanäle und den Konfigurationsspeicher des vFPGAs (vCS) kann erfolgen (C).

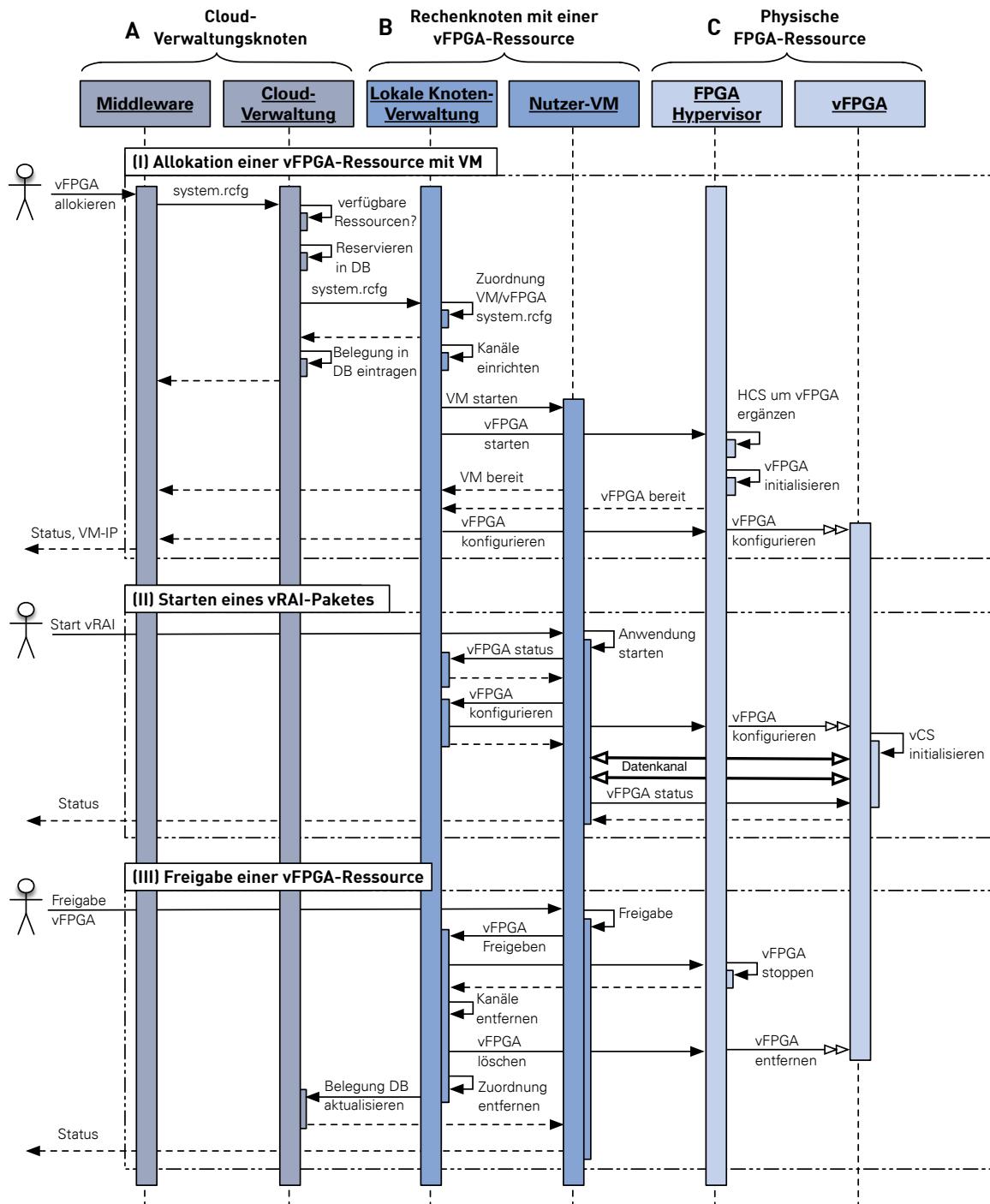


Abbildung 5.4: Sequenzdiagramm mit Interaktion der Ebenen im RC3E-System über Cloud-Verwaltungsknoten, dem Rechenknoten mit einer (freien) vFPGA-Ressource bis hin zum physischen FPGA für drei exemplarische Szenarien.

(III) Freigabe einer vFPGA-Ressource: Bei der Freigabe eines vFPGAs, welche ebenfalls über die VM möglich ist (B), wird der vFPGA zunächst über den FPGA-Hypervisor gestoppt (C), die Kanäle zwischen vFPGA und VM innerhalb des RC2F Host-Hypervisors werden entfernt (B) und der vFPGA wird mit seinem vFPGA-Image für die entsprechende Region physisch gelöscht (C), damit Nutzerdaten nicht durch den nächsten Nutzer ausgelesen werden können. Anschließend wird die neue Zuordnung der Ressourcen des Nutzers an die zentrale Cloud-Verwaltung übermittelt (A).

Die Zuteilung von vFPGAs zu den entsprechenden physischen Hardwareressourcen und vFPGA-Slots stellt ein typisches Scheduling-Problem dar, wie es auch neben der Informatik in vielen anderen Bereichen wie beispielsweise in der Betriebswirtschaftslehre auftritt. In [Bru01] werden unterschiedlichste Algorithmen vorgestellt, welche zum Teil eine NP-vollständige Komplexität [GJ79] besitzen. Beispiele sind das Scheduling von Prozessen im Betriebssystem, aber auch zur Planung von Transaktionen in Datenbankverwaltungssystemen. Das hier angesprochene Scheduling ist vergleichbar mit dem Job-Scheduling, wie es in großen IT-Systemen beispielsweise bei der Abarbeitung von Batchjobs [Bru+98] oder auch innerhalb von Cloud-Systemen eingesetzt wird [BC11]. Die Voraussetzungen für ein Scheduling von vFPGA-Ressourcen werden in Abschnitt 5.2.4 näher betrachtet.

5.2.3 Erweiterung des Host-Hypervisors innerhalb der Rechenknoten

Die Rolle des Hypervisors innerhalb der Rechenknoten sowie der FPGAs wird in Abschnitt 4.3.7 als Entwurfsraum aufgezeigt. Dieser Abschnitt geht konkret auf die Aufgaben ein, welche bei der Diskussion des Entwurfsraums sowie im vorherigen Abschnitt vom Ablauf her skizziert wurden:

- Verwaltung der lokalen Ressourcen hinsichtlich deren Auslastung, Größe sowie konkreter Zuordnung zum Nutzer.
- Starten der VM und Konfiguration der FPGAs in Abhängigkeit vom Servicemodell:
 - vFPGA-Konfiguration auf zulässige FPGA-Slots validieren und über den FPGA-Hypervisor durchführen,
 - Einrichten der Kommunikationskanäle zwischen VMs und vFPGAs (Abschnitt 4.4.6) und
 - Konfigurationsspeicher (HCS) des FPGA-Hypervisors entsprechend der Daten aus der RCFG-Datei ergänzen (Abschnitt 5.1.4).
- Administrieren der Zustände von sowohl VM als auch vFPGA-Instanzen.
- Validieren und Weiterleiten der Befehle von der VM (RC2F-API) an den vFPGA.
- Stoppen der Ressourcen:
 - Signale zum Stoppen an sowohl die VM als auch den vFPGA senden,
 - Entfernen der Kommunikationskanäle zwischen VMs und vFPGAs,
 - Konfigurationsspeicher des FPGA-Hypervisors (HCS) zurücksetzen und
 - vFPGA-Region vollständig entfernen (leere partielle vFPGA-Konfiguration).
- Zusätzliche Schritte für ein Anhalten oder Migrieren der vFPGA-Instanz:
 - Kontrolliertes Anhalten der Datentransfers (FIFO-Interfaces) über die RC2F-API und den FPGA-Hypervisor,
 - Auslesen des vollständigen FPGA-Kontextes mit entsprechendem DDR-Speicherbereich auf dem FPGA-Board und
 - Stoppen von sowohl vFPGA als auch VM.

5.2.4 Auslastung der vFPGAs im adaptiven elastischen Scheduling

In einer Cloud-Architektur, wie sie in Abschnitt 2.3 beschrieben wird, stellt die Elastizität ein wesentliches Kriterium dar. Wenn zusätzlich die FPGA-Ressourcen sowohl horizontal als auch vertikal skaliert werden können (siehe Abschnitt 2.3.2.2), stellt dies besondere Herausforderungen an ein Scheduling. Insbesondere bei größeren Mengen von Ressourcen und potentiell zahlreichen Endnutzern von Diensten, deren genaue Anzahl nicht vorhersehbar ist, ist Elastizität notwendig, um eine effiziente Auslastung des Systems zu erreichen.

Die Zuteilung der Ressourcen erfolgt dabei mittels Scheduling, wobei es notwendig ist, möglichst schnell eine Entscheidung zu treffen, um trotz der unterschiedlichen Ressourcen innerhalb eines Knotens ein SLA (siehe Abschnitt 2.3.2.3) einhalten zu können. Das Scheduling hat daher nicht das Ziel, eine optimale Zuordnung zu finden, sondern lediglich die Ressourcen annähernd optimal auszulasten. In der Arbeit von Proaño et al. [PCC16] werden die Arbeitspakete für die FPGAs einfach auf die kompletten FPGAs verteilt, wobei durch vorher bekannte Laufzeiten das Scheduling einen anderen Schwerpunkt aufweist. Andere Arbeiten mit virtualisierten FPGAs, wie [Che+14; Wee+15], nutzen, wenn sie sich mit der Ressourcenverteilung beschäftigen, nur vFPGAs identischer Größe.

Um vFPGAs unterschiedlicher Größe möglichst effizient einem physischen System zuordnen zu können, muss die Ressourcenzuteilung in zwei Stufen durchgeführt werden. In der ersten Stufe wird dabei ein Rechenknoten mit ausreichenden freien Ressourcen ausgewählt, in der zweiten Stufe schließlich wird der konkrete vFPGA zugewiesen, wie auch bereits im vorherigen Abschnitt erläutert wurde.

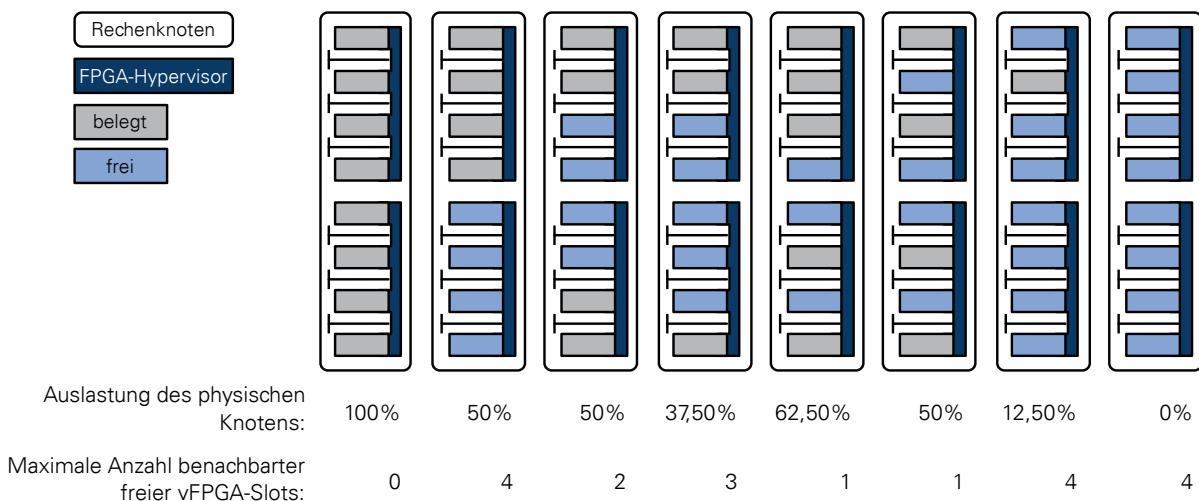


Abbildung 5.5: Berechnung der Auslastung für einen physischen FPGA mit 4 vFPGAs für unterschiedliche Belegungen.

Eine weiterer zu beachtender Aspekt besteht in dem Umstand, dass über einen längeren Zeitraum eine interne Fragmentierung des physischen FPGAs durch die vFPGAs entstehen kann, ähnlich wie bei der Speicherverwaltung³ aus dem Bereich der Betriebssysteme [TB14]. Um diese Fragmentierung in eine einfach zu verarbeitende Größe zu überführen, ist für das angestrebte Scheduling neben der Auslastung die Anzahl benachbarter freier vFPGA-Slots als Parameter erforderlich. Abbildung 5.5 zeigt ein Beispiel eines Systems mit zwei FPGAs pro Rechenknoten, die wiederum je vier vFPGAs aufnehmen können. Des Weiteren zeigt die Abbildung die beiden Parameter für die Auslastung der FPGAs innerhalb

³Die relevanten Techniken aus dem Bereich der Speicherverwaltung sind hierbei *Multiprogramming with a Fixed number of Tasks (MFT)*, beziehungsweise *Multiprogramming with a Variable number of Tasks (MVT)* [TB14].

5.3 Aufbau einer Umgebung aus virtuellen Komponenten für unterschiedliche Szenarien

des Rechenknotens, sowie die maximale Anzahl benachbarter freier vFPGA-Slots. Da die Parameter Auskunft über die Belegung des kompletten Rechenknotens geben, ist eine Entscheidung für einen Knoten im Scheduling auf oberster Ebene der Cloud-Verwaltung möglich, ohne die genauen vFPGA-Slots und deren Belegungen innerhalb der FPGAs zu kennen.

Für ein entsprechendes Scheduling kann ein Algorithmus dienen, der innerhalb der Cloud-Verwaltung die Liste der verfügbaren vFPGAs durchsucht, wobei das Ziel in einer möglichst hohen Auslastung besteht. Die Entscheidung erfolgt auf Basis der zuvor eingeführten Kennzahl zur Beschreibung der Auslastung und der benachbarten freien vFPGA-Slots innerhalb eines Rechenknotens. Die für die Entscheidung erforderlichen Daten liegen in der Datenbank der Cloud-Verwaltung, und der Zugriff auf die detaillierte Platzierung der vFPGAs innerhalb des Rechenknotens ist nicht erforderlich. Die Auslastung sowie der größtmögliche vFPGA sollen dabei auf möglichst einfache Weise beschrieben werden. Die spätere konkrete Platzierung der vFPGAs erfolgt über den Host-Hypervisor des Rechenknotens, welcher den physischen FPGA enthält.

5.3 Aufbau einer Umgebung aus virtuellen Komponenten für unterschiedliche Szenarien

In diesem Abschnitt werden die bisherigen Erkenntnisse und Konzepte genutzt, um die Beschreibung einer virtuellen Systemarchitektur auf Basis des in Abschnitt 5.1 gezeigten Entwurfsprozesses innerhalb der in Abschnitt 5.2 aufgezeigten Cloud-Verwaltung zu ermöglichen und einordnen zu können. Dazu wird zunächst im Folgenden in Abschnitt 5.3.1 aufgezeigt, wie eine virtuelle Systemarchitektur auf die zugrundeliegende physische Systemarchitektur der Cloud übertragen werden kann. Anschließend wird für drei exemplarische Anwendungsszenarien die Beschreibung der virtuellen Architekturen zum Aufbau unterschiedlicher Systeme in den Abschnitten 5.3.2, 5.3.3 und 5.3.4 veranschaulicht.

5.3.1 Abbildung der virtuellen Systembeschreibung auf die physische Cloud-Architektur

Ein wichtiger Aspekt der zugrundeliegenden Motivation dieser Arbeit wird durch die in Abschnitt 3.1 und Abschnitt 3.2 gezeigte Abbildung eines virtuellen Systems auf die physischen Hardwareressourcen verdeutlicht. Die Beschreibungen des vom Nutzer gewünschten virtuellen Systems werden in Form der RCFG-Datei (siehe Abschnitt 5.1.4) eingelesen, analysiert und schließlich von der Cloud-Verwaltung allokiert, entsprechend konfiguriert und initialisiert.

Für den Nutzer existieren entsprechend der drei Servicemodelle aus Abschnitt 3.1.2 unterschiedliche Sichtweisen auf ihre Ressourcen, wobei die Modelle RAaaS und Reconfigurable Silicon as a Service (RSaaS) immer vollständige physische FPGAs erfordern. Für das Modell Background Acceleration as a Service (BAaaS) existiert aus Sicht der Anbieter eines Dienstes lediglich eine Funktion, welche aufgerufen und in Form einer vRAI ausgeführt wird (siehe Abschnitt 5.1.6). Durch die vRAI wird ebenso ein virtuelles System aufgebaut, wobei im Gegensatz zu den Modellen RAaaS und RSaaS die effiziente Auslastung der FPGA-Ressourcen durch unterschiedliche Nutzer im Vordergrund steht.

Das Ziel der Abbildung des virtuellen auf das physische System besteht darin, zusammengehörige vFPGA-Allokationen möglichst auch auf demselben physischen FPGA beziehungsweise Rechenknoten

zu instanzieren. Zu diesem Zweck wird zunächst die Anzahl der erforderlichen physischen FPGAs ermittelt, indem:

1. Die Liste der benötigten vFPGA-Slots `size` in absteigender Reihenfolge sortiert wird,
2. Eine Liste für erforderliche physische FPGAs angelegt wird,
3. Die vFPGAs auf physische FPGAs mit ausreichend freien Ressourcen (vFPGA-Slots `size` und externer Speicherbedarf `memory`) verteilt werden und
4. Neue physische FPGAs der Liste hinzugefügt werden, sobald keine freien benachbarten vFPGA-Slots oder externer Speicher auf den bisherigen physischen FPGAs verfügbar sind.

Das Ergebnis ist eine Zuordnung der vom Nutzer allokierten vFPGAs zu den erforderlichen physischen FPGAs. Ist die Beschreibung innerhalb der vRAI nicht auf die verfügbare Hardware übertragbar, wird eine entsprechende Fehlermeldung ausgegeben. Mit Hilfe der zuvor in Abschnitt 5.2.4 beschriebenen Kennzahlen zur realen Systemauslastung können die benötigten vFPGAs innerhalb der Cloud-Verwaltung auf die physische Cloud-Architektur übertragen werden. Im Folgenden wird der Aufbau der einzelnen Konfigurationsdateien für drei ausgewählte Anwendungsszenarien exemplarisch skizziert:

- Hardwarebeschleuniger für Cloud-Dienste (BAaaS),
- Virtueller FPGA-Cluster (RAaaS) und
- FPGAs zur Ver- und Entschlüsselung im Datenstrom (RAaaS).

Ein spezielles Anwendungsszenario für physische FPGAs ohne Virtualisierung innerhalb des Modells RSaaS erfolgt analog zu den aufgezeigten Szenarien.

5.3.2 Hardwarebeschleuniger (BAaaS)

Listing 5.4 zeigt eine exemplarische RCFG-Datei für die Verwendung eines vFPGAs innerhalb des Dienstes BAaaS als einfacher Hardwarebeschleuniger, welcher direkt einer VM zugeordnet ist. Der Schwerpunkt liegt dabei auf der einfachen Kopplung zwischen der vFPGA-Instanz `name = ['vfpca-bsmc']` und der VM `vm = ['vml-pvm']`, wie sie für eine Hintergrundbeschleunigung nach Abschnitt 2.1.2.3 üblich ist. Das Beispiel allokiert innerhalb des Modells BAaaS einen einzelnen vFPGA `vfpca = [1]`, bestehend aus zwei vFPGA-Slots `size = [2]`. Sobald die Ressource bereitsteht, wird diese mit dem erforderlichen vFPGA-Image für die zugewiesenen vFPGA-Slots aus der vRAI `design = ['bsmc.vrai']` konfiguriert und in den Wartezustand `boot = ['idle']` versetzt. Das Starten des dazugehörigen Host-Programms innerhalb der VM ist dabei Aufgabe des Nutzers innerhalb des übergeordneten Cloud-Servicemodells.

Listings 5.4: RCFG-Datei für einen FPGA-basierten Hardwarebeschleuniger mit VM.

```

service = 'ba'                                     #Service Model BAaaS
name = ['vfpca-bsmc']                            #vFPGA/User Design Name
vm = ['vml-pvm']                                 #VM-Instance Name

vfpca = [1]                                       #Number of vFPGAs
size = [2]                                        #vFPGA-Slots
memory = [4000]                                   #DDR-Memory Size

boot= ['idle']                                    #Initial vFPGA-State
design = ['bsmc.vrai']                           #Initial Design

```

5.3.3 FPGA-Cluster (RAaaS)

Der Aufbau eines FPGA-Clusters kann für speziellere Anwendungen mit stärkerem Fokus auf der Entwicklung komplexerer Systeme innerhalb des Modells RAaaS, aber auch in der Hintergrundbeschleunigung BAaaS erfolgen. In Listing 5.5 wird beispielsweise der Aufbau eines FPGA-Clusters `name = ['cluster1']` innerhalb des Modells RAaaS mittels der eingeführten RC2F-Virtualisierung gezeigt. Entscheidend für den Aufbau des Systems ist die Reihenfolge innerhalb der Felder `vm`, `vfpga`, `size` und `memory`, da die Einträge untereinander entsprechend dieser Reihenfolge zugeordnet werden. Das entsprechende System besteht dabei aus zwei VMs `vm = ['vm1', 'vm2']`, wobei zwei vFPGAs an `vm1` und vier vFPGAs an `vm2` durchgereicht werden (`vfpga = [2, 4]`). Die vFPGAs belegen dabei eine unterschiedliche Anzahl von vFPGA-Slots `size = [6, 2]`. Da keine vFPGA-Images angegeben sind, muss die Konfiguration der vFPGAs manuell über die Cloud-Verwaltung oder die RC2F-API erfolgen.

Für den Aufbau eines komplexeren FPGA-Systems ist die Kenntnis der physischen System-Architektur der Rechenknoten notwendig, da nicht jede Beschreibung einer virtuellen Architektur auf das physische System abgebildet werden kann. Beispielsweise können an eine VM nur so viele vFPGAs durchgereicht werden, wie maximal auf den physischen FPGAs des Host-Systems instanziert werden können. Ist das in einer RCFG-Datei beschriebene System nicht realisierbar, wird die Cloud-Verwaltung die Allokation der entsprechenden Ressourcen mit einer Fehlermeldung abbrechen und vollständig verwerfen.

Listings 5.5: RCFG-Datei für einen FPGA-Cluster mit zwei Host-VMs und zugeordneten vFPGAs.

```

service = 'ra'                                     #Service Model RAaaS
name = ['cluster1']                                #vFPGA/User Design Name
vm = ['vm1', 'vm2']                                 #VM-Instance Name

vfpga = [2, 4]                                     #Number of vFPGAs
size = [6, 2]                                      #vFPGA-Slots

boot= ['idle']                                     #Initial vFPGA-State

```

Der Aufbau eines FPGA-Clusters, welches für spezielle Anwendungsfälle optimiert werden soll, ist unter Umständen für das Modell RSaaS besser geeignet, da eigene Kommunikations-Schnittstellen und spezielle Infrastruktur auf der bereitgestellten FPGAs realisiert werden können. Im Modell RSaaS ist aber die FPGA-Virtualisierung RC2F optional und fällt vollständig in den Verantwortungsbereich des Nutzers.

5.3.4 Direkter Zugang über das Netzwerk (RAaaS)

Ein Anwendungsfall aus dem Bereich der sicherheitskritischen Anwendungen ist der Einsatz des FPGAs als kryptographischer Coprozessor innerhalb des Modells RAaaS. Dadurch, dass sich in diesem Modell lediglich ein Nutzer auf dem physischen FPGA befindet, kann ein hohes Sicherheitslevel erreicht werden, da Einflüsse anderer Nutzer entfallen. Konkrete Anwendungen können dabei die direkte Ver- und Entschlüsselung des Datenverkehrs über den FPGA als sicheren Zugangspunkt zwischen der Außenwelt und der VM des Anwenders, aber auch eine Anonymisierung der Nutzerdaten sein. Listing 5.6 zeigt exemplarisch einen entsprechenden Aufbau einer vFPGA-Konfiguration. Die Anwendung ist entsprechend klein und erfordert lediglich einen vFPGA-Slot, welchem eine feste IP durch `vif = ['ip=10.0.0.43']` zugeordnet ist.

Wichtig bei dieser Anwendung ist der öffentliche Schlüssel innerhalb der RCFG `key = ['...']`, welcher über Host-/FPGA-Hypervisor initialisiert wird. Die weiteren geheimen Schlüssel müssen in der

5 Entwurfsprozess und Verwaltungshierarchie der Cloud

verschlüsselten vFPGA-Instanz, wie in Abschnitt 4.4.8 diskutiert, eingebettet werden, um ein höheres Sicherheitslevel zu erreichen. Die Realisierung der Flusskontrolle und des Aufbaus der Pakete auf den höheren Open Systems Interconnection (OSI)-Schichten sowie die Bereitstellung der eigentlichen Anwendung ist Aufgabe des Entwicklers beziehungsweise des Anbieters des Dienstes. Der vFPGA kann auch mithilfe der Einbettung der RC2F-API in einen Treiber als Netzwerkkarte fungieren, was ebenfalls im Verantwortungsbereich des Entwicklers liegt und hier nicht weiter betrachtet wird.

Listings 5.6: RCFG-Datei für den Einsatz des vFPGA zwischen Netzwerk und VM.

```
service = 'ra'                                #Service Model RAaaS
name = ['secure-endpoint']                    #vFPGA/User Design Name
vm = ['vml-pvm']                             #VM-Instance Name

vfga = [1]                                    #Number of vFPGAs
size = [1]                                    #vFPGA-Slots
vif = ['ip=10.0.0.43']                         #vFPGA-IPs

key = ['AAAAABC1yc2 ... Buay74HNE']          #User AES-Key
design = ['secure.bit']                        #Initial Design

boot = ['run']                                 #Initial vFPGA-State after config
```

Neben dem Zugang zur Cloud über den vFPGA ist ebenso der Zugang zu einem Network Attached Storage (NAS) oder einer Datenbank in der Cloud möglich. Dabei kann der FPGA zum Beispiel sämtliche Daten ver- und entschlüsseln, damit die Klartextinformationen nur innerhalb der VM sichtbar sind. Für ein noch größeres Maß an Sicherheit ist ebenfalls eine komplette homomorphe Verschlüsselung möglich, bei welcher der FPGA die Operationen direkt in Hardware mit hoher Effizienz durchführt oder andererseits der FPGA die Klartextinformationen einsehen kann. Ein solcher Ansatz benötigt unter Umständen zusätzliche Modifikationen der FPGA-Architektur, um die vertrauenswürdige Spezialhardware dem Nutzer vollständig zugänglich zu machen. Beim aktuellen Stand der Technik wäre dazu der Einsatz einer TA erforderlich, wie auch bereits in Abschnitt 4.4.8 und Abschnitt 5.1.7 erläutert.

6 Prototypische Implementierung und Ergebnisse

Die in Kapitel 4 vorgestellte Virtualisierung von FPGAs – Reconfigurable Common Computing Frame(work) (RC2F) – und deren Eingliederung in eine Architektur zur Verwaltung – Reconfigurable Common Cloud Computing Environment (RC3E) –, die in Kapitel 5 skizziert wurde, werden in diesem Kapitel prototypisch implementiert und evaluiert.

In Abschnitt 6.1 werden zunächst die Implementierung des Cloud-Prototypen RC3E, die Modellierung des RC3E-Simulators zur Evaluation eines größeren Cloud-Systems sowie der FPGA-Virtualisierung RC2F vorgestellt. Die Virtualisierung RC2F wird dazu genutzt, um den Bedarf an FPGA-Ressourcen, Systemparameter sowie die Eignung einer aktuellen FPGA-Architektur für das Konzept der Virtualisierung bewerten zu können. Abschnitt 6.2 stellt Demonstratoren auf Basis der vFPGAs vor. In einem darauf aufbauenden Anwendungsszenario für die FPGA-Virtualisierung wird der Einsatz in einer produktiven Cloud-Umgebung skizziert. Anschließend wird in Abschnitt 6.3 an den Demonstratoren der Einsatz der virtualisierten FPGAs evaluiert und das System validiert. Abschließend werden in Abschnitt 6.4 die entwickelten Prototypen bewertet und mit anderen Arbeiten verglichen.

6.1 Prototypische Implementierung und Aufbau einer Cloud

Zunächst zeigt der folgende Abschnitt 6.1.1 den Testaufbau des Cloud-Prototypen auf. Der prototypischen Aufbau der Cloud-Ressourcenverwaltung RC3E wird in Abschnitt 6.1.2 beschrieben. Abschnitt 6.1.3 zeigt die Modellierung des RC3E-Simulators zur Evaluation des Verhaltens der Cloud-Verwaltung für ein größeres Cloud-System auf. Die eigentliche Virtualisierung der FPGAs wird in Abschnitt 6.1.4 prototypisch umgesetzt, wobei sich das Hauptaugenmerk auf die Auslastung der Ressourcen sowie die Kosten der Bereitstellung homogener vFPGAs richtet.

6.1.1 Hardwareaufbau des Cloud-Prototypen zur Integration von FPGAs in eine Cloud-Architektur

Der Hardwareaufbau des Cloud-Prototypen zur Integration von FPGAs in eine Cloud umfasst einen Verwaltungsknoten und zwei Rechenknoten, wie in Abbildung 6.1 aufgezeigt. Die eigentliche Cloud-Architektur ist entsprechend der Anforderungsanalyse aus Abschnitt 3.3.1 aufgebaut. Auf dem Cloud-Verwaltungsknoten befindet sich die RC3E-Ressourcenverwaltung. Auf den Rechenknoten werden die lokale Knoten-Verwaltung sowie der RC2F Host-Hypervisor zur Überwachung der physischen und virtuellen FPGA-Ressourcen ausgeführt (siehe Abbildung 5.3). Über den Verwaltungsknoten kann über das interne Verbindungsnetzwerk der Cloud (Gigabit-Ethernet) auf die Rechenknoten sowie die FPGA-Boards zugegriffen werden.

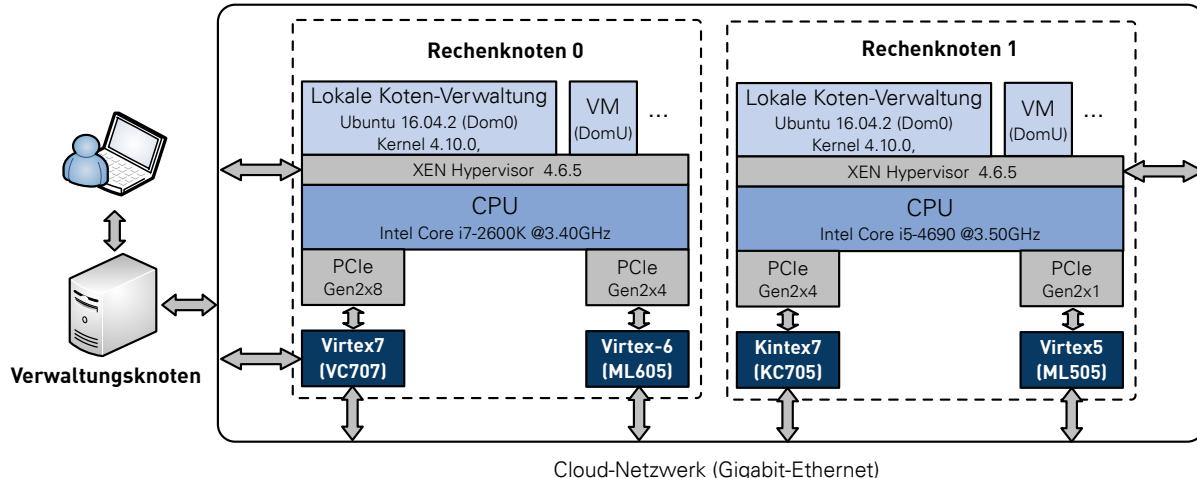


Abbildung 6.1: Aufbau des Cloud-Prototypen zur Evaluation der Ressourcenverwaltung RC3E mit einem Verwaltungsknoten und zwei Rechenknoten, welche je zwei physische FPGAs beinhalten.

In den beiden Rechenknoten des Cloud-Prototypen befinden sich je zwei eng über PCIe gekoppelte FPGAs des Hersteller Xilinx, welche für eine Reihe von Projekten und studentischen Arbeiten im Rahmen der Servicemodelle RSaaS oder RAaaS eingesetzt werden:

Rechenknoten 0: (Intel Core i7-2600K):

- VC707 Evaluation Kit – Virtex-7 XC7VX485T-2FFG1761C, PCIe Gen2x8 [Xil16f].
- ML605 Evaluation Kit – Virtex-6 XC6VLX240T-1FFG1156, PCIe Gen2x4 [Xil10a].

Rechenknoten 1: (Intel Core i5-4690):

- KC705 Evaluation Kit – Kintex-7 XC7K325T-2FFG900C, PCIe Gen2x4 [Xil16e].
- ML505 Evaluation Platform – Virtex-5 XC5VSX50-TFFG1136, PCIe Gen2x1 [Xil11].

Der für die Virtualisierung von FPGAs innerhalb des Modells BAaaS relevante FPGA ist ein Virtex-7 XC7VX485T [Xil17a] auf einem VC707 Evaluation Kit [Xil16f]. Der Virtex-7 wird nachfolgend in Abschnitt 6.1.4 genutzt, um das RC2F-System prototypisch zu realisieren und zu evaluieren.

6.1.2 Prototypischer Aufbau der Cloud-Ressourcenverwaltung – RC3E

Die Ressourcenverwaltung RC3E, wie sie in Abschnitt 5.2 beschrieben wird, wurde als Prototyp entwickelt, um die FPGA-spezifischen Komponenten für die angestrebte Virtualisierung analysieren und in Form einer Machbarkeitsstudie evaluieren zu können. Die in Abschnitt 5.2.1 aufgezeigte System-Architektur der Cloud-Ressourcenverwaltung ist entsprechend, wie in Abbildung 5.3 gezeigt, prototypisch umgesetzt¹. Die Komponenten sind im Einzelnen ein Nutzer-Frontend auf Kommandozeilebene, die Cloud-Verwaltung auf dem Verwaltungsknoten sowie die darin eingebetteten Komponenten zum Monitoring und zum Scheduling der eingehenden Anfragen und Arbeitspakete (siehe Abschnitt 5.2.4).

Auf der Ebene der Rechenknoten ist ein *System-Hypervisor* (siehe Definition 3.6) zur Virtualisierung des Betriebssystems und zur Bereitstellung der paravirtualisierten VMs (DomU) für die Nutzerumgebungen innerhalb der Rechenknoten erforderlich. Als System-Hypervisor wird der Typ 1-Hypervisor Xen

¹Die Implementierung der einzelnen Teilkomponenten des RC3E erfolgt in Python 3.5.x [Fou17].

[Bar+03] (siehe auch Abschnitt 2.2.3.1) eingesetzt. Innerhalb der privilegierten Domain (Dom0) wird die lokale Knoten-Verwaltung als System-Dienst ausgeführt, welcher sich nach Systemstart bei der Cloud-Verwaltung registriert. Die lokale Knoten-Verwaltung nutzt den RC2F Host-Hypervisor, um die FPGA-Ressourcen für die Nutzer zu administrieren und bereitzustellen, wie in Abschnitt 5.2.3 beschrieben. Die Abläufe zur Allokation, Interaktion und Freigabe der Ressourcen sind wie in Abschnitt 5.2.2 dargestellt prototypisch implementiert.

Das Durchreichen der vFPGA-Ressourcen, wie es Abschnitt 4.4.6 beschreibt, erfolgt mittels einer Inter-Domain Kommunikation auf Basis von vChan² [Zha+07]. Die Zuordnung der Kommunikationskanäle zwischen den VMs und den vFPGAs, wie sie in Abbildung 4.14 dargestellt sind, erfolgt über die lokale Knoten-Verwaltung innerhalb der privilegierten Domain (Dom0), um einen vollen Zugriff auf die Geräte zu gewährleisten. Für das Modells RSaaS läuft das Durchreichen der vollständigen FPGAs über PCI-Passthrough ab³. Zur Messung des Durchsatzes werden aufgrund der Einbußen bei der Inter-Domain Kommunikation keine VMs eingesetzt, um entsprechend die Leistungsfähigkeit der reinen FPGA-Virtualisierung zu demonstrieren.

6.1.3 Modellierung des RC3E-Simulators

Der zuvor vorgestellte Prototyp der Ressourcenverwaltung RC3E ermöglicht die Integration virtualisierter FPGAs in den in Abschnitt 6.1.1 vorgestellten Cloud-Prototypen. Um das Verhalten der Cloud-Verwaltung und insbesondere die Lastverteilung (siehe Abschnitt 5.2.4) der eingehenden Arbeitspakete auf die Ressourcen in einem größeren Cloud-System untersuchen zu können, wurde der *RC3E-Simulator* entwickelt [Kan15; KLS16]. Speziell die Eignung der aufgezeigten prototypischen FPGA-Virtualisierung RC2F für den Einsatz in einem horizontal skalierbaren Cloud-System mit mehreren Rechenknoten wird dabei unter verschiedenen Lastbedingungen evaluiert. Die virtualisierten FPGAs können dabei unterschiedliche Größen auf dem physischen FPGAs einnehmen, wie es innerhalb der RC2F-Virtualisierung ermöglicht wird.

Die wesentlichen Fragestellungen sind dabei, ob und wie stark die Menge der aktiven Ressourcen durch Virtualisierung optimiert werden kann, und inwiefern der Energiebedarf der Cloud durch den Einsatz virtualisierter FPGAs reduziert werden kann. Neben der Virtualisierung selbst wird auch das Konzept der Migration von vFPGAs zur systemweiten Optimierung der Auslastung evaluiert.

6.1.3.1 Aufbau des Simulators

Der Aufbau des Simulators orientiert sich an dem zuvor in Abschnitt 6.1.2 vorgestellten Prototypen der Cloud-Ressourcenverwaltung mit den Komponenten der Cloud-Verwaltung, der Lastverteilung und der lokalen Knoten-Verwaltung. Untersucht wird dabei im Speziellen die Zuordnung von vFPGAs zu Rechenknoten und auf zu auf diesen verfügbaren vFPGA-Slots, wie sie in Abschnitt 5.2.4 eingeführt wurde. Dabei spielt die Ermittlung sowohl der Auslastung des physischen Rechenknotens als auch der maximalen Anzahl benachbarter freier vFPGA-Slots eine zentrale Rolle. Ein wesentliches Ziel besteht darin, sowohl den Nutzen der virtualisierten FPGAs wie auch deren Migration in einer Cloud zu evaluieren. Die durch die Cloud-Prototypen ermittelten Leistungsdaten bilden dabei die Basis, um das entsprechende Verhal-

²Durch die Erweiterung XVMSocket für XEN 4.6.5 [ZGR15] kann vChan innerhalb des Cloud-Prototypen eingesetzt werden.

³Für den späteren Einsatz von vFPGAs in einer Cloud ist für die Servicemodelle RAaaS und BAaaS eine Single Root I/O-Virtualisierung für PCIe 3.0-Geräte [San+14a] des System-Hypervisors zu bevorzugen, da keine Einbußen in Latenz und Datenrate auftreten [San+14b].

6 Prototypische Implementierung und Ergebnisse

ten innerhalb einer größeren Cloud mit mehreren Rechenknoten durch den Simulator nachzubilden und zu untersuchen.

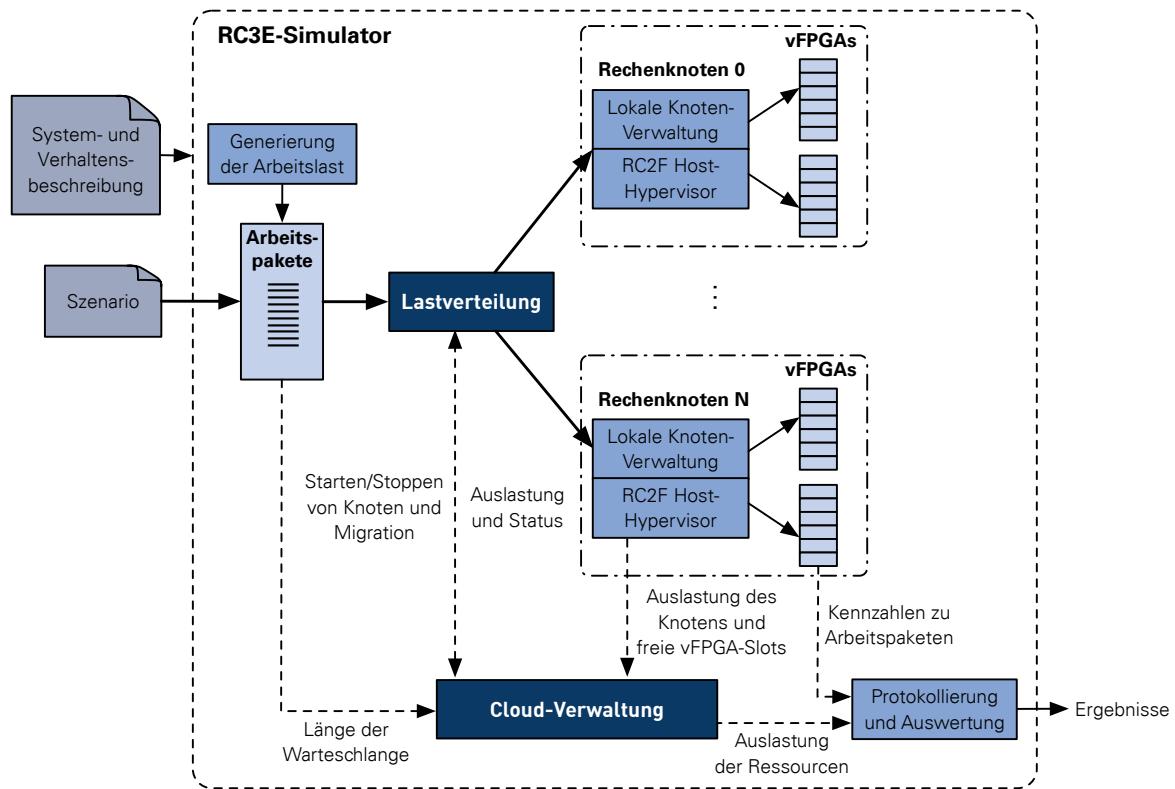


Abbildung 6.2: Aufbau des RC3E-Simulators zur Evaluation der Lastverteilung innerhalb der Cloud-Verwaltung für eine Cloud mit virtualisierten FPGAs und deren Migration.

Bei der RC3E-Simulation handelt es sich um eine ereignisgesteuerte Simulation [Lie95, S. 9 ff.], welche die generierten oder aus einer externen Datei eingelesenen Arbeitspakete entsprechend der Lastszenarien in einer Warteschlange bereitstellt oder über einen Generator innerhalb des Simulator erzeugt. Der grundlegende Aufbau des RC3E-Simulators⁴ ist in Abbildung 6.2 dargestellt.

Die Arbeitspakete werden vom Simulator in eine zentrale Warteschlange eingereiht, und die Lastverteilung des RC3E (siehe Abschnitt 6.1.2) verteilt die Arbeitspakete an die simulierten Rechenknoten mit der lokalen Knoten-Verwaltung. Diese weist über einen simulierten RC2F Host-Hypervisor den Arbeitspaketen Positionen in einem Vektor zu, welche in der Simulation den eigentlichen vFPGA-Slots entsprechen. Entsprechend der Auslastung der Rechenknoten und Informationen über die eingehenden Arbeitspakete werden von der Cloud-Verwaltung weitere Rechenknoten hinzugefügt beziehungsweise (bei geringer Last) entfernt, um eine elastische Cloud simulieren zu können. Die einzelnen Systeme (Verwaltungs- und Rechenknoten) sind dabei als Prozesse modelliert und kommunizieren über feste Schnittstellen miteinander. Ein Arbeitspaket, welches verarbeitet wird, entspricht in der Simulation einem Thread, welcher pausiert wird, um die Rechenzeit zu simulieren. Die Zeitbasis des Simulators wird über das Verhältnis von realer zu Simulationszeit angegeben.

⁴Die Implementierung des Simulators erfolgt in Python 3.5.x [Fou17].

6.1.3.2 Parameter zur Anpassung des Simulators

Anhand von Parametern und Kenngrößen erfolgt die Beschreibung des zu simulierenden Cloud-Systems und das Verhalten des Simulators. Der Simulator kann dadurch an unterschiedliche Anwendungsfälle angepasst werden, um bestehende oder zukünftige Cloud-Architekturen zu evaluieren. Die wichtigsten Parameter zur Systembeschreibung sind dabei:

- Die Maximal Anzahl der:
 - Nutzbaren Rechenknoten Max_{Knoten} ,
 - (Virtuellen) Prozessorkerne innerhalb eines Rechenknotens Max_{Kerne} ,
 - Physischen FPGAs innerhalb eines Rechenknotens Max_{FPGAs} und
 - vFPGA-Slots innerhalb eines physischen FPGAs $Max_{vFPGA-Slots}$.
- Die Leistungsaufnahmen:
 - Eines Rechenknotens unter Volllast $P_{CPU-Last}$ und im Leerlauf $P_{CPU-Leerlauf}$ in Watt, sowie
 - Eines FPGAs unter Volllast $P_{FPGA-Last}$ und ohne Last $P_{FPGA-Leerlauf}$ in Watt.
- Länge der Warteschlange $N_{Warteschlange}$ für eingehende Arbeitspakete,
- Anzahl wartender Arbeitspakete $N_{Waiting}$ bis neue Rechenknoten aktiviert werden,
- Zeitdauer zum Aktivieren eines weiteren Rechenknotens $t_{Knoten-Start}$,
- Auslastung eines Rechenknotens ρ_{Knoten} und Zeitdauer zum Deaktivieren des Rechenknotens $t_{Knoten-Stop}$,
- Zeitspanne $t_{Migration}$ und Auslastung des physischen FPGAs ρ_{FPGA} bis zum Migrieren von vFPGA-Ressourcen auf andere Rechenknoten innerhalb der Zeitspanne $t_{Migration-FPGA}$.

Wird die Anzahl der vFPGA-Slots auf 1 gesetzt, entspricht das einem System ohne Virtualisierung der FPGAs. Innerhalb des Simulators wird die Leistungsaufnahme der Rechenkerne P_{Kern} in Watt beziehungsweise der vFPGAs $P_{vFPGA-Slot}$ in Watt über Gleichung 6.1 beziehungsweise Gleichung 6.2 auf Basis der zuvor festgelegten Parameter berechnet.

$$P_{Kern} = (P_{CPU-Last} - P_{CPU-Leerlauf}) / Max_{Kerne} \quad (6.1)$$

$$P_{vFPGA-Slot} = (P_{FPGA-Last} - P_{FPGA-Leerlauf}) / Max_{vFPGA-Slots} \quad (6.2)$$

6.1.3.3 Modellierung der Arbeitspakete

Die Arbeitslasten selbst werden entweder innerhalb des Simulators generiert oder über ein Szenario in Form einer externen Datei eingelesen, in welcher jedes Arbeitspaket neben Laufzeit und Ressourcenbedarf einen Startzeitpunkt für die Übergabe an die Ressourcenverwaltung/Lastverteilung enthält. Die einzelnen Arbeitspakete innerhalb der Simulation sind angelehnt an die in Abschnitt 5.1.6 eingeführten vRAIs innerhalb des Servicemodells BAaaS. Abbildung 6.3 zeigt ein exemplarisches Arbeitspaket innerhalb des RC2F-Simulators. Die wesentlichen Kennwerte zur Beschreibung eines Arbeitspaketes sind:

- Startzeitpunkt innerhalb der Simulation zur Übergabe des Arbeitspaketes an die Warteschlange zur Ressourcenverwaltung/Lastverteilung,

6 Prototypische Implementierung und Ergebnisse

- Gesamtaufzeit $t_{Arbeitspaket}$ zur Bearbeitung eines Arbeitspakets auf vFPGA und Rechenknoten,
- Beschreibung des Ressourcenbedarfs:
 - Anzahl der erforderlichen (zusammenhängenden) vFPGA-Slots $N_{vFPGA-Slots}$ und
 - Anzahl der erforderlichen Prozessorkerne auf dem Rechenknoten N_{Kerne} .

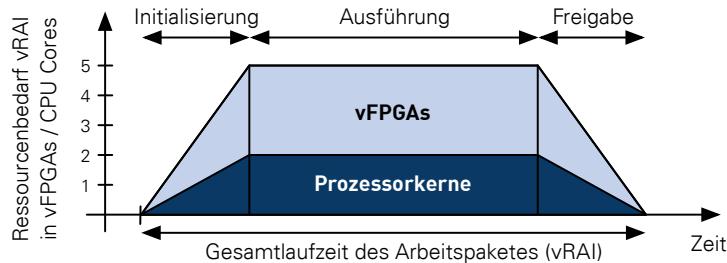


Abbildung 6.3: Aufbau eines Arbeitspakets innerhalb des RC3E-Simulators. Das exemplarische Arbeitspaket besteht aus einem vFPGA, der über fünf vFPGA-Slots reicht, und einer VM mit zwei Prozessorkernen.

Die Arbeitspakete werden für die Simulation über einen Ressourcenbedarf in Form der Anzahl der vFPGAs und Prozessorkerne, sowie über eine Gesamtaufzeit in Sekunden modelliert. Die Gesamtaufzeit setzt sich dabei zusammen aus der Zeitdauer von Initialisierung, der eigentlichen Ausführung des Rechenkernes und der Freigabe der Ressourcen.

6.1.3.4 Ergebnisse und Ausgabedaten

Im Rahmen der Simulation werden die Anzahl der benötigten Rechenknoten für die unterschiedlichen Szenarien, deren Auslastung und der sich daraus ergebende Energiebedarf berechnet, um eine prognostische Abschätzung zum Verhalten einer FPGA-Cloud geben zu können. Ein Service Level Agreement (SLA) (siehe Abschnitt 2.3.2.3 und insbesondere Definition 2.5) wird ebenso ermittelt wie die durchschnittliche Wartezeit eines Arbeitspakets, um Aussagen über die Auswirkungen der Virtualisierung auf die Güte des Cloud-Dienstes treffen zu können. Die wesentlichen Ergebnisparameter der Simulation sind:

- Anzahl der aktiven Rechenknoten der Cloud und deren Auslastung,
- Der aus der Auslastung und den allokierten Rechenknoten resultierende Energiebedarf der Cloud W_{Cloud} (in kWh) und
- Das erreichte Service Level Agreement (SLA) der Cloud auf Basis der Zeitspanne zur Bearbeitung der Arbeitspakete t_{SLA} .

Der gesamte Energiebedarf W_{Cloud} in Kilowattstunden (kWh) innerhalb des Simulationszeitraumes wird aus dem Energiebedarf aller verarbeiteten Arbeitspakete $W_{Arbeitspaket}$ und aller aktiven Rechenknoten im Leerlauf W_{Knoten} über Gleichung 6.3 berechnet. Durch den Energiebedarf der allokierten Rechenknoten im Leerlauf werden zusätzlich zu den durch Arbeitspakete anteilig genutzten Rechenknoten auch die aktiven, aber ungenutzten Knoten in der gesamten Energiebilanz berücksichtigt.

$$W_{Cloud} = \sum_{n=0}^N W_{Arbeitspaket} + \sum_{m=0}^M W_{Knoten} \quad (6.3)$$

Mit Gleichung 6.4 wird der Energiebedarf für ein einzelnes Arbeitspaket $W_{Arbeitspaket}$ berechnet. Die Gleichung orientiert sich an den zuvor in Abschnitt 6.1.3.3 beschriebenen Arbeitspaketen und ist von der Gesamtaufzeit $t_{Arbeitspaket}$, den genutzten Ressourcen ($N_{vFPGA-Slots}$ und N_{Kerne}) und deren Energiebedarf (siehe Gleichung 6.1 und Gleichung 6.2) abhängig.

$$W_{Arbeitspaket} = t_{Arbeitspaket} \cdot (N_{vFPGA-Slots} \cdot P_{vFPGA-Slot} + N_{Kerne} \cdot P_{Kern}) \quad (6.4)$$

Der Energiebedarf eines Rechenknotens im Leerlauf W_{Knoten} wird über die Zeitspanne, in welcher der Knoten aktiv ist t_{Knoten} , und der Leistungsaufnahme von Prozessor $P_{CPU-Leerlauf}$ als auch dem physischen FPGA $P_{FPGA-Leerlauf}$ im Leerlauf wird durch Gleichung 6.5 berechnet.

$$W_{Knoten} = t_{Knoten} \cdot P_{CPU-Leerlauf} + Max_{FPGAs} \cdot P_{FPGA-Leerlauf} \quad (6.5)$$

6.1.4 Realisierung eines Prototypen der FPGA-Virtualisierung – RC2F

Die prototypische Implementierung der FPGA-Virtualisierung RC2F auf einer aktuellen FPGA-Architektur dient dazu, das in Abschnitt 4 entwickelte Konzept in Form einer Machbarkeitsstudie zu evaluieren. Dazu wird zunächst die Platzierung der vFPGAs in homogenen Regionen auf dem physischen FPGA vorgenommen. Darauf aufbauend wird die in Abschnitt 4.4.1 erläuterte RC2F-Infrastruktur realisiert. Der Begriff *FPGA-Ressourcen* wird im Folgenden dazu genutzt, um die Grundbausteine (Slice Register, Slice LUTs, Block-RAM (BRAM) Tiles, DSPs) auf dem physischen FPGA zu bezeichnen.

Die für den RC2F-Prototypen eingesetzte Zielplattform (physischer FPGA) ist im folgenden ein Virtex-7 XC7VX485T [Xil17a] auf einem VC707 Evaluation Kit [Xil16f]. Das FPGA-Board befindet innerhalb des Cloud-Prototypen RC3E (siehe Abschnitt 6.1.1). Für den RC2F-Prototypen sind die genutzten Ressourcen des FPGA-Boards ein PCIe Gen2x8-Schnittstelle, ein Gigabit-Ethernet-Anschluss für den direkten Netzwerzugang und ein 1GB großer On-Board DDR3 SODIMM Speicher [Xil16d].

6.1.4.1 Anordnung der vFPGAs auf einem physischen FPGA

Die Anordnung der vFPGAs auf dem physischen FPGA selbst ist zunächst abhängig von den genutzten externen Ressourcen auf dem FPGA-Board. Abbildung 6.4 zeigt entsprechend die Positionen der für das RC2F erforderlichen I/O- und Infrastruktur-Komponenten (ICAP, BSCAN etc.). Die Platzierung der Komponenten ist durch das physische Layout des VC707 FPGA-Boards [Xil16f] und der daraus resultierenden festen Positionen der I/O-Blöcke sowie des PCIe-Endpoints vorgegeben. Die RC2F-Infrastruktur (siehe nachfolgend Abschnitt 6.1.4.4) befindet sich innerhalb eines statischen Bereiches. Dieser statische Bereich wird entsprechend vertikal über alle sieben Taktregionen an der rechten Seite des FPGAs angeordnet, wie in Abbildung 6.5 dargestellt. Die Frontends werden entsprechend auf der linken Seite des FPGA-Hypervisors platziert. Die unterste Taktregion ist für den Ethernet-Controller und als Zugang zur FPGA-Infrastruktur zur Rekonfiguration der vFPGA-Slots vorgesehen.

Resultierend aus den in Abschnitt 4.4.3.1 erläuterten Abwägungen und den festen Positionen der Komponenten ergeben sich insgesamt sechs vertikal übereinander angeordnete vFPGA-Slots auf dem Virtex-7. Die Bereiche für die vFPGA-Slots sind entsprechend dynamisch partiell rekonfigurierbar. Der Zugang zu den vFPGA-Slots erfolgt aufgrund ihrer Anordnung von deren rechten Seite über die Frontends innerhalb der statischen Region und die Partition Pins als Zugangspunkt über die Partition Pin Regions (PPRs) innerhalb der dynamisch rekonfigurierbaren vFPGA-Slots.

6 Prototypische Implementierung und Ergebnisse

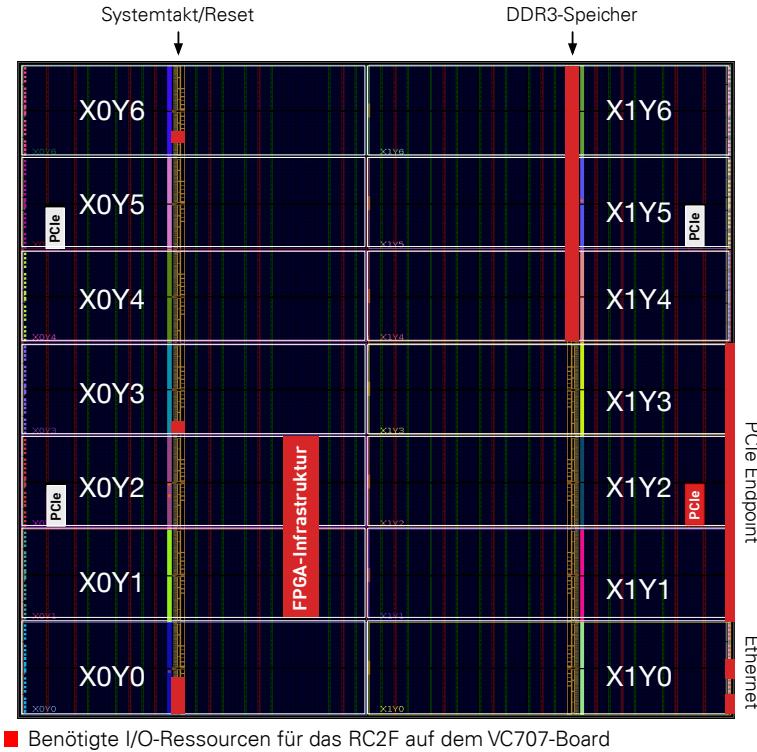


Abbildung 6.4: Übersicht zu den I/O- und Infrastruktur-Komponenten auf dem Virtex-7 mit Kennzeichnung der für das RC2F benötigten Hardware-Ressourcen. Die Positionen sind durch das Layout des FPGA-Boards VC707 [Xil16f] vorgegeben. Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 von Xilinx [Xil16g].

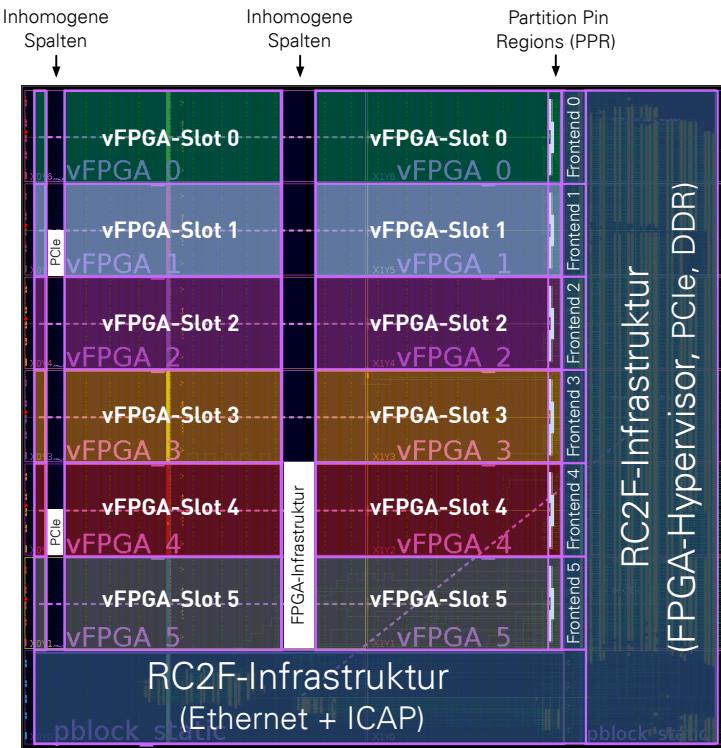


Abbildung 6.5: RC2F-Prototyp mit den unterschiedlichen Bereichen für die statische RC2F-Infrastruktur, den sechs dynamisch rekonfigurierbaren homogenen vFPGA-Slots und den Partition Pin Regions (PPRs) auf dem Virtex-7. Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 von Xilinx [Xil16g].

Des Weiteren kommt den Frontend-Interfaces eine besondere Bedeutung zu, da diese sich innerhalb sämtlicher vFPGA-Slots stets an den gleichen Positionen befinden müssen, um eine Schnittstelle zu bieten, welche die geforderte 1D-Relocation zur Optimierung der Migration bereitstellt. Die Partition Pins (siehe Abschnitt 2.1.4.3) werden dazu direkt vor dem Mapping des statischen vFPGA-Designs mit entsprechendem Offset innerhalb der vFPGA-Slots platziert, wie in Abbildung 6.5 dargestellt.

6.1.4.2 Kosten der Bereitstellung homogener vFPGAs

Um die Migration von vFPGA-Instanzen zwischen unterschiedlichen vFPGA-Slots und unterschiedlichen physischen FPGAs zu ermöglichen, ist es – wie bereits erwähnt – erforderlich, dass sich Register und Speicher innerhalb unterschiedlicher vFPGA-Slots an identischen relativen Positionen befinden. Inhomogene Strukturen, wie die FPGA-Infrastruktur und die PCIe-Endpunkte (siehe Abbildung 6.4), welche Homogenität verhindern, müssen entsprechend innerhalb aller vFPGA-Slots über die kompletten Spalten des FPGAs ausgespart werden. Diese Bereiche, welche die vFPGA-Slots unterbrechen, enthalten dabei keine FPGA-Ressourcen, werden aber für Leitungen innerhalb der vFPGAs genutzt.

Für die Inhomogenität der in Abbildung 6.5 aufgezeigten vFPGA-Slots sind die PCIe-Endpunkte in vFPGA-Slot 1 und vFPGA-Slot 4 sowie die FPGA-Infrastruktur in vFPGA-Slot 5 und vFPGA-Slot 6 verantwortlich. Die Anzahl der auf dem physischen FPGA verfügbaren Hardware-Ressourcen ist daher innerhalb der vFPGA-Slots unterschiedlich, wenn diese über die komplette Breite des FPGAs reichen. Abbildung 6.6 zeigt die maximale Anzahl der Hardware-Ressourcen innerhalb der inhomogenen vFPGA-Slots. Die größten Einbußen entstehen demnach bei vFPGA-Slot 4, welcher entsprechend die Basis für eine Homogenisierung der vFPGA-Slots darstellt.

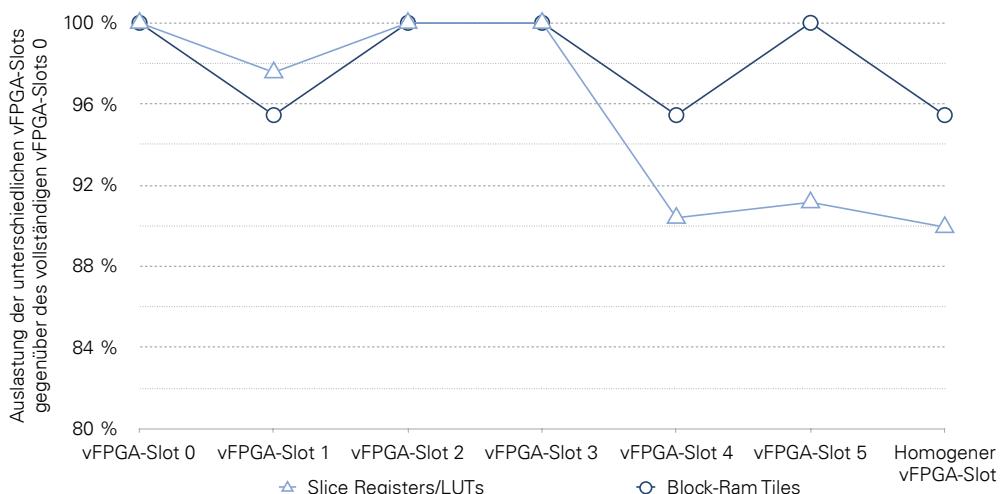


Abbildung 6.6: Einbußen in den Ressourcen der unterschiedlichen vFPGAs, verglichen mit einer vollständigen Region wie vFPGA-Slot 0.

Das Ausgrenzen derjenigen Spalten, welche die inhomogenen Strukturen enthalten, resultiert in den in Abbildung 6.5 gezeigten, in den FPGA-Ressourcen allerdings homogenen vFPGA-Slots. Die entsprechenden FPGA-Ressourcen der homogenen vFPGA-Slots sind ebenso in Abbildung 6.6 dargestellt. Die Einbußen auf Seiten der Slice LUTs und Slice Register liegen bei 10,06 %, die der BRAM Tiles bei 4,55 %. Weitere Ressourcen wie DSPs werden nicht beeinträchtigt.

6.1.4.3 Auslastung der unterschiedlichen Bereiche innerhalb des RC2F-Prototypen

Auf dem physischen FPGA ergeben sich somit für den in Abbildung 6.5 gezeigten RC2F-Prototypen drei unterschiedliche Arten von Bereichen:

- Die dynamisch rekonfigurierbaren Bereiche für die unterschiedlichen vFPGA-Slots,
- Der statische Bereich für die RC2F-Infrastruktur und die Frontend-Interfaces und
- Die aufgrund der Homogenität innerhalb der vFPGA-Slots nicht für die Platzierung von Hardware-Ressourcen nutzbaren Bereiche.

Abbildung 6.7 zeigt den Anteil der FPGA-Ressourcen innerhalb der unterschiedlichen Bereiche im Verhältnis zu den Hardware-Ressourcen des gesamten physikalischen FPGAs. Aufgrund des großen statischen Bereiches stehen effektiv nur zwischen 58,30 % (Slice Registers) und 61,17 % (BRAM Tiles) der Hardware-Ressourcen für vFPGAs bereit. Eine Ausnahme bilden die DSPs, welche sich zu 72,86 % innerhalb der vFPGA-Slots befinden. Anteilig stehen somit insgesamt 62,34 % aller FPGA-Ressourcen des physischen FPGAs in Form von vFPGA-Slots dem Nutzer zur Verfügung. Dabei ist zu berücksichtigen, dass sämtliche Kommunikationsinfrastruktur innerhalb des statischen Bereiches den Nutzern indirekt zur Verfügung steht. Die FPGA-Ressourcen innerhalb des statischen Bereichs für die Infrastruktur umfassen 31,07 % der gesamten FPGA-Ressourcen. Durch die homogenen Bereiche sind nicht nutzbare Regionen entstanden, welche 6,59 % der FPGA-Ressourcen des physischen FPGAs umfassen. Bei größeren FPGAs ist der Anteil der statischen RC2F-Infrastruktur entsprechend kleiner, da die RC2F-Implementierung und die unterschiedlichen Bereiche nicht linear mit der Größe des physischen FPGAs skalieren (siehe nachfolgend Abschnitt 6.4.3).

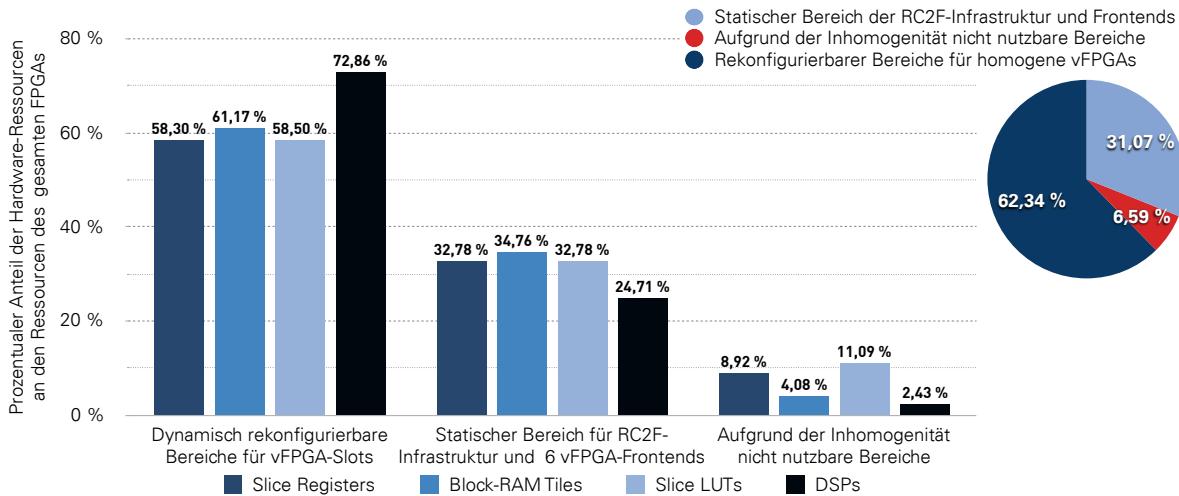


Abbildung 6.7: Anteil der Hardware-Ressourcen in den unterschiedlichen Bereichen auf dem physischen FPGA innerhalb des RC2F-Prototypen an den Hardware-Ressourcen des gesamten physikalischen FPGAs.

6.1.4.4 Komponenten der RC2F-Infrastruktur

Die in Abschnitt 4.4 eingeführte RC2F-Infrastruktur wird exemplarisch mit den Komponenten, wie sie in Abbildung 4.6 aufgezeigt sind, innerhalb des statischen Bereichs realisiert. Für die Infrastruktur sind, wie Abbildung 6.5 zeigt, sowohl die rechte Seite des physischen FPGAs, als auch die untere Taktregion vorgesehen. Die konstanten Komponenten der statischen Infrastruktur innerhalb der RC2F-Infrastruktur sind dabei im Einzelnen:

FPGA-Hypervisor: Kernstück der Implementierung ist der FPGA-Hypervisor, welcher die Frontends zu den vFPGAs, wie in Abbildung 4.9 gezeigt, bereitstellt. Wesentliche Komponenten sind der in Abschnitt 4.4.1.2 erläuterte Konfigurationsspeicher des FPGA-Hypervisors, über den sämtliche Steuerbefehle und Signale an den FPGA übermittelt werden, sowie der ICAP-Controller [Gen15] zur Rekonfiguration der vFPGA-Slots und zum Auslesen einer partiell rekonfigurierbaren vFPGA-Instanz für eine Migration. Die Speicher sind aus Komponenten der Pile of Cores (PoC)-Bibliothek [Leh16; Pre+16] realisiert und wie in Abbildung 4.7 gezeigt aufgebaut. Die interne Taktrate (Systemtakt) des FPGA-Hypervisors und der Geräte-Virtualisierung beträgt 250 MHz. Um den FPGA-Hypervisor sowohl von der internen Logik der vFPGAs als auch von den I/O-Komponenten zu entkoppeln, befinden sich an den Schnittstellen zwischen den Taktdomänen Cross-Clocking-FIFOs. Die weiteren Kommunikationskanäle sind, wie in Abschnitt 4.4.5 erläutert, aufgebaut, wobei die Kanal-Virtualisierung im Prototypen nicht vollständig umgesetzt wurde.

PCIe-Controller: Der PCIe-Controller ist der von Xilinx bereitgestellte Intellectual Property Core (IP-Core) *7 Series FPGAs Integrated Block for PCI Express v3.3* [Xil17b] mit einem aufgesetzten Xillybus-Controller [Xil16], welcher auf dem FPGA sowohl FIFO- und Speicherschnittstellen, als auch einen Treiber innerhalb des Host-Hypervisors bereitstellt (vergleiche Abbildung 4.14).

DDR3-Controller und Speicher-Virtualisierung: Der verwendete DDR3-Controller ist der IP-Core *Xilinx MIG V1.4* [Xil12a], welcher die Schnittstelle zum Backend-Interface, wie in Abschnitt 4.4.7 dargestellt, bereitstellt. Die darauf aufsetzende Speicher-Virtualisierung, die über den Host-Hypervisor verwaltet wird, organisiert die konkrete Übersetzung von den virtuellen auf die physischen Adressen und separiert somit die Nutzerbereiche im Speicher voneinander.

Ethernet-Controller: Der verwendete Ethernet-Controller basiert auf dem IP-Core *LogiCORE IP Tri-Mode Ethernet MAC v5.2* [Xil12b], welcher ein Interface auf der Media-Access-Control (MAC)-Ebene des OSI-Referenzmodells [Zim80] bietet. Darauf aufbauend werden Teile der PoC-Bibliothek genutzt, um die Schnittstellen zu den vFPGAs zu realisieren.

Neben den zuvor erläuterten Komponenten der RC2F-Infrastruktur sind weitere Komponenten erforderlich, deren Hardwareressourcen von der Anzahl der physischen vFPGA-Slots abhängig ist und die sich ebenfalls im statischen Bereich befinden:

Geräte-Virtualisierung: Die Geräte-Virtualisierung, wie sie in Abschnitt 4.4.5 erläutert wurde, stellt die nebenläufigen Kommunikationskanäle für die vFPGAs bereit. Die Realisierung der PCIe-Virtualisierung erfolgt mittels der durch den Xillybus-Controller [Xil16] bereitgestellten Komponenten. Die bereitgestellten FIFOs werden dabei an die vFPGAs durchgereicht und entsprechend entkoppelt (Cross-Clocking), um unterschiedliche Taktdomänen für den FPGA-Hypervisor und das vFPGA-Design zu ermöglichen. Für die Speicher-Virtualisierung wird je eine Seitentabelle für jeden Nutzer benötigt. In der prototypischen Implementierung werden Seitengrößen von 8 MByte genutzt (siehe Abschnitt 4.4.7).

vFPGA-Frontends: Die Frontends werden, wie in Abschnitt 4.4.1.3 skizziert, umgesetzt. Die Konfigurationsspeicher sind entsprechend Abbildung 4.8 aufgebaut und bestehen einerseits aus einem im statischen Bereich des FPGAs angesiedelten Teil und andererseits aus einem Anwenderbereich, welcher frei genutzt werden kann. Neben den Speichern werden die Zustände jedes vFPGAs, wie in Abschnitt 4.4.2 skizziert, verwaltet.

6.1.4.5 Ressourcenverbrauch der RC2F-Infrastruktur

Die benötigten FPGA-Ressourcen des zuvor beschriebenen Prototypen stellen eine entscheidende Kennzahl dar, um die Auslastung des statischen Bereiches sowie die Skalierung des Ressourcenverbrauchs in Abhängigkeit der Anzahl der vFPGAs bewerten zu können. Die FPGA-Ressourcen für einen bestimmten Bereich werden im Folgenden über den 4-dimensionalen Vektor $\vec{\rho}$ aus Gleichung 6.6 beschrieben.

$$\vec{\rho} = \begin{pmatrix} \text{Slice LUTs} \\ \text{Slice Register} \\ \text{BRAM Tiles} \\ \text{DSPs} \end{pmatrix} \quad (6.6)$$

Die FPGA-Ressourcen des statischen Bereiches des RC2F-Prototypen $\vec{\rho}_{RC2F-\text{Statisch}}(N_{vFPGA-Slots})$ bestehend aus den Ressourcen der RC2F-Infrastruktur $\vec{\rho}_{RC2F-\text{Infrastruktur}}$ und den Ressourcen für eine bestimmten Anzahl vFPGA-Slots $\vec{\rho}_{vFPGA-Slots}(N_{vFPGA-Slots})$ werden über Gleichung 6.7 berechnet.

$$\vec{\rho}_{RC2F-\text{Statisch}}(N_{vFPGA-Slots}) = \vec{\rho}_{RC2F-\text{Infrastruktur}} + \vec{\rho}_{vFPGA-Slots}(N_{vFPGA-Slots}) \quad (6.7)$$

Die FPGA-Ressourcen für die eigentliche Infrastruktur $\vec{\rho}_{RC2F-\text{Infrastruktur}}$ innerhalb des statischen Bereiches wird mittels über Gleichung 6.8 berechnet. Die FPGA-Ressourcen der einzelnen Komponenten sind dabei die des FPGA-Hypervisor $\vec{\rho}_{FPGA-\text{Hypervisor}}$, des DDR $\vec{\rho}_{DDR3}$, des Ethernet $\vec{\rho}_{Ethernet}$ sowie des PCIe-Controllers $\vec{\rho}_{PCIe}$.

$$\vec{\rho}_{RC2F-\text{Infrastruktur}} = \vec{\rho}_{FPGA-\text{Hypervisor}} + \vec{\rho}_{PCIe} + \vec{\rho}_{DDR3} + \vec{\rho}_{Ethernet} \quad (6.8)$$

Für den Virtex-7 ergeben sich die in Tabelle 6.1 aufgezeigten FPGA-Ressourcen für die statische RC2F-Infrastruktur. Sämtliche FPGA-Ressourcen wurden durch die Synthese- und Entwicklungsumgebung Vivado 2016.4 von Xilinx [Xil16g] ermittelt. Dabei werden zunächst sechs vFPGAs auf dem FPGA realisiert. Neben der Anzahl der FPGA-Ressourcen zeigen Tabelle 6.1 und Abbildung 6.8 den prozentualen Anteil der jeweiligen FPGA-Ressourcen an der statischen Region des RC2F-Prototypen.

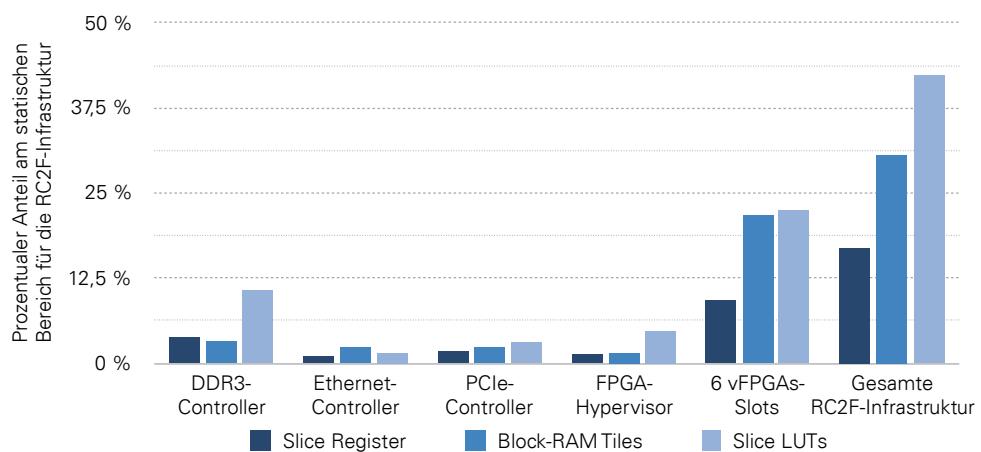


Abbildung 6.8: Prozentualer Bedarf an FPGA-Ressourcen innerhalb der statischen RC2F-Infrastruktur mit den einzelnen Komponenten des RC2F-Prototypen auf dem Virtex-7 mit sechs vFPGAs.

Tabelle 6.1 zeigt weiterhin, dass die Slice LUTs hauptsächlich innerhalb des DDR-Controllers benötigt werden, die BRAMs hingegen innerhalb der Frontend-Interfaces. Dieses Verhalten kann mit den zu-

sätzlichen Kommunikationskanälen und deren massivem Pipelining erklärt werden. Die ebenso innerhalb der RC2F-Infrastruktur enthaltenen Frontends und die zusätzliche Geräte-Virtualisierung für die sechs vFPGA-Slots benötigt den größten Teil der FPGA-Ressourcen aus dem Grunde, dass die entsprechenden Komponenten, wie beispielsweise die Kommunikationskanäle und Seitentabellen, für jeden der vFPGA-Slots vorhanden sind. Der in Abschnitt 6.1.4.1 für die RC2F-Infrastruktur und die sechs vFPGA-Slots festgelegte statische Bereich ist nach Tabelle 6.1 mit maximal 42,30 % (Slice LUTs) ausgelastet.

Tabelle 6.1: Benötigte FPGA-Ressourcen der statischen RC2F-Infrastruktur und deren Einzelkomponenten auf dem Virtex-7 XC7VX485T für sechs vFPGA-Slots. Erzeugt/Gemessen mit Vivado 2016.4 [Xil16g].

RC2F-Komponenten	Slice LUTs		Slice Register		BRAM Tiles	
	Anzahl	Statischer Bereich ^a (%)	Anzahl	Statischer Bereich ^a (%)	Anzahl	Statischer Bereich ^a (%)
RC2F-Infrastruktur						
FPGA-Hypervisor	4.417	4,66	2.458	1,30	5	1,47
PCIe-Controller	2.864	3,02	3.475	1,83	8	2,17
DDR3-Controller	10.156	10,71	7.002	3,69	12	3,25
Ethernet-Controller	1.448	1,53	1.723	0,91	8	2,17
Summe – $\vec{p}_{RC2F-Infrastruktur}$	18.885	19,92	14.658	7,73	33	9,06
6 vFPGA-Slots						
Geräte-Virtualisierung	16.235	17,12	9.283	4,89	20	5,30
vFPGA-Frontends	4.995	5,27	9.385	4,95	60	16,26
Summe – $\vec{p}_{vFPGA-Slots(6)}$	21.230	22,39	18.668	9,84	80	21,56
Summe – $\vec{p}_{RC2F-Statisch(6)}$	40.115	42,30	33.326	17,57	113	30,62

^a FPGA-Ressourcen des statischen Bereichs auf dem Virtex-7 XC7VX485T (Slice LUTs: 94.824, Slice Register: 189.648, BRAM Tiles: 369).

Tabelle 6.2: FPGA-Ressourcen für die RC2f-Infrastruktur mit entsprechenden Frontend-Interfaces in Abhängigkeit der Anzahl der vFPGA-Slots auf dem Virtex-7 XC7VX485T. Erzeugt/Gemessen mit Vivado 2016.4 [Xil16g].

vFPGA-Slots	Slice LUTs		Slice Register		BRAM Tiles	
	$n_{vFPGA-Slots}$	Anzahl	Statischer Bereich ^a (%)	Anzahl	Statischer Bereich ^a (%)	Anzahl
0	18.885	19,92	14.658	7,73	33	9,06
1	22.249	23,46	17.519	9,27	47	12,65
2	25.194	26,57	20.582	10,81	60	16,25
3	28.748	30,32	23.528	12,35	73	19,84
4	31.870	33,61	26.355	13,90	86	23,44
5	36.776	38,78	29.879	15,44	100	27,03
6	40.115	42,30	33.326	17,57	113	30,62
7 ^b	43.648	46,03	34.683	18,53	126	34,22
8 ^b	47.286	49,87	38.811	20,06	140	37,81

^a FPGA-Ressourcen des statischen Bereichs auf dem Virtex-7 XC7VX485T (Slice LUTs: 94.824, Slice Register: 189.648, BRAM Tiles: 369).

^b Die FPGA-Ressourcen für 7 und 8 vFPGA-Slots sind ohne eine partielle Region auf dem FPGA ermittelt.

Um eine Aussage über das Wachstum der FPGA-Ressourcen in Abhängigkeit von der Anzahl der FPGA-Slots treffen zu können, zeigt Tabelle 6.2 die erforderlichen FPGA-Ressourcen für eine unterschiedliche Anzahl von vFPGA-Slots. Hierbei sind bis zu acht vFPGA-Slots auf dem FPGA realisiert, wobei für mehr

6 Prototypische Implementierung und Ergebnisse

Tabelle 6.3: Approximation der Messwerte aus Tabelle 6.2 durch lineare Regression mit den Konstanten K_1 und K_2 für Gleichung 6.9, sowie dem Bestimmtheitsmaß R^2 .

Regression	Anzahl Slice LUTs	Anzahl Slice Register	Anzahl BRAM Tiles
Konstanter Anteil K_1	14.780	11.589	20,17
Linearer Faktor K_2	3594,50	14.780	13,26
Bestimmtheitsmaß R^2	0,9978	0,9971	1,00

als sechs vFPGA-Slots keine physischen Bereiche zugewiesen sind. Mit den Messwerten aus Tabelle 6.2 können durch lineare Regression die FPGA-Ressourcen für den vollständigen statischen Bereich mit RC2F-Infrastruktur und vFPGA-Slots $\vec{\rho}_{RC2F-\text{Statisch}}(N_{vFPGA-\text{Slots}})$ nach Gleichung 6.9 approximiert werden. Abbildung 6.9 zeigt die Messreihen für die unterschiedlichen FPGA-Ressourcen, sowie die lineare Approximation, bei welcher der linearer Zusammenhang zu erkennen ist.

$$\vec{\rho}_{RC2F-\text{Statisch}}(N_{vFPGA-\text{Slots}}) = \vec{\rho}_{K_1} + (N_{vFPGA-\text{Slots}} + 1) \cdot \vec{\rho}_{K_2} \quad (6.9)$$

Die Konstanten $\vec{\rho}_{K_1}$ und $\vec{\rho}_{K_2}$ der Regression sind entsprechend für die unterschiedlichen FPGA-Ressourcen in Tabelle 6.3 aufgezeigt. Diese Konstanten wurden mit Hilfe der Methode der kleinsten Quadrate [FKL07] ermittelt und sind für die unterschiedlichen FPGA-Ressourcen in Tabelle 6.2 aufgelistet. Die Tabelle enthält ebenso das Bestimmtheitsmaß R^2 , um den linearen Zusammenhang nachzuweisen [CW97]. Da das Bestimmtheitsmaß die FPGA-Ressourcen in allen Fällen über 0,99 liegt ist der lineare Zusammenhang somit nachgewiesen.

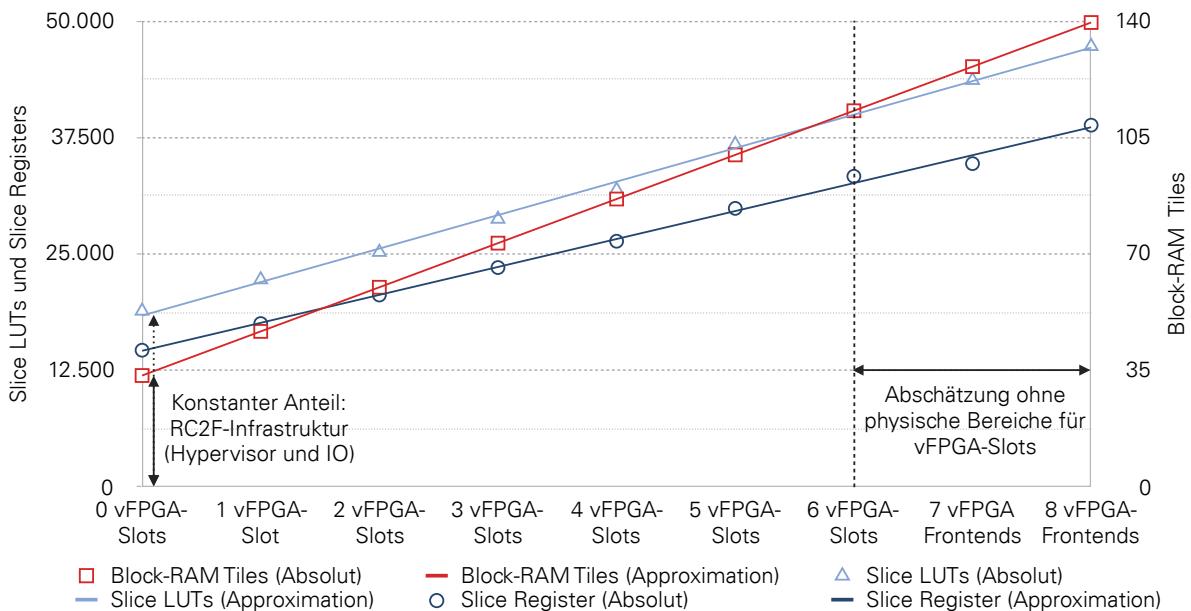


Abbildung 6.9: Bedarf an FPGA-Ressourcen der RC2F-Infrastruktur in Abhängigkeit von der Anzahl der vFPGA-Slots nach Gleichung 6.9 mit den Konstanten aus Tabelle 6.3 sowie den Messpunkten aus Abbildung 6.9.

6.1.4.6 Aggregierte Gruppen von vFPGAs

Neben den einzelnen vFPGA-Slots, wie sie in Abbildung 6.5 gezeigt wurden, sind des Weiteren, wie in Abschnitt 4.3.1 erläutert, unterschiedliche große vFPGAs durch Zusammenschluss der einzelnen vFPGA-Slots möglich (vergleiche Abbildung 4.3). Ein vFPGA, welcher über einen vFPGA-Slot reicht wird im Folgenden als *Single*, zusammengeschlossene homogene vFPGAs werden entsprechend der Anzahl an aggregierten vFPGA-Slots als *Double*-, *Triple*-, *Quad*-, *Quint*- und *Hexa*-vFPGA bezeichnet. Da der Hexa-vFPGA, welcher sich entsprechend über alle sechs vFPGA-Slots beziehungsweise den kompletten dem Nutzer zur Verfügung stehenden Bereich auf dem Virtex-7 erstreckt, nur an genau einer Stelle platziert werden kann, kann auf die untereinander homogenen Bereiche verzichtet werden, da dieser vFPGA nur genau eine Position auf dem physischen FPGA einnehmen kann.

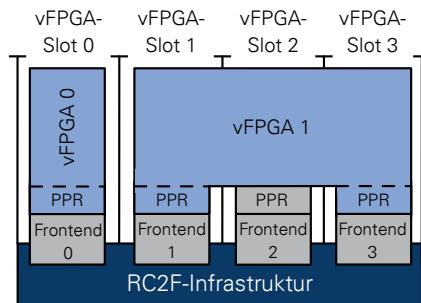


Abbildung 6.10: Beispiel eines aggregierten vFPGAs über drei vFPGA-Slots und zwei Frontend-Interfaces mit den erforderlichen Partition Pin Regions (PPRs).

Da die einzelnen homogenen vFPGAs-Slots zu größeren Bereichen aggregiert werden, sind die sich ergebenden FPGA-Ressourcen für die unterschiedlichen vFPGAs immer ein ganzzahliges Vielfaches eines einzelnen vFPGA-Slots (Single), wie in Tabelle 6.4 dargestellt. Um jedem aggregierten vFPGA des Weiteren mindestens ein, und maximal so viele Frontend-Interfaces bereitzustellen zu können, wie vFPGA-Slots genutzt werden, ist für die Berechnung der konkreten FPGA-Ressourcen die Berücksichtigung der Partition Pin Regions (PPRs) erforderlich. Für die innerhalb der RCFG-Datei angeforderten Frontend-Interfaces wird entsprechend die Konfiguration der zusätzlichen PPR zugelassen, wie in Abbildung 6.10 gezeigt. Der Bereich eines nicht genutztes Frontends steht dem vFPGA-Design entsprechend nicht zur Verfügung.

Tabelle 6.4: FPGA-Ressourcen der aggregierten vFPGA-Slots, welche durch den RC2F-Prototypen auf einem Virtex-7 XC7VX485T dem Nutzer zur Verfügung gestellt werden. Die vFPGAs sind in ihrer Größe immer ein ganzzahliges Vielfaches eines einfachen (Single) vFPGAs, welcher einem einzelnen homogenen vFPGA-Slot entspricht.

FPGA-Ressourcen	PPR	Größe des vFPGAs (auf Basis homogener vFPGA-Slots)						
		Single	Double	Triple	Quad	Quint	Hexa-homogen ^a	Hexa ^b
Slice LUTs	1200	28.400	56.800	85.200	113.600	142.000	170.400	188.400
Slice Registers	2400	59.000	118.000	177.000	236.000	295.000	354.000	376.800
BRAM Tile	0	105	210	315	420	525	600	630
DSPs	20	340	680	1.020	1.360	1.700	1.940	2.040

^a Zusammenfassung der homogenen vFPGA-Slots.

^b Größtmöglicher Bereich ohne Einhaltung der Homogenität in den vFPGA-Slots, da Verschiebung nicht möglich ist.

6 Prototypische Implementierung und Ergebnisse

Durch die Angabe der Anzahl der aggregierten vFPGA-Slots $N_{vFPGA-Slots}$ und der Frontends $N_{Frontends}$ können die FPGA-Ressourcen $\vec{\rho}_{vFPGA}(N_{vFPGA-Slots}, N_{Frontends})$ mittels Gleichung 6.10 berechnet werden. Dabei steht $\vec{\rho}_{FPGA-Slot}$ für die FPGA-Ressourcen eines einzelnen vFPGAs (Single) und $\vec{\rho}_{ppr}$ für die Ressourcen der Partition Pin Region.

$$\vec{\rho}_{vFPGA}(N_{vFPGA-Slots}, N_{Frontends}) = N_{vFPGA-Slots} \cdot \vec{\rho}_{FPGA-Slot} + N_{Frontends} \cdot \vec{\rho}_{ppr} \quad (6.10)$$

Die von den aggregierten vFPGAs ungenutzten Fontend-Schnittstellen in Form der Partition Pins werden durch einen speziellen partiellen Bitstream versiegelt, welcher einen einfachen Endpunkt bereitstellt, um definierte Signalzustände zu garantieren. Der vFPGA-Konfigurationsspeicher enthält entsprechende Basisinformationen, um sowohl Host- als auch FPGA-Hypervisor den entsprechenden Zustand des vFPGA-Frontends mitzuteilen.

6.1.4.7 Entwurfsablauf für homogene vFPGAs zur Kontextmigration

Der in Abschnitt 5.1.5 skizzierte Entwurfsablauf für die Nutzung von homogenen und insbesondere migrierbaren vFPGAs ist prototypisch umgesetzt beziehungsweise erweitert. Als Basis dient dafür die Entwicklungsumgebung Xilinx Vivado 2016.4 [Xil16g]. Das Ziel des modifizierten Entwurfsablaufs besteht darin, für ein gegebenes vFPGA-Design die vFPGA-Images (Bitstreams) für sämtliche (aggregierten) vFPGA-Slots zu erzeugen.

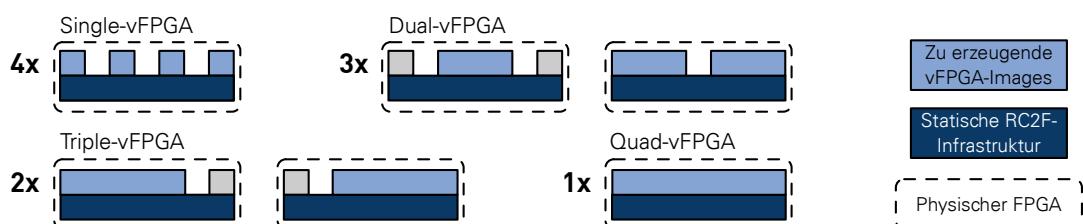


Abbildung 6.11: Erforderliche vFPGA-Images für unterschiedliche Größen von (aggregierten) vFPGAs. Ein Dual-vFPGA kann sich beispielsweise innerhalb von drei unterschiedlichen aggregierten vFPGA-Slots auf dem physischen FPGA befinden.

Abbildung 6.11 zeigt exemplarisch die unterschiedlichen zu erzeugenden vFPGA-Images für alle möglichen (aggregierten) vFPGA-Slots auf dem physischen FPGA anhand eines Beispiels mit vier vFPGA-Slots. Dabei müssen sich für eine einfache Migration sämtliche Slice Registers/LUTs, BRAM Tiles und DSPs an identischen relativen Positionen innerhalb der vFPGAs befinden. Als Basis dienen dabei die in Abbildung 6.5 aufgezeigten homogenen Regionen. Abbildung 6.12 gibt einen Überblick über den modifizierten Entwurfsablauf zur Erzeugung sowohl von statischer FPGA-Infrastruktur als auch von partiellen vFPGA-Images für die vFPGA-Slots. Der Entwurfsablauf beginnt zunächst mit der einmaligen Erzeugung der statischen RC2F-Infrastruktur vom Anbieter der Cloud:

Schritt 0 – Erzeugen der statischen RC2F-Infrastruktur: Zunächst wird mit dem vom FPGA-Hersteller vorgesehenen Entwurfsablauf, unter Berücksichtigung der partiellen dynamischen Rekonfiguration [Xil17g], die statische RC2F-Infrastruktur, wie sie in Abschnitt 6.1.4.4 aufgezeigt ist, generiert. Dabei werden die Partition Pins über ein Skript in jedem vFPGA-Slots an der gleichen relativen Positionen innerhalb der Partition Pin Region platziert (siehe Abbildung 6.5), um entsprechend das Interface zu den vFPGA-Slots bereitzustellen. Platzierung und Routing können erfolgen, und das fertigge-

stellte Hardwaredesign wird als Checkpoint gesichert und für die weiteren Schritte bereitgestellt. Anschließend wird der Bitstream für die RC2F-Infrastruktur im statischen Bereich erzeugt.

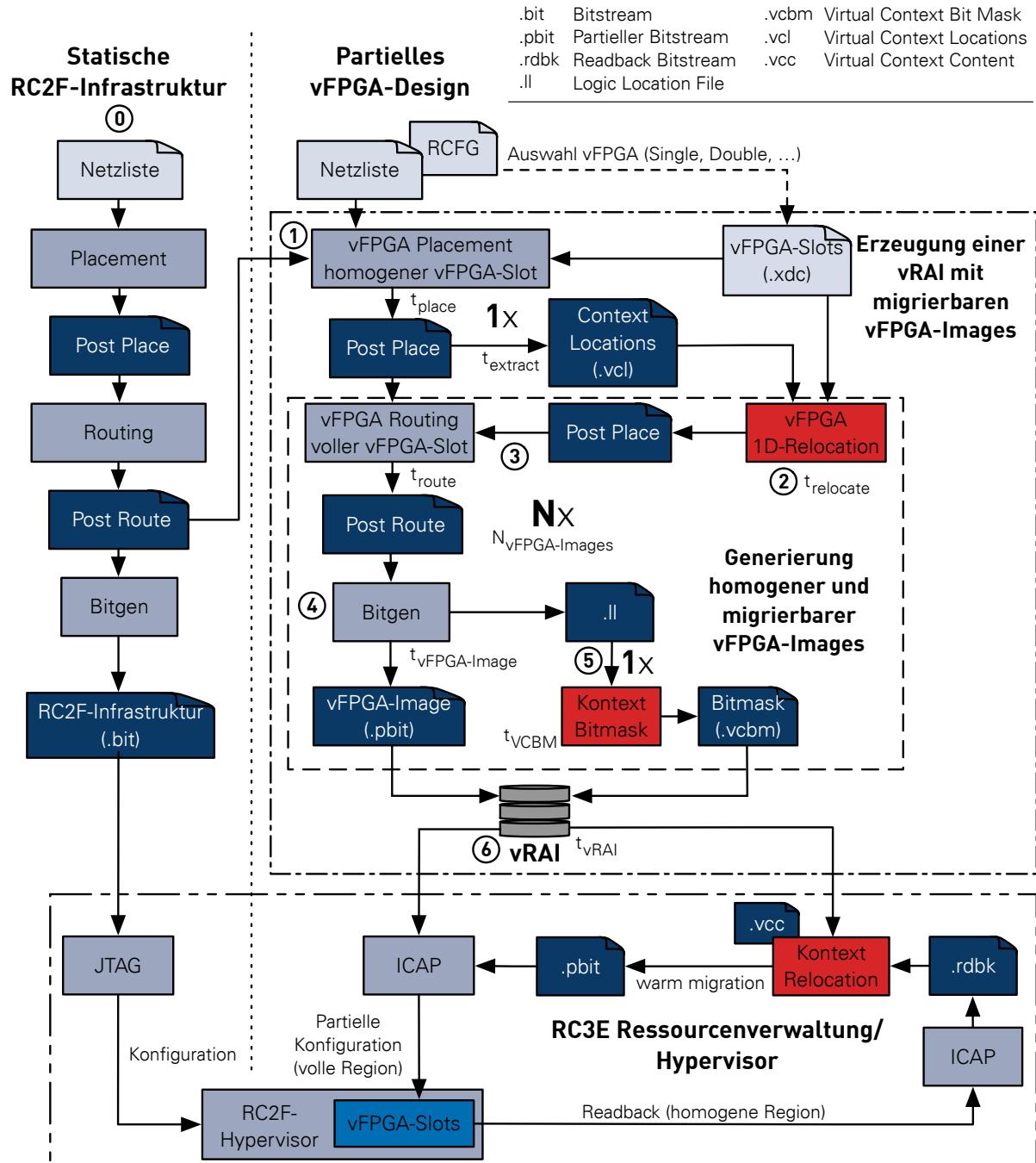


Abbildung 6.12: Modifizierter Entwurfsablauf (auf Basis von Vivado 2016.4 [Xil16g]) zur Erzeugung sowohl von statischer FPGA-Infrastruktur als auch von homogenen vFPGA-Images, welche eine Migration der vFPGA-Instanzen auf Basis homogener vFPGA-Slots ermöglichen.

In den nächsten Schritten werden die vFPGA-Images für ein konkretes vFPGA-Design eines Nutzers erzeugt. Für die Ausnahme, dass statische Leitungen der RC2F-Infrastruktur innerhalb der vFPGA-Slots enthalten sind, wie es im RC2F-Prototypen der Fall ist, ist der angepasste Entwurfablauf zur Erzeugung der unterschiedlichen vFPGA-Images entsprechend in sechs Hauptschritte unterteilt:

Schritt 1 – vFPGA-Designs in den ersten (aggregierten) vFPGA-Slot(s) platzieren: Auf Basis der

Netzliste des vFPGA-Designs wird in einem ersten Schritt ein vFPGA passender Größe ausgewählt. Die Netzliste wird daraufhin in einen homogenen vFPGA-Slot geeigneter Größe (Single, Double etc.) platziert, wobei die inhomogenen Spalten nicht zur Verfügung stehen. Die Partition Pin Regions (PPRs) als Schnittstellen zu den Frontends innerhalb der statischen RC2F-Infrastruktur, welche die Partition Pins (PPs) enthält, werden entsprechend der Beschreibung des vFPGAs aus der RCFG-Datei (siehe Abschnitt 5.1.4) hinzugefügt.

Nachdem das erste vFPGA-Designs innerhalb (aggregierter) vFPGA-Slots platziert ist, erfolgt die Übertragung der platzierten FPGA-Ressourcen auf die anderen möglichen (aggregierten) vFPGA-Slots, um ein vFPGA-Image innerhalb beliebiger (aggregierter) vFPGA-Slots platzieren zu können. Die Anzahl der notwendigen $N_{vFPGA-Images}$ wird nach Gleichung 6.11 berechnet (siehe auch Abbildung 6.11). Die Anzahl ist dabei ebenso abhängig von der Anzahl der auf dem physischen FPGA vorhandenen vFPGA-Slots $N_{vFPGA-Slots}$ und von der Anzahl der aggregierten vFPGA-Slots N_{Agg} über welche sich der vFPGA erstreckt.

$$N_{vFPGA-Images} = N_{vFPGA-Slots} - N_{Agg} + 1 \quad (6.11)$$

Die folgenden Schritte innerhalb des Entwurfsablaufes müssen entsprechend $N_{vFPGA-Images}$ Male durchgeführt werden und berücksichtigen die statischen Leitungen innerhalb der vFPGA-Slots des RC2F-Prototypen. Sind die vFPGA-Slots vollkommen identisch und frei von statischen Leitungen und inhomogenen Komponenten, können die vFPGA-Images mit sämtlichen platzierten Komponenten und Leitungen verschoben werden, indem lediglich die FAR-Adressen innerhalb des Bitstreams modifiziert werden.

Schritt 2 – Übertragen des platzierten vFPGA-Designs auf weitere vFPGA-Slots (1D-Relocation):

Die Register-, BRAM- und DSP-Positionen auf dem physischen FPGA, welche durch Location Constraintss (LOCs) und Basic Element Locations (BELs) bestehen, werden aus dem platzierten vFPGA-Design ausgelesen und in einer separaten Virtual Context Locations (VCL) Datei gespeichert. Anschließend wird die Netzliste der weiteren (aggregierten) vFPGA-Slots geladen und die Komponenten entsprechend der zuvor erzeugten VCL Datei mit den Offsets der vFPGA-Regionen innerhalb des physischen FPGAs platziert.

Schritt 3 – Erweitern der homogenen vFPGA-Slots auf die inhomogenen vFPGA-Slots: Nach dem Platzieren der Komponenten wird der homogene vFPGA-Slot über die gesamte Breite des inhomogenen vFPGA-Slots ausgedehnt (Hinzunahme sämtlicher inhomogener Spalten). Nach der Platzierung der Komponenten folgt das Routing der vFPGA-Designs. Durch das vollständige Routing werden auch statische Leitungen innerhalb einzelner vFPGAs berücksichtigt.

Schritt 4 – Erzeugung von vFPGA-Images: Für die vollständig platzierten und verdrahteten vFPGA-Designs wird im nächsten Schritt des Entwurfsablaufes das vFPGA-Image in Form des partiellen Bitstreams (.pbit) erzeugt. Aus dem Bitstream werden für die Verifikation der physischen Bereiche der vFPGA-Instanz innerhalb des FPGAs sämtliche Informationen, wie die Frame Adress Register (FAR) (siehe Abschnitt 2.1.4.3), extrahiert und als Header an das vFPGA-Image angehängt. Der Header wird vor der eigentlichen Konfiguration der vFPGA-Slots innerhalb des FPGA-Hypervisors analysiert und an den ICAP zur Konfiguration weitergereicht, wenn die allokierten vFPGA-Slots mit denen des vFPGA-Images übereinstimmen.

Schritt 5 – Erzeugung der Bit-Maske für die Kontextmigration: Für die Kontextmigration ist die zusätzliche VCBM erforderlich, um die relevanten Daten aus der vFPGA-Instanz extrahieren bezie-

hungswweise in einem vFPGA-Image wiederherstellen zu können. Die VCBM wird innerhalb des Entwurfsablaufes mit Hilfe der nach der Generierung der vFPGA-Images bereitgestellten Metadaten, wie der Logic Location (LL)-Datei, erzeugt. Der über die VCBM extrahierte Kontext wird in einer separaten Virtual Context Content (VCC)-Datei zwischengespeichert. Mit Hilfe der VCC werden die Inhalte sämtlicher Slice Registers/LUTs und BRAM Tiles in anderen vFPGA-Images für das äquivalente vFPGA-Design mit Hilfe der dazugehörigen VCBM wiederhergestellt.

Der CRC des Bitstreams wird durch die Wiederherstellung des Kontextes verändert. Daher müssen für die Rekonfiguration der vFPGA-Slots entsprechende Befehle in den Bitstream ergänzt werden, um den CRC innerhalb des FPGAs zurückzusetzen. Das Laden des neuen Kontextes erfolgt über die Initialisierungs-Register (INIT) der Slice Register/LUTs, beziehungsweise direkt in die BRAM Tiles und wird über das *Global Set/Reset-Signal* für die entsprechenden zu rekonfigurierenden Spalten in die Komponenten auf dem physischen FPGA übernommen (siehe auch [KGS16; Xil16b]).

Schritt 6 – Erzeugung der vRAI für das initiale vFPGA-Design: In einem letzten Schritt werden sämtliche vFPGA-Images mit ihren dazugehörigen VCBMs zur Kontextmigration und der Beschreibung des vFPGAs in Form der RCFG zu einem vRAI-Paket für den Einsatz innerhalb des Modells BAaaS zusammengefasst und signiert (siehe Abschnitt 5.1.6).

Die Laufzeit, welche zur Generierung einer vRAI t_{vRAI} benötigt wird, kann nach Gleichung 6.12 berechnet werden. Die einzelnen Laufzeiten der zuvor erläuterten Schritte innerhalb des modifizierten Entwurfsablaufes (siehe auch Abbildung 6.12) sind dabei die Laufzeiten zum Platzieren des vFPGA-Designs t_{place} und das Extrahieren der Positionen sämtlicher Slice Registers/LUTs und BRAM Tiles $t_{extract}$, welche einmalig erforderlich sind.

$$t_{vRAI} = t_{place} + t_{extract} + N_{vFPGA-Images} \cdot (t_{relocate} + t_{route} + t_{vFPGA-Image} + t_{VCBM}) \quad (6.12)$$

Das Übertragen der Positionen auf die Anderen (aggregierten) vFPGA-Slots $t_{relocate}$, sowie das erneute Routing t_{route} , die Generierung des vFPGA-Images (bitstream) $t_{vFPGA-Image}$ und das Generieren der VCBM t_{VCBM} erfolgen für jede mögliche Position der (aggregierten) vFPGA-Slots $N_{vFPGA-Slots}$ (siehe Gleichung 6.11). Die Anzahl der Durchläufe $N_{vFPGA-Slots}$ ist dabei maßgeblich für die Laufzeit des Entwurfsprozesses zur Erzeugung der vRAIs verantwortlich.

6.1.4.8 Speicherbedarf der vRAI-Pakete

Die zuvor erzeugten vFPGA-Images werden, wie in Abschnitt 6.1.4.7 und Abschnitt 5.1.6 erläutert, in einem vRAI-Paket gebündelt. Bei der Ausführung einer vRAI werden, nach Allokation eines vFPGAs vom Host-Hypervisor die entsprechenden vFPGA-Images für die zugewiesenen vFPGA-Slots ausgewählt und der vFPGA wird konfiguriert. Neben den vFPGA-Images enthält das vRAI-Paket ebenfalls die VCBMs zur Kontextmigration und der Beschreibung der zu allokierten vFPGA-Instanz in Form der RCFG (siehe Abschnitt 5.1.4). Die Anzahl der erforderlichen Bitstreams in Abhängigkeit der aggregierten vFPGA-Regionen und die entsprechenden Dateigrößen der vRAIs sind in Tabelle 6.5 aufgezeigt.

Zusätzlich zeigt Abbildung 6.13 die Dateigrößen der einzelnen (aggregierten) vFPGA-Images sowie die notwendige Anzahl der erforderlichen vFPGA-Images, um sämtliche Positionen der vFPGA-Slots innerhalb des RC2F-Prototypen abzudecken. Die Dateigröße der vFPGA-Images ist unabhängig von dem grundlegenden vFPGA-Design, da jeder Bitstream die gesamten vFPGA-Slots konfiguriert und keine

6 Prototypische Implementierung und Ergebnisse

Kompression eingesetzt wird. Sind die Anzahl der innerhalb des RC2F-Prototypen verfügbaren vFPGA-Slots $N_{vFPGA-Slots}$ sowie der Speicherbedarf eines vFPGA-Images $S_{vFPGA-Image}$ bekannt, lässt sich die Größe einer vRAI S_{vRAI} mit Hilfe von Gleichung 6.11 und Gleichung 6.13 berechnen.

$$S_{vRAI} = 2 \cdot N_{vFPGA-Images} \cdot S_{vFPGA-Image} \quad (6.13)$$

Tabelle 6.5: Größe der vFPGA-Images und resultierenden vRAIs in MByte innerhalb des RC2F-Prototypen.

	Größe des vFPGAs (auf Basis homogener vFPGA-Slots)					
	Single	Double	Triple	Quad	Quint	Hexa
Bitstream (MByte)	4,8	9,0	13,0	17,3	21,3	20,3
Mögliche Positionen	6	5	4	3	2	1
vRAI (MByte)	57,6	90	104	103,8	85,2	40,6

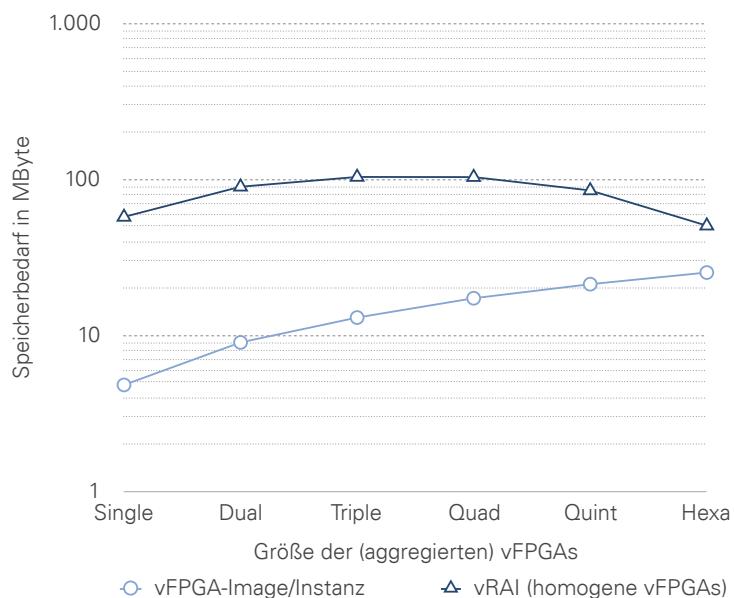


Abbildung 6.13: Größe der vRAI-Pakete für den RC2F-Prototypen (Virtex-7) in Abhängigkeit der unterschiedlichen Größe des zugrundeliegenden vFPGA-Designs.

6.2 Demonstratoren und ausgewählte Szenarien

Um den flexiblen Einsatz des in Abschnitt 6.1.4 entwickelten RC2F-Prototypen für die FPGA-Virtualisierung zu demonstrieren, wird eine Evaluation mit beispielhaften Anwendungen in Form von Demonstratoren genutzt, welche zunächst in Abschnitt 6.2.1 vorgestellt werden. Darauf aufbauend wird in Abschnitt 6.2.2 ein Szenario aufgezeigt, in welchem die vFPGA-Images/-Instanzen dynamisch innerhalb des RC2F-Prototypen migriert werden. Abschnitt 6.2.3 zeigt in einem nächsten Schritt die Szenarien, anhand derer das Verhalten des RC3E-Simulators evaluiert wird.

6.2.1 Demonstratoren zur Evaluation der FPGA-Virtualisierung RC2F

Um den RC2F-Prototypen zu evaluieren, werden exemplarisch vier Demonstratoren, welche sich in Komplexität und Kommunikationsmuster unterscheiden, auf Basis der in Abschnitt 6.1.4.6 entwickelten vFPGAs realisiert. Die Demonstratoren werden im Folgenden kurz erläutert, um darauf aufbauend die Flexibilität der entwickelten Virtualisierung zu demonstrieren. Die Demonstratoren unterscheiden sich im Aufbau der virtuellen Umgebung, in welche sie eingebettet werden (siehe auch Abschnitt 5.3), und in ihrer Bereitstellung (HDL, HLS, IP-Core) voneinander. Alle Demonstratoren nutzen die RC2F-API (siehe Abschnitt 5.1.3 und Abschnitt C.1) und liegen in Form einer vRAI über den zuvor in Abschnitt 6.1.4.7 aufgezeigten modifizierten Entwurfsablauf vor.

I. Matrix-vFPGA (HLS): Als Anwendung zur Evaluation der Integration eines durch Vivado HLS 2016.4 [Xil17f] erzeugten Rechenkernes innerhalb eines vFPGAs unter Nutzung der RC2F-API zur Initialisierung und Konfiguration dient eine Matrixmultiplikation. Dabei wird das FIFO-Interface als Basis für das Streaming unterschiedlicher Matrizen und der vFPGA-Konfigurationsspeicher als Zustandsmaschine eingesetzt. Eine Matrix wird komplett innerhalb des vFPGAs gehalten, die andere wird zeilenweise eingelesen (siehe Abbildung 6.14).

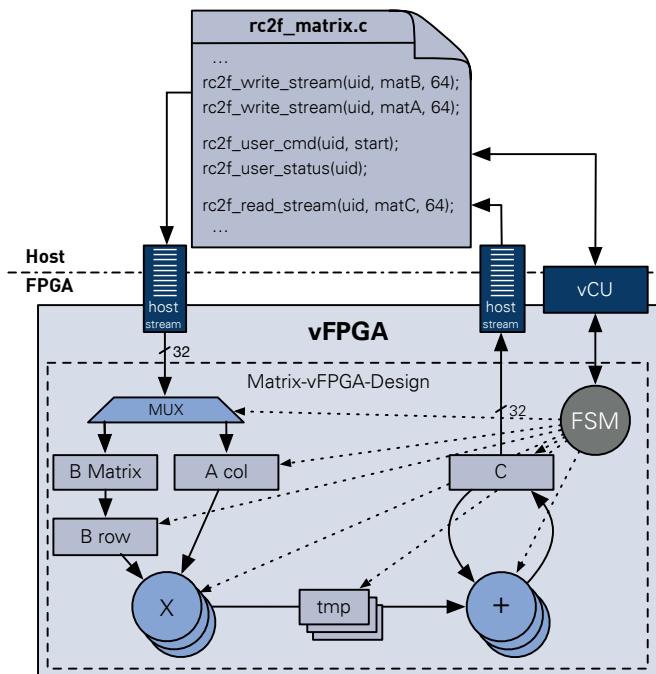


Abbildung 6.14: Aufbau der Matrixmultiplikation auf Basis von vFPGAs und RC2F-API.

Die Anwendung profitiert daher von einer Virtualisierung und Anpassung der FPGA-Ressourcen eines vFPGAs an das erzeugte vFPGA-Design [KS15b]. Eine Matrix der Größe 32×32 benötigt beispielsweise drei aggregierte vFPGA-Slots (Triple-vFPGA). Die erforderlichen FPGA-Ressourcen für sowohl eine 16×16 als auch eine 32×32 Matrix, und deren notwendige vFPGA-Slots sind Tabelle 6.6 zu entnehmen. Der erreichte Speedup ist verglichen mit einem Prozessor⁵ mit 4 Kernen aufgrund der Floating Point Rechenleistung des FPGAs entsprechend gering. Der *Matrix-vFPGA* ist lediglich eine Machbarkeitsstudie und kann zur Entlastung des Host-System eingesetzt werden, nicht aber zur konkreten Beschleunigung einer Anwendung.

⁵Als Referenzsystem für die Softwareimplementierungen dient Rechenknoten0 des in Abschnitt 6.1.1 vorgestellte Cloud-Prototyps mit einem Intel Core i7-2600K mit 4 Rechenkernen.

II. BSMC-vFPGA (HDL): Eine typische rechenintensive Anwendung für eine Cloud, welche ideal auf rekonfigurierbare Hardware ausgelagert werden kann, ist eine Monte-Carlo-Simulation des Black-Scholes-Modells (BSMC) zur Bewertung von Finanzoptionen [Hig04]. Insbesondere die Erzeugung der Vielzahl der hierfür benötigten Zufallszahlen ist auf rekonfigurierbarer Hardware effizient möglich [SPV06]. Des Weiteren können sämtliche Berechnungen mit mehreren Millionen Iterationen auf dem FPGA durchgeführt werden, nachdem eine Initialisierung mit wenigen Parametern erfolgte. Die zu übertragende Datenmenge zum und vom FPGA umfasst lediglich wenige kByte. Der Demonstrator *BSMC-vFPGA* ist in HDL entwickelt und in einen vFPGA mit der entsprechenden RC2F-API eingebettet [Rus15].

Die Besonderheit der BSMC besteht darin, dass ein einzelner Rechenkern nur eine geringe Anzahl an FPGA-Ressourcen benötigt und durch die effiziente Generierung der Zufallszahlen ein hoher Speedup im Vergleich zum Host-System erreicht werden kann. Beispielsweise kann ein BSMC Hardwaredesign mit 8 Monte-Carlo Kernen in einem einzelnen vFPGA (Single) realisiert werden (siehe Tabelle 6.6). Ein solcher *BSMC-8-vFPGA* erreicht gegenüber einem Prozessor⁵ mit 4 Kernen einen Speedup (siehe Abschnitt 2.1.1.2) von 7,82 bei 100 Millionen Durchläufen [Rus15].

III. Alignment (HDL): Eine weitere Anwendung, die sich aufgrund ihrer einfachen Datentypen ideal auf einem FPGA realisieren lässt, ist das Sequenzalignment von kurzen DNA-Sequenzen (Reads) an ein vollständiges Genom, um deren Position darin zu ermitteln. Das Verfahren, welches als Short Read Mapping Problem (SRMP) [TS09] bekannt ist, wurde in [KPS11; PKS12] auf einen FPGA übertragen. Das vFPGA-Design für einen *Alignment-vFPGA* mit 160 Einheiten, welche jeweils einen Read beinhalten, benötigt vier aggregierte vFPGA-Slots (Quad). Da die Genom-Datenbank nicht lokal auf dem FPGA gehalten werden kann, wird diese konstant über die PCIe-Schnittstelle mit maximal 52 MByte/s gleitet. Der so erzielte Speedup liegt für das Alignment von 500.000 Reads (50 Basenpaare) innerhalb des menschlichen Genoms (3,27 Milliarden Basenpaare) bei 4,52, gegenüber dem eines Prozessors⁵ mit vier Kernen und dem vergleichbaren Basic Local Alignment Search Tool (BLAST) [Alt+90] des National Center for Biotechnology Information (NCBI) [Mad13].

IV. k-Means-vFPGA (HDL): Der kMeans Algorithmus [DG08; JD88] zur Vektorquantisierung, welcher auch zur Clusteranalyse von unterschiedlichsten Daten eingesetzt wird, ist eine typische rechenintensive Anwendung im Bereich des Cloud-Computings [Mis+10]. Dabei wird aus einer Menge von ähnlichen Objekten eine vorher bekannte Anzahl von k Clustern erzeugt. Mit der Übertragung des k-Means Algorithmus auf einen FPGA kann ein hohen Speedup erreicht werden, wie bereits in der Liste der Cloud-Anwendungen in Abschnitt 2.4.2 gezeigt wurde.

Durch Kombination von HLS und HDL ist ein vFPGA-Design realisiert, welches durch lokales Speichern des zu analysierenden Datensatzes einen vergleichsweise großen Speedup auf dem FPGA erreicht. Der in [Knö17] entwickelte *k-Means-vFPGA* benötigt insgesamt für ein Bild der Größe von 640×480 und einer Clusterzahl von $k = 6$ insgesamt sechs aggregierte vFPGA-Slots (Hexa) [Knö17] und erreicht einen Speedup von 8,53 gegenüber einem Prozessors⁵ mit vier Kernen. Durch das Speichern des kompletten Datenatzes auf den FPGA werden 532 BRAMs, verteilt auf sechs vFPGA-Slots (Hexa), benötigt (siehe Tabelle 6.6).

V. Crypto-vFPGA (IP-Core): Des Weiteren ist ein Rechenkern für den Einsatz als kryptographischer Coprozessor (*Crypto-vFPGA*) oder als Endpunkt für eine abgesicherte Datenkommunikation zwischen Client-System und VM-Hostsystem in der Cloud entwickelt, wie es in Abschnitt 3.2.1 und speziell in Abbildung 3.5(f) dargelegt wurde. Wie in [GG13] gezeigt, kann durch Auslagerung einer derartigen Anwendung zur Ver- und Entschlüsselung (AES-CBC-128-SHA1) eines Datenstroms ein moderner Serverprozessor deutlich entlastet werden. Auf Seiten der Cloud bilden dahingehend

die Energieersparnis und die Auslagerung von Rechenlast die Hauptmotivationen für den Einsatz von vFPGAs. Auch auf dem Client-System kann der FPGA als Endpunkt zum Einsatz kommen, um dieses ebenfalls zu entlasten und durch die Nutzung von Spezialhardware Angriffsmöglichkeiten zu reduzieren und somit die Sicherheit bei der Datenübertragung zu erhöhen.

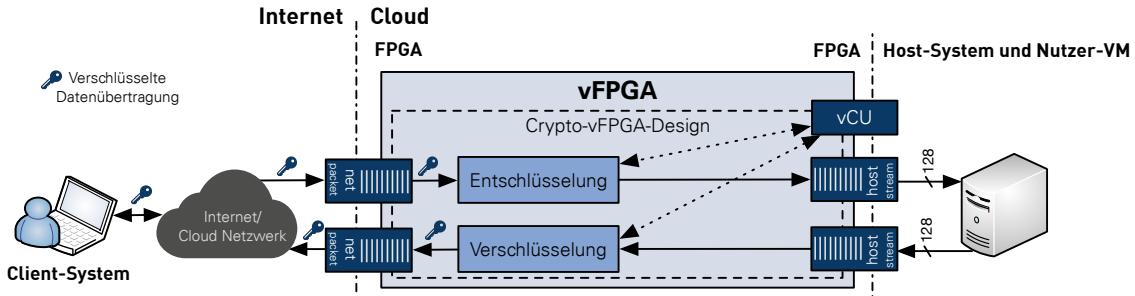


Abbildung 6.15: Einsatz des Crypto-vFPGAs zur Ver- und Entschlüsselung eines Datenstromes.

Als Beispiel für das RC2F dient hierbei ein 256-Bit AES ECB-Core [Rus17], dessen Schnittstellen entsprechend modifiziert wurden, um in einen mittels des RC2F bereitgestellten vFPGA eingebettet werden zu können. Aufgrund des geringen Bedarfes an FPGA-Ressourcen (siehe Tabelle 6.6) benötigt der Rechenkern lediglich einen vFPGA-Slot (Single). Der Crypto-vFPGA entschlüsselt den vom externen Client-System verschlüsselten und über die Gigabit-Ethernet Schnittstelle des FPGA-Boards eingehenden Datenstrom und leitet diesen im Klartext an eine Nutzer-VM auf dem Host-System innerhalb der Cloud weiter (siehe Abbildung 6.15). Das Senden eines Datenstromes aus der VM verläuft äquivalent in die umgekehrte Richtung. Bei der Implementierung übernimmt der FPGA entsprechend auch die Rolle der Netzwerkvirtualisierung für den Host-Hypervisor.

Durch die gewählten Anwendungen ist die Virtualisierung der FPGAs evaluiert. Die vier Anwendungen repräsentieren verschiedene Einsatzmöglichkeiten der vFPGAs und zeigen, wie flexibel diese in ihrem Bedarf an FPGA-Ressourcen und physischem Platz an die Bedürfnisse der Nutzer und Anwendungen angepasst werden können. Die flexible Programmierung mittels HLS und die entsprechende Einbettung in die RC2F-Virtualisierung ist ebenso evaluiert. Des Weiteren können durch eine Kombination der unterschiedlichen Rechenkerne und Anwendungen die verschiedenen Kommunikationskanäle besser ausgelastet werden, da beispielsweise der Matrix-vFPGA (I) die PCIe-Schnittstelle vergleichsweise stark auslastet, der BSMC-vFPGA dagegen lediglich Daten zur Initialisierung und die Ergebnisse überträgt.

Tabelle 6.6: Bedarf an FPGA-Ressourcen und Anzahl der erforderlichen vFPGAs-Slots für die vier Demonstratoren innerhalb des RC2F-Prototypen auf dem Virtex-7, sowie der erreichbare Speedup und die Datenrate zum Hostsystem.

Demonstrator	vFPGA-Slots	Anzahl FPGA-Ressourcen / Anteil des vFPGAs				Datenrate Host / vFPGA	Speedup zur CPU ^a
		Slice LUTs	Slice Registers	BRAM Tiles	DSPs		
I.a) Matrix 16x16	1	25.298/89,08 %	41.654/70,60 %	14/13,33 %	80/23,53 %	509 MByte/s	0,42
I.b) Matrix 32x32	3	64.711/75,95 %	125.715/71,03 %	68/21,59 %	160/15,69 %	279 MByte/s	0,63
II. BSMC-8	1	20.059/70,63 %	33.289/56,42 %	126/120,00 %	70/20,59 %	0,85 MByte/s	7,82
III. Alignment	4	108.436/95,45 %	156.765/66,43 %	338/80,48 %	0/0 %	52 MByte/s	4,52
IV. Crypto	1	10.654/37,51 %	12.054/6,81 %	48/45,71 %	0/0 %	125 MByte/s	1,00
V. k-Means	6	22.592/11,99 %	14.309/3,80 %	532/88,67 %	18/0,93 %	43 MByte/s	8,53

^a Als Referenzsystem dient Rechenknoten 0 des in Abschnitt 6.1.1 vorgestellte Cloud-Prototyp mit einem Intel Core i7-2600K.

6.2.2 Allgemeiner Einsatz der virtualisierten FPGAs

Die unterschiedlichen Demonstratoren, welche in Abschnitt 6.2.1 vorgestellt wurden, haben insbesondere die Untersuchung der effizienten Auslastung des physischen FPGAs durch die direkte Anpassung der vFPGAs an die benötigten FPGA-Ressourcen zum Ziel. Als weiterführendes Szenario für den allgemeinen Einsatz der Virtualisierung innerhalb der Ressourcenverwaltung RC3E zeigt Abschnitt 6.16, wie ein exemplarisches System durch den Einsatz der Migration bezüglich der Auslastung optimiert werden kann. Zwischen den einzelnen Aktionen (A) bis (F) vergehen jeweils 60 Sekunden. Das Szenario wird im nachfolgenden Abschnitt 6.3.3.4 genutzt, um die Flexibilität der Virtualisierung nachzuweisen und mögliche gegenseitige Beeinflussungen der vFPGAs durch ihre Konfiguration aufzuzeigen, beziehungsweise auszuschließen.

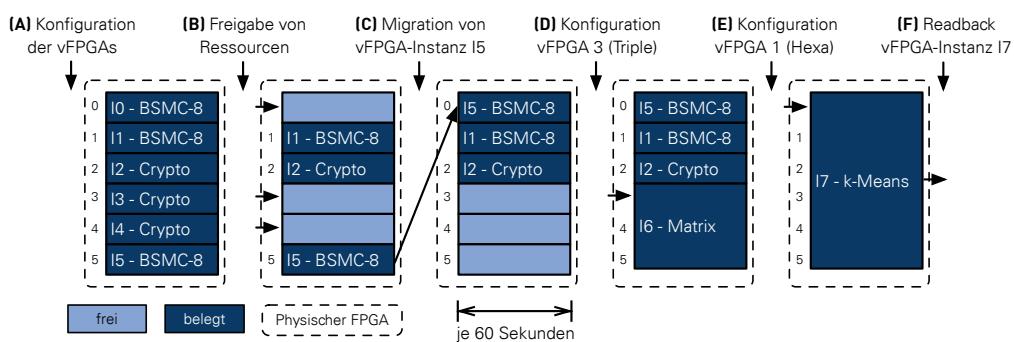


Abbildung 6.16: Szenario für den allgemeinen Einsatz eines mittels RC2F virtualisierten FPGAs zur Optimierung der Auslastung des physischen FPGAs innerhalb der Ressourcenverwaltung RC3E.

6.2.3 Lastszenarien für die Simulation des RC3E

Die RC3E-Simulation wird mit einer Reihe unterschiedlicher Werte für die im vorherigen Abschnitt erläuterten Parameter durchgeführt, wobei die Szenarien für Arbeitspakete in zwei Gruppen eingeteilt werden können. Zum einen werden kryptografische Arbeitspakete mit langer Laufzeit eingesetzt, welche lediglich einen vFPGA (Single) einnehmen und zum anderen Arbeitspaket mit einer vergleichsweise kurzen Laufzeit, welche aber mehrere zusammenhängende vFPGAs benötigen.

Die Simulation untersucht die Dynamik des Systems, was über zwei unterschiedliche Lastszenarien erfolgt. Zum einen werden (I) synthetische Arbeitslasten für unterschiedliche Szenarien und zum anderen (II) die Lastdaten eines Webservers genutzt, um eine Tendenz für reale Lastszenarien abschätzen zu können. In Szenario (I) erfolgt zunächst ein spontaner Anstieg der eingehenden Arbeitspakete (a), gefolgt von einem deutlichen Abfall (b) und einer Phase mit annähernd konstanter Anzahl eingehender Arbeitspakete über einen längeren Zeitraum (c). Die Zeitpunkte und die Anzahl der eintreffenden Arbeitspakete ist in Abbildung 6.17 aufgezeigt.

Das Lastszenario (II) mit realen Anfragen auf Basis eines Webservers [ITA16], welche über einen Zeitraum von 24 Stunden (1.440 Minuten) eintreffen, ist in Abbildung 6.18 gezeigt. Die 47.685 Arbeitspakete haben feste Zeitpunkte, zu denen sie vom RC3E-Simulator entgegengenommen werden. Die Arbeitspakete werden des Weiteren um Laufzeit und die Beschreibung der Ressourcen ergänzt.

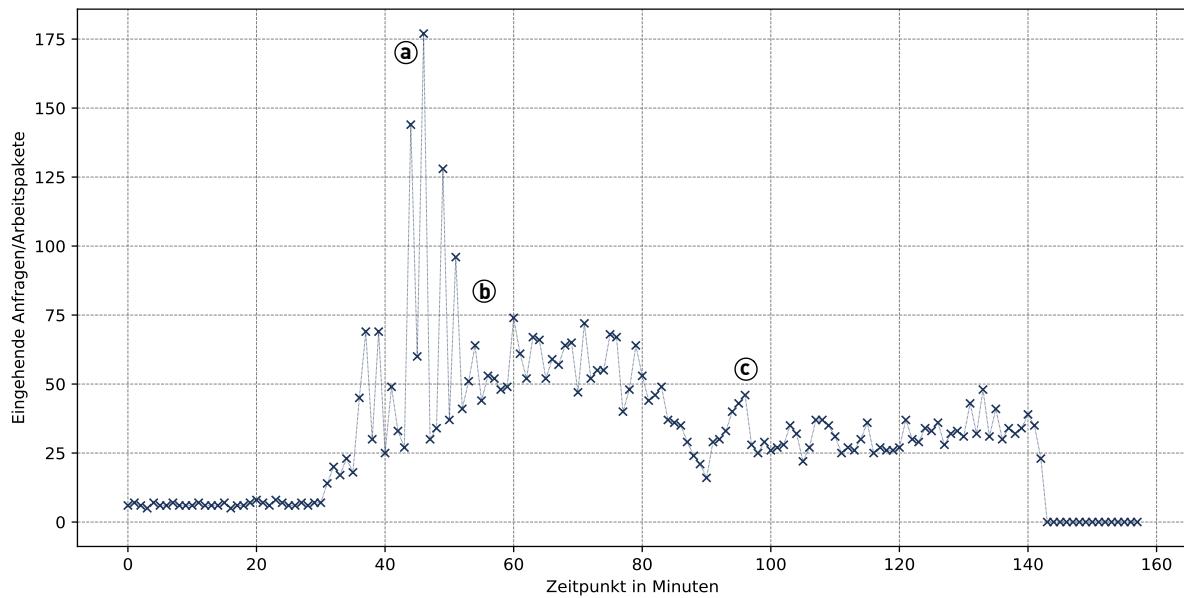


Abbildung 6.17: Lastszenario (I) mit einer synthetischen Arbeitslast von 4.981 Arbeitspaketen über einen Zeitraum von 150 Minuten. Es erfolgt zunächst ein spontaner Anstieg der eingehenden Arbeitspakete (a), gefolgt von einem deutlichen Abfall (b) und einer Phase mit annähernd konstanter Anzahl eingehender Arbeitspakete über einen längeren Zeitraum (c).

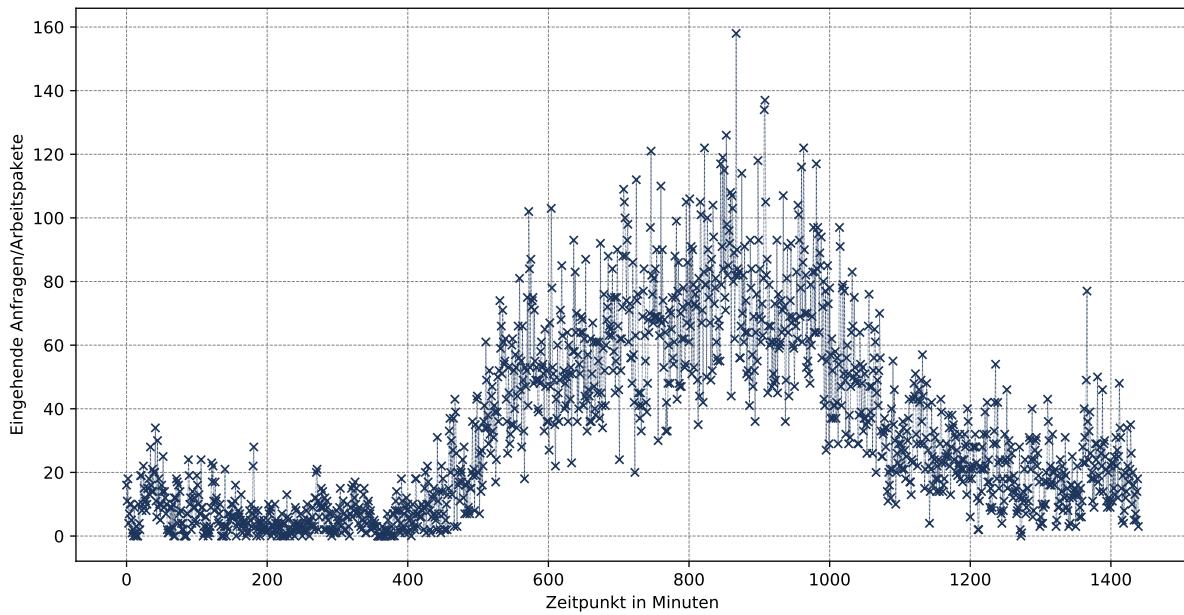


Abbildung 6.18: Lastszenario (II) mit 47.748 Anfragen auf Basis eines Webservers über einen Zeitraum von 24 Stunden [ITA16].

6.2.3.1 Arbeitspakete innerhalb der Lastszenarien

Für die beiden Lastszenarien werden jeweils Arbeitspakete basierend auf den vier in Abschnitt 6.2 eingeführten Demonstratoren generiert. Die Demonstratoren sind entsprechend in sechs unterschiedliche Arbeitspakete gemäß Tabelle 6.6 eingeteilt. Ein weiterer Parameter nach Abschnitt 6.1.3.3 ist die Gesamtaufzeit $t_{Arbeitspaket}$, welche gemäß einer Gleichverteilung zwischen 10 und 300 Sekunden liegt. Die durchschnittliche Gesamtaufzeit über alle Arbeitspakete liegt entsprechend bei 2,5 Minuten. Die Anzahl der benötigten vFPGA-Slots $N_{vFPGA-Slots}$ ergibt sich ebenfalls gemäß der Demonstratoren nach Tabelle 6.6, was eine durchschnittliche Belegung von 26,67 % beziehungsweise zwei vFPGA-Slots entspricht. Die Anzahl der erforderlichen Prozessorkerne N_{Kerne} beschränkt sich je Arbeitspaket auf einen.

6.2.3.2 Simulationsparameter und Systemkonfigurationen der simulierten Cloud

Das zu simulierende System ist angelehnt an den in Abschnitt 6.1.1 beschriebenen Cloud-Prototypen. Die Parameter innerhalb des RC3E-Simulators, wie sie in Abschnitt 6.1.3.2 eingeführt wurden, sind im Einzelnen:

- $Max_{Knoten} = 100$
- $P_{FPGA-Last} = 24 \text{ W}$
- $t_{Knoten-Stop} = 20 \text{ s}$
- $Max_{Kerne} = 8$
- $P_{FPGA-Leerlauf} = 20 \text{ W}$
- $t_{Migration} = 1 \text{ Minute}$
- $Max_{FPGAs} = 2$
- $N_{Warteschlange} = 200$
- $t_{Migration-FPGA} = 10 \text{ s}$
- $Max_{vFPGA-Slots} = 1 \text{ oder } 6$
- $N_{Waiting} = 20 \text{ s}$
- $\rho_{FPGA} = 16 \%$
- $P_{CPU-Last} = 100 \text{ W}$
- $t_{Knoten-Start} = 10 \text{ s}$
- $t_{SLA} = 1,5 \text{ s}$
- $P_{CPU-Leerlauf} = 20 \text{ W}$
- $\rho_{Knoten} = 20 \%$

Die RC3E-Simulation ist für drei unterschiedliche Systemkonfigurationen zu evaluieren:

- (1) Basis: FPGA-Cloud ohne Virtualisierung der FPGA-Ressourcen,
- (2) RC2F-Virtualisierung der FPGAs ($Max_{vFPGA-Slots} = 6$) und
- (3) Zusätzliche Migration der Virtualisierten FPGAs.

6.3 Evaluation und Validierung der Prototypen

Im Folgenden werden sowohl der Cloud-Prototyp RC3E als auch die prototypische FPGA-Virtualisierung RC2F anhand der zuvor in Abschnitt 6.2 eingeführten Demonstratoren und Szenarien evaluiert. Zunächst wird in Abschnitt 6.3.1 die Vorgehensweise in Bezug auf den Test der Einzelkomponenten innerhalb der Ressourcenverwaltung RC3E skizziert. Der zur Evaluierung einer Cloud-Architektur entwickelte RC3E-Simulator aus Abschnitt 6.1.3 wird in Abschnitt 6.3.2 genutzt, um die Ressourcenverwaltung weiterführend zu evaluieren. Die zuvor in Abschnitt 6.2 aufgezeigten Demonstratoren und das darauf aufbauende Anwendungszenario für den allgemeinen Einsatz der virtualisierten FPGAs aus Abschnitt 6.2.2 werden abschließend in Abschnitt 6.3.3 zur Validierung und Evaluation des RC2F-Prototypen aus Abschnitt 6.1.4 herangezogen, um ausgewählte Einsatzmöglichkeiten zu veranschaulichen.

6.3.1 Validierung des RC3E-Prototypen auf Ebene der Cloud-Verwaltung

Der RC3E-Prototyp wurde mittels ausgewählter Testfälle und komplexer Anwendungsszenarien untersucht und auf seine Eignung für die in Abschnitt 3.2.3 formulierten Anforderungen hin evaluiert beziehungsweise validiert [Bal98, S. 101]. Dabei ist insbesondere der Test der Einzelkomponenten von großer Bedeutung, um die entsprechende Reaktion des Systems zu beobachten und ein Fehlverhalten auszuschließen.

Der RC2F Host-Hypervisor ist durch einen Testlauf über mehrere Jahre mit realen Nutzern für die Servicemodelle RSaaS und RAaaS exemplarisch evaluiert. Indem des Weiteren konkurrierende Zugriffe auf die physischen (v)FPGA-Ressourcen durchgeführt wurden, konnte dabei ein Fehlverhalten ausgeschlossen werden. Insbesondere bezüglich der Zugriffsrechte und der wesentlichen Abläufe kann im Ergebnis von einem stabilen System ausgegangen werden und die in Abschnitt 5.2 erarbeiteten Abläufe sind in der Praxis umgesetzt. Lediglich die Lastverteilung innerhalb des Modells BAaaS wurde durch die RC3E-Simulation (siehe nachfolgend Abschnitt 6.3.2) validiert und evaluiert.

6.3.2 Evaluation der Ressourcenverwaltung durch die RC3E-Simulation

Um das Verhalten der Ressourcenverwaltung und insbesondere sowohl den Nutzen der virtualisierten FPGAs als auch deren Migration in einer Cloud zu evaluieren, wird der RC3E-Simulator aus Abschnitt 6.1.3.1 eingesetzt. Die durch den Cloud-Prototypen RC2F in Abschnitt 6.1.4 ermittelten Leistungsdaten bilden dabei die Basis, um das entsprechende Verhalten innerhalb einer größeren Cloud mit mehreren Rechenknoten durch den Simulator nachzubilden und zu untersuchen. Betrachtet wird dabei im Speziellen die Zuordnung von vFPGAs zu Rechenknoten und zu auf diesen verfügbaren vFPGA-Slots.

Die Ergebnisse der Simulation auf Basis der in Abschnitt 6.2.3.2 eingeführten Simulationsparameter und Systemkonfigurationen sowie den in Abschnitt 6.2.3 vorgestellten Lastszenarien sind in Tabelle 6.7 aufgezeigt. Die Tabelle zeigt neben der durchschnittlichen Anzahl der allokierten Rechenknoten deren Energiebedarf sowie die Auslastung der dem Nutzer zur Verfügung stehenden FPGA-Ressourcen. Das Service Level Agreement (SLA) gibt zusätzlich an, welcher Anteil der Arbeitspakete innerhalb einer bestimmten Zeitspanne (2,5 s) bearbeitet wird, um das Systemverhalten in Bezug auf die Qualität der Bereitstellung der Ressourcen bewerten zu können.

Die Auslastung der FPGAs liegt in der Simulation, wie in Abschnitt 6.2.3.1 gefordert, bei den Referenzsystemen mit FPGAs ohne Virtualisierung (1) bei circa 27 % (siehe Tabelle 6.7). Der Energiebedarf der Cloud wird in beiden Szenarien durch die Virtualisierung (2) um zusätzliche 64,78 % gegenüber dem FPGA-System (1) für Szenario (I) und 60,25 % für Szenario (II) gesenkt. Entsprechend sinkt durch eine Steigerung der Auslastung der Energiebedarf. Das SLA steigt leicht um 0,04, beziehungsweise 0,02 an, da die virtualisierten Ressourcen schneller verfügbar sind als ein neuer Rechenknoten. Die zusätzliche Migration (3) trägt nur unwesentlich zur Einsparung von Ressourcen und Energie bei. Durch den Prozess der Migration wird allerdings das SLA beeinträchtigt, da die Migration eine entsprechend hohe Priorität hat und neue Ressourcen erst verzögert bereitgestellt werden. Lastszenario (III) liefert in den Konfigurationen (2) und (3) bessere Werte, da keine Extremwertsituationen, wie in Lastszenario (I), auftreten.

Zusätzlich zu Tabelle 6.7 zeigt Abbildung 6.19 exemplarisch das Ergebnis einer RC3E-Simulation für Lastszenario (I). In der Grafik sind die allokierten vFPGA-Slots, deren Auslastung sowie die Zeitpunkte, zu denen neue Rechenknoten allokiert werden oder Migrationen stattfinden, veranschaulicht. Die Migrationen (d) finden verstärkt bei neu eintreffenden Arbeitspaketen (a) statt, da vor dem Hinzufügen neuer Rechenknoten zunächst versucht wird, bestehende vFPGAs zu migrieren.

6 Prototypische Implementierung und Ergebnisse

Tabelle 6.7: Ergebnisse der RC3E-Simulation für die Systemkonfigurationen (1) zusätzliche FPGAs (+FPGA), (2) RC2F FPGA-Virtualisierung (+RC2F) und (3) zusätzlicher Migration (+Mig). Die Szenarien sind (I) eine synthetische Arbeitslast und (II) die Lastdaten eines realen Webservers [ITA16].

	Lastszenario (I) – 4.981 Arbeitspakete über 150 Minuten				Lastszenario (II) – 47.748 Arbeitspakete über 1.440 Minuten			
	Cloud ^a	+FPGA (1)	+RC2F (2)	+Mig (3)	Cloud ^a	+FPGA (1)	+RC2F (2)	+Mig(3)
	Rechenknoten ^b	357	132	26	24	376	128	25
FPGA Auslastung (%)	—	27,34	78,14	85,07	—	26,74	94,24	97,82
Energiebedarf (kWh)	35,37	24,53	8,64	8,13	287,37	225,12	89,48	83,06
Energiebedarf (%)	100,00	69,35	24,43	22,99	100,00	78,34	31,14	28,90
SLA ^c	0,91	0,87	0,91	0,85	0,96	0,90	0,92	0,91

^a FPGA-Cloud ohne Virtualisierung der FPGA-Ressourcen.

^b Durchschnittliche Anzahl der allokierten Rechenknoten.

^c Anteil der Arbeitspakete, welche innerhalb von 2,5 s bearbeitet werden.

Bei einem Einbruch der Arbeitslast (b) findet die Migration nur noch bei den Arbeitspaketen mit längerer Laufzeit statt, da die Freigabe der Ressourcen erst spät erfolgt. Die zusätzlichen Einsparungen gegenüber der reinen Virtualisierung (2) betragen durch die Migration (3) weitere 2,08 % für Szenario (I) und 2,85 % für Szenario (II) verglichen mit dem FPGA-System (1).

Aufgrund der Ergebnisse der RC3E-Simulation kann davon ausgegangen werden, dass sowohl die Virtualisierung als auch die damit verbundene Migration von vFPGAs zu Einsparungen von Ressourcen und somit Energie führen können (siehe Abbildung 6.20), ohne das SLA deutlich zu verringern. Die hohen Einsparungen sind mit den gewählten Demonstratoren und den darauf basierenden Arbeitspaketen (siehe Abschnitt 6.2.3.1) zu erklären. Lasten die vFPGA-Designs den physischen FPGA vollständig aus, können durch die Virtualisierung keine Rechenknoten eingespart und entsprechend der Energiebedarf reduziert werden. Die Migration ermöglicht jedoch die Verlagerung der vFPGA-Images auf andere Rechenknoten und bietet somit die Möglichkeit, zu Wartungszwecken Teile des Systems örtlich zu verschieben.

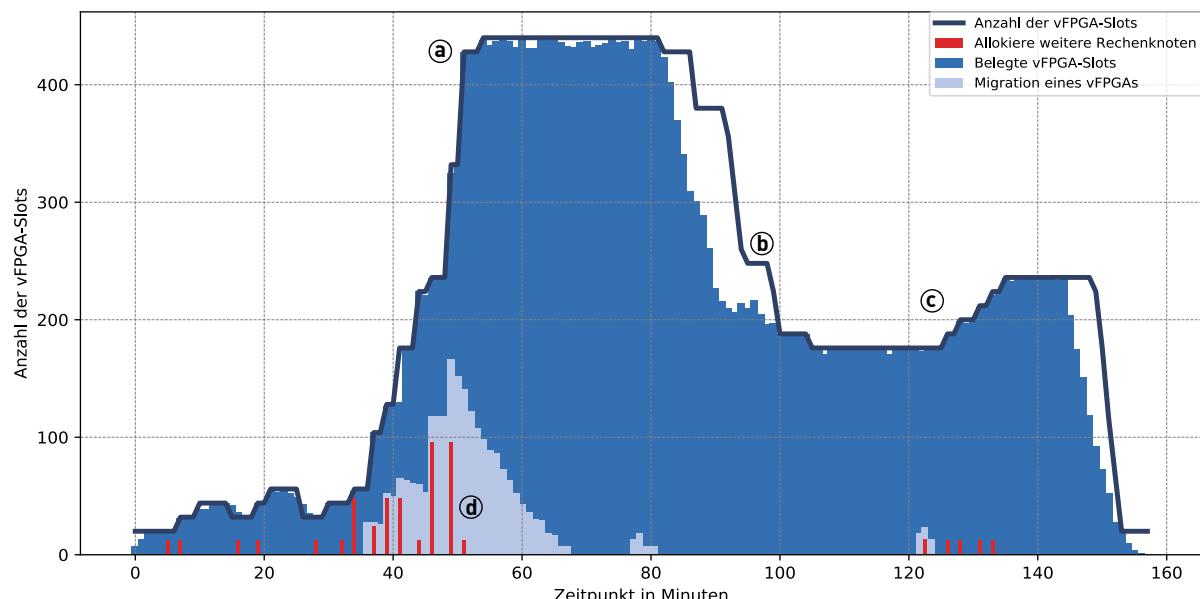


Abbildung 6.19: RC3E-Simulation für Lastszenario (I) mit FPGA-Virtualisierung und Migration (3).

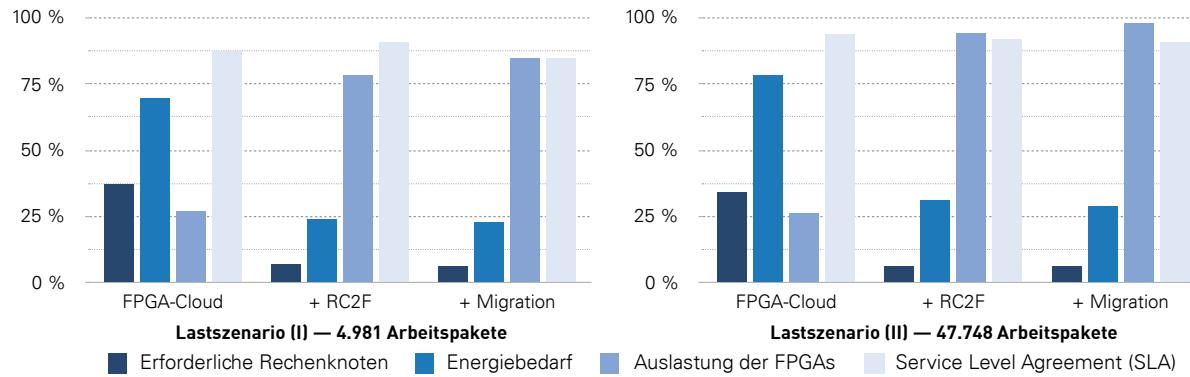


Abbildung 6.20: Vergleich der unterschiedlichen Systemkonfigurationen innerhalb der RC3E-Simulation.

Neben der Evaluation, wie sich Virtualisierung und Migration auf die Optimierung von Auslastung und Energieverbrauch auswirken, wurde des Weiteren durch den RC3E-Simulator die Zuordnung von vFPGAs zu physischen FPGAs und den Rechenknoten validiert.

6.3.3 Evaluation des Lebenszyklus eines virtualisierten FPGAs

Eine Validierung des RC2F und der API erfolgt im Einzelnen zunächst in Abschnitt 6.3.3.1 durch die Erzeugung der migrierbaren vFPGA-Images auf Basis der vFPGA-Designs der vier in Abschnitt 6.2.1 vorgestellten Demonstratoren. Die Validierung der Kontextmigration anhand einfacher Register und Speicher erfolgt in Abschnitt 6.3.3.3. Darauf folgend wird in Abschnitt 6.3.3.2 der Test der konkurrierenden Datentransfers zwischen Host und vFPGAs innerhalb des physischen Cloud-Prototypen demonstriert, bevor Abschnitt 6.3.3.4 mit dem Szenario zum allgemeinen Einsatz der vFPGAs den Abschnitt abschließt.

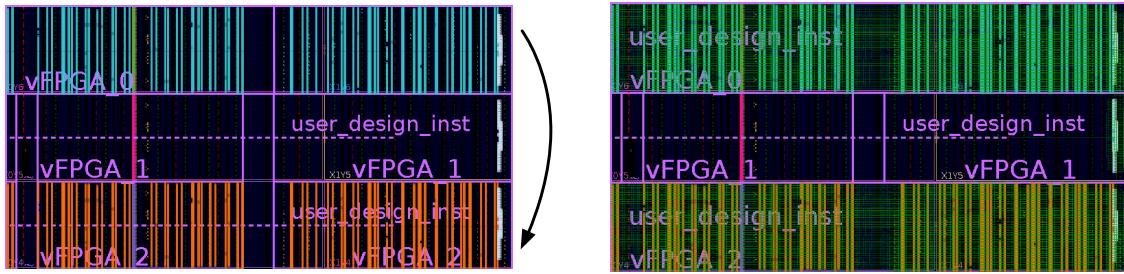
6.3.3.1 Erzeugung der vRAI-Pakete

Der in Abschnitt 6.1.4.7 eingeführte Entwurfsablauf zur Erzeugung homogener vFPGAs, welche sich für eine Migration eignen, ist entsprechend für die in Abschnitt 6.2.1 vorgestellten Demonstratoren durchgeführt. Abbildung 6.21 zeigt am Beispiel eines BSMC-vFPGA-Designs das Kopieren der platzierten Komponenten (Slice Register, Slice LUTs, BRAM Tiles, DSPs) der wesentlichen Schritte innerhalb des Entwurfsprozesses. Das darauf folgende Routing und die anschließende Generierung des Bitstreams erfolgte mittels des vom Hersteller vorgegebenen Entwurfsablaufes.

Tabelle 6.8: Paketgrößen in MByte für die einzelnen vFPGAIImages und vRAIs für die vFPGA-Demonstratoren.

I. Matrix 16×16	I. Matrix 32×32	II. BSMC-8	III. Alignment	IV. Crypto	V. k-Means
Bitstream (MByte)	4,8	13,0	4,8	17,3	4,8
vRAI (MByte)	57,6	104	57,6	103,8	57,6

Die unterschiedlichen Bitstreams beziehungsweise vFPGA-Images sind, wie in Abschnitt 6.1.4.7 beschrieben, mit den zusätzlichen Bitmasken (VCBMs) zu vRAIs zusammengefasst. Die sich ergebenden Dateigrößen für die erzeugten vFPGA-Images und die vRAIs sind in Tabelle 6.8 aufgezeigt und auf der physischen Hardware hinsichtlich ihrer Funktionsfähigkeit validiert.



(a) Schritt 2: Übertragen der relativen Positionen (LOC, BEL) der platzierten Komponenten in weitere (aggregierte) vFPGA-Slots.

(b) Schritt 3: Vergrößerung des homogenen vFPGA-Slots auf die volle Breite der inhomogenen vFPGA-Slots und vollständiges Routing der vFPGAs.

Abbildung 6.21: Teilschritte des Entwurfsablaufes zur Erzeugung homogener vFPGA-Images am Beispiel der BSMC. Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 [Xil16g].

6.3.3.2 Validierung des RC2F-Prototypen

Auf Seiten der FPGA-Virtualisierung RC2F erfolgte ein Test der in Abschnitt 6.1.4 erläuterten prototypischen Implementierung auf der physischen Hardware (Xilinx Virtex-7 XCVU9P). Sowohl die Kommunikation und die Interaktion des RC2F Host- und des FPGA-Hypervisors über die FPGA-Konfigurationsspeicher Hypervisor Control Unit (HCU) und vFPGA Control Unit (vCU) (siehe Abschnitt 4.4.1.3 und Abschnitt 4.4.1.2) sind innerhalb des in Abschnitt 6.1 dargestellten Versuchsaufbaus mit den in Abschnitt 6.2.1 entwickelten Demonstratoren und dem in Abschnitt 6.2.2 aufgezeigte Szenario auf der physischen Hardware evaluiert. Insbesondere beim Zusammenspiel zwischen RC2F Host- und FPGA-Hypervisor können Verstöße, wie die Konfiguration oder das Auslesen fremder vFPGAs ausgeschlossen werden. Der Test der physischen Hardware und des Zusammenspiels innerhalb des Cloud-Prototypen, welcher in Abschnitt 6.1 vorgestellt wurde, erfolgt sowohl in der RTL-Simulation für die FPGA-Virtualisierung RC2F und insbesondere den FPGA-Hypervisor, als auch innerhalb des Cloud-Testsystems (siehe Abschnitt 6.1.1).

Hinsichtlich der Kommunikation zwischen Host und FPGA wurde beispielsweise mittels nebenläufiger *Loopback-vFPGAs* (siehe auch Abbildung C.1), welche die eingehenden Daten innerhalb der vFPGAs umgehend an den Host zurücksenden, sowohl die erreichbaren Datenraten als auch die Korrektheit der Daten innerhalb des Cloud-Prototypen (siehe refsecsec:ergebnis:impl:cloud) überprüft. Insbesondere durch die konkurrierenden vFPGAs konnte die Funktion der Arbitrierung innerhalb des FPGA-Hypervisors validiert werden. Die erreichbare Datenrate der Kommunikation zwischen Host und einer unterschiedlichen Anzahl vFPGAs ist in Abbildung 6.22 aufgezeigt. Die maximal erreichbare Datenrate innerhalb des physischen Testsystems mit mit PCIe Gen2x8 und dem Virtex-7 FPGA liegt bei 3,5 GByte/s [Xil16j].

Im Testaufbau ist jeweils eine unterschiedliche Anzahl nebenläufiger vFPGAs aktiv, und in Abbildung 6.22 ist jeweils der Netto-Datendurchsatz für einen exemplarischen vFPGA aufgezeigt. Verringert sich die Anzahl der aktiven vFPGAs, steigen entsprechend die Datendurchsätze der verbleibenden vFPGAs an. Der Datendurchsatz eines einzelnen aktiven vFPGAs ist in diesem Beispiel aufgrund dessen Taktfrequenz von 100 MHz auf 1.604,74 MByte begrenzt. Erst ab zwei aktiven vFPGAs setzt eine Begrenzung des Datendurchsatzes auf 800 MByte/s ein. Der aufgrund des PCIe-Controllers maximal gemessene Datendurchsatz liegt bei 2.746,93 MByte/s und ist in Abbildung 6.22 als maximal aggregierter Datendurchsatz aufgezeigt. Alle weiteren Messkurven ab drei vFPGAs zeigen die Begrenzung des Datendurchsatzes aufgrund der konkurrierenden vFPGAs. Der maximale Datendurchsatz setzt ab einer übertragenen Datenmenge von 128 MByte ein.

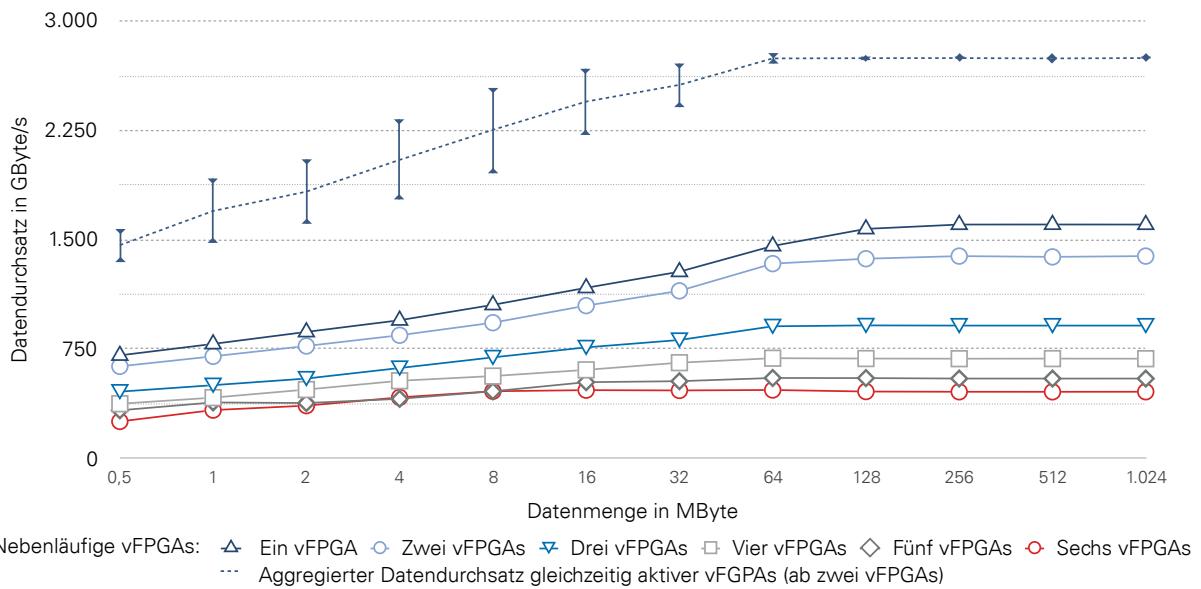


Abbildung 6.22: Erreichbarer Netto-Datendurchsatz zwischen Host und vFPGAs in Abhängigkeit der Anzahl vFPGAs.

6.3.3.3 Validierung der Kontextmigration der vFPGA-Instanzen

Die Validierung der Kontextmigration ist anhand der Migration von Register-, LUT-RAM- und BRAM-Inhalten auf dem Virtex-7 innerhalb des Cloud-Prototypen (siehe Abschnitt 6.1.2) mit dem in Abschnitt 6.1.4.7 beschriebenen Entwurfsablauf durchgeführt. Die drei wesentlichen Schritte sind dabei:

1. Auslesen der vFPGA-Instanz (Readback),
2. Übertragen auf einen anderen physischen vFPGA oder FPGA mit Hilfe der VCBM (Relocate) und
3. Konfiguration der vFPGA-Slots mit der neuen vFPGA-Instanz.

Für den Test auf der physischen Hardware wurden über die RC2F-API Daten in den dynamisch rekonfigurierbaren Bereich der vCSs innerhalb eines vFPGAs geschrieben, welche innerhalb der vFPGA-Designs sowohl als einfache Register, LUT-RAMs und BRAMs realisiert sind. Anschließend wird das vFPGA-Image ausgelesen (1), und die Inhalte werden entsprechend über die VCBM extrahiert und auf Korrektheit überprüft. Daraufhin werden die Inhalte über eine weitere VCBM in ein anderes vFPGA-Image übertragen (2). Die somit erzeugte vFPGA-Instanz wird auf eine andere physische Position als die ursprüngliche vFPGA-Instanz übertragen (konfiguriert) (3). Das darauf folgende Auslesen und Überprüfen der Register-, LUT-RAM- und BRAM-Inhalte über die RC2F-API lieferte die Daten, welche initial innerhalb des vFPGAs auf einer anderen physischen Position innerhalb des FPGAs geschrieben sind, womit die Funktion anhand eines einfachen Beispiels nachgewiesen wurde.

Tabelle 6.9 und Abbildung 6.23 zeigen die gemessenen Zeiten sowohl für die Konfiguration als auch das für das Auslesen des Kontextes und das Migrieren des Bitstreams für unterschiedliche Größen der vFPGAs. In allen Fällen erfolgte die Migration auf einem FPGA innerhalb des gleichen Rechenknotens, und eine entsprechende Übertragung der vFPGA-Instanzen über das Netzwerk entfiel. Die vFPGA-Designs sind die Demonstratoren aus Abschnitt 6.2, wobei das BSMC-vFPGA-Design entsprechend für die vFPGA-Größen Dual und Quint in der Anzahl der Kerne angepasst ist. Durch die Arbeit mit Masken auf den ausgelesenen vFPGA-Instanzen verhält sich die Laufzeit annähernd linear und die FPGA-

6 Prototypische Implementierung und Ergebnisse

Tabelle 6.9: Laufzeiten in Sekunden (s) für die Rekonfiguration und die Migration unterschiedlich großer vFPGA-Instanzen.

Arbeitsschritt	Größe des vFPGAs					
	Single	Dual	Triple	Quad	Quint	Hexa
(1) Readback (s)	0,76	1,43	2,07	2,75	3,39	4,03
(2) Relocate (s)	0,0528	0,078	0,102	0,1278	0,1518	0,1758
(3) Konfiguration (s)	0,044	0,065	0,085	0,1065	0,1265	0,1465
Gesamte Migration (s)	1,72	3,15	4,51	5,98	7,34	8,79

Ressourcen des zugrundeliegende vFPGA-Images haben keinen Einfluss. Die Zeit für das Auslesen (Readback) der vFPGA-Instanzen (des partiellen Bitstreams) liegt mit bis zu einer Sekunde deutlich höher als die Zeit zur Konfiguration des vFPGAs, da das Auslesen spaltenweise innerhalb der Konfigurations-Frames erfolgt.

Ein weiterer wichtiger Aspekt besteht in den verwendeten DSPs, da die internen Pipelining-Register nicht über den Bitstream ausgelesen werden können, wie bereits in Abschnitt 4.4.2 erläutert. Der Nutzer muss bei aktuellen FPGA-Architekturen entsprechende Vorkehrungen treffen, um eine Migration zu ermöglichen.

Die prototypische Implementierung hat hierbei gezeigt, dass das Auslesen und Migrieren eines vFPGA-Kontextes möglich ist. Es wurde bewiesen, dass selbst bei einer großen vFPGA-Instanz (Hexa) weniger als 8,79 Sekunden für eine komplette Migration benötigt werden. Die ebenso erforderliche Migration eines korrespondierenden Inhaltes des DDR-Speichers des FPGA-Boards dauert bei einer der vFPGA-Instanzen zugeordneten Datenmenge von 1 GByte lediglich 2,34 Sekunden.

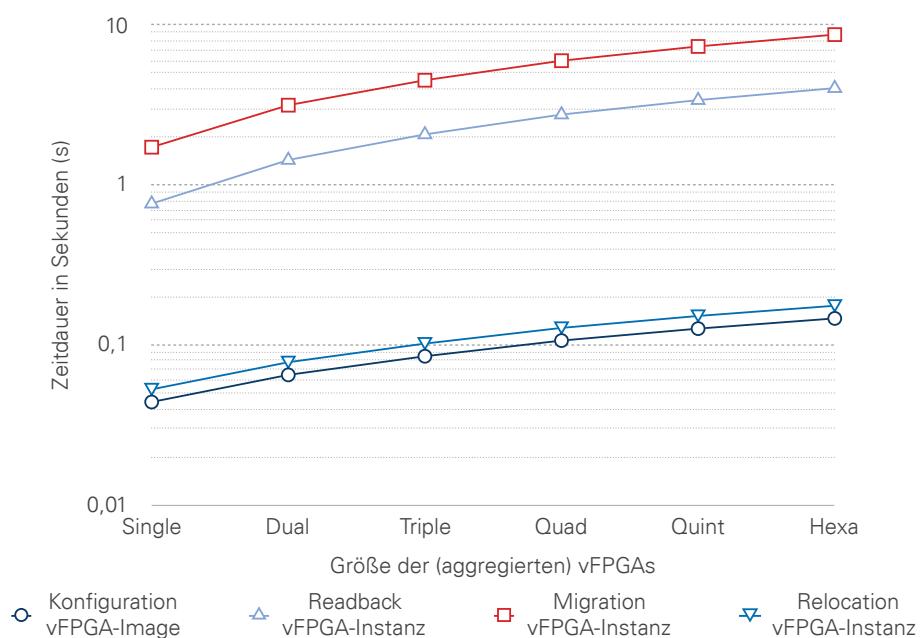


Abbildung 6.23: Zeiten für die Migration unterschiedlich Großer vFPGA-Instanzen (innerhalb des selben Rechenknotens).

6.3.3.4 Kontextmigration innerhalb eines Szenarios für den allgemeinen Einsatz der FPGAs

Der allgemeine Einsatz von virtualisierten FPGAs, wie er in Abschnitt 6.2.2 in Form eines Szenarios skizziert wurde, wird im Folgenden anhand des in Abbildung 6.16 skizzierten Ablaufes vorgestellt. Abbildung 6.25 zeigt unterschiedliche vFPGA-Images/Instanzen innerhalb des RC2F-Prototypen auf dem Virtex-7 FPGA. Abbildung 6.25(a) stellt zunächst ein platziertes Hardwaredesign dar, bei dem lediglich ein Nutzer ein vFPGA-Design samt Schnittstellen innerhalb des Servicemodells RSaaS realisiert hat. Die Auslastung der FPGA-Ressourcen des gesamten FPGAs liegt bei 63 %.

Um die Auslastung der Hardware für den Cloud-Einsatz zu maximieren, kann durch die RC2F-Virtualisierung in einem ersten Schritt (A) eine Belegung wie in Abbildung 6.25(b) aufgezeigt erzielt werden, wobei aus den zuvor in Abschnitt 6.2 erläuterten Anwendungen sowohl der BSMC- als auch der Crypto-vFPGA jeweils einen vFPGA-Slot (Single) benötigen. Durch die Freigabe mehrere vFPGA-Slots (B) entsteht ein, wie in Abbildung 6.25(c) dargestellter, fragmentierter physischer FPGA mit einer Auslastung von 50% und zwei benachbarten freien vFPGA-Slots (0,5 / 2). Die Auslastung lässt zwar einen vFPGA über drei vFPGA-Slots (Triple) zu, allerdings verhindert die Fragmentierung eine entsprechende Platzierung. Das Problem kann von der RC3E-Ressourcenverwaltung durch Migration der vFPGA-Instanz von vFPGA-Slot 5 auf vFPGA-Slot 0 (C) gelöst werden, und es entsteht die in Abbildung 6.25(d) gezeigte Belegung. Die erforderliche Zeit für die Migration innerhalb eines FPGAs des selben Rechenknotens beträgt dabei 1,72 Sekunden.

Anschließend wird der Bereich ab vFPGA-Slot 3 durch ein vFPGA-Image der Größe Triple belegt (D) und es wird eine Auslastung von 100 % erreicht, wie in Abbildung 6.25(e) gezeigt. Um das Szenario abzuschließen, werden alle vFPGAs entfernt und der größtmögliche vFPGA (Hexa) mit dem k-Means-vFPGA konfiguriert (E), wie in Abbildung 6.25(f) gezeigt. Als letzter Schritt folgt das Auslesen der vFPGA-Instanz des k-Menas-vFPGAs (F) innerhalb von 4,03 Sekunden.

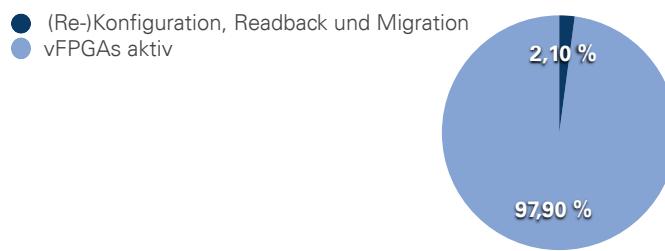


Abbildung 6.24: Verhältnis von Konfiguration und Migration zur Laufzeit der Arbeitspakete.

Werden sämtliche Verwaltungsschritte wie Konfiguration der vFPGA-Slots, Entfernen von vFPGA-Instanzen, Migration und Auslesen aufsummiert und einer Laufzeit der Arbeitspakete von je einer Sekunde zwischen den Schritten (A) bis (F) gegenübergestellt, ergibt sich das in Abbildung 6.24 gezeigte Verhältnis. Die Verwaltungsaufgaben nehmen entsprechend einen Anteil von lediglich 2,10 % ein. Des Weiteren wurde innerhalb des Szenarios gezeigt, dass sich die vFPGA-Instanzen der unterschiedlichen Nutzer nicht beeinflussen.

6 Prototypische Implementierung und Ergebnisse

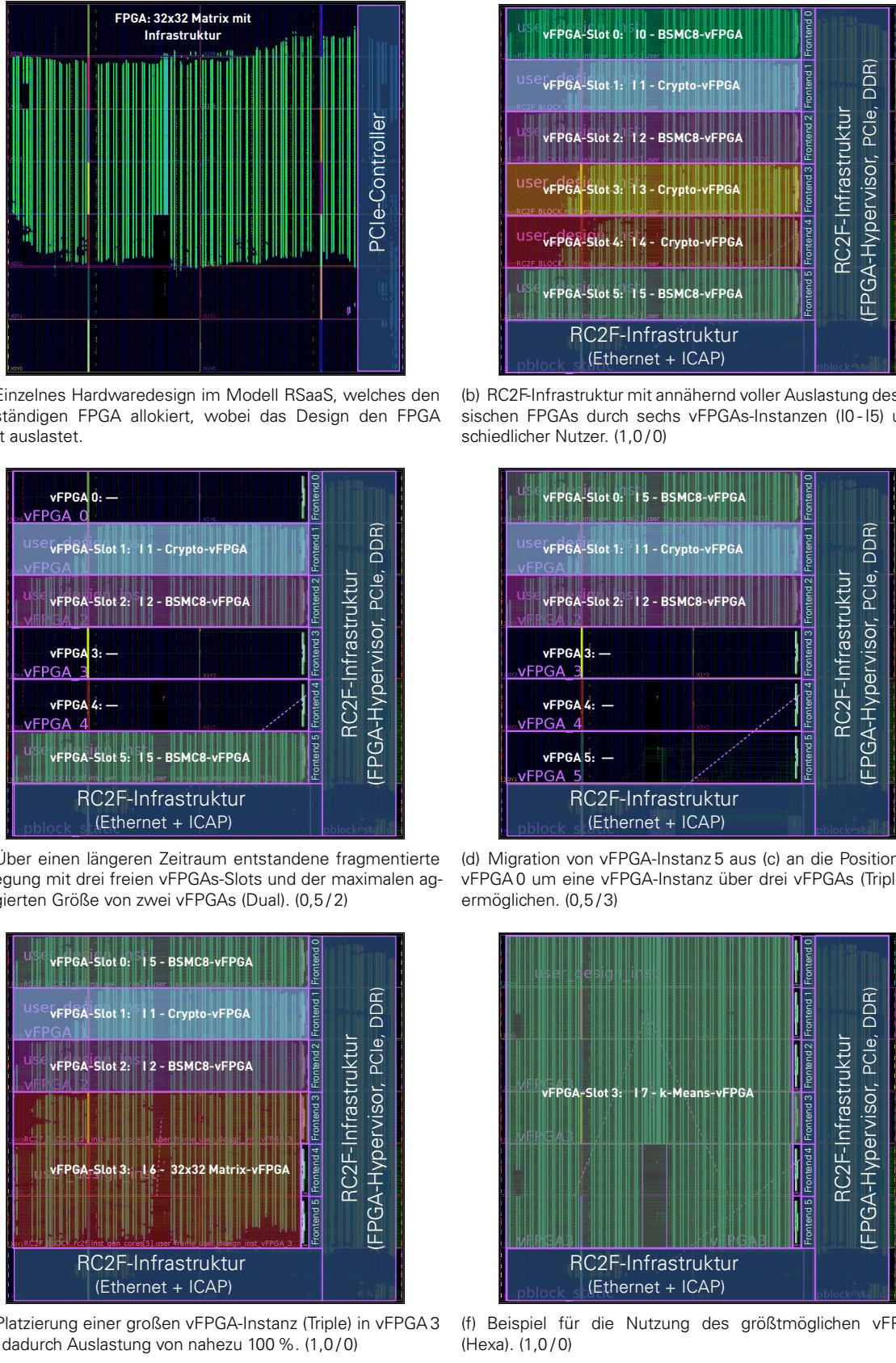


Abbildung 6.25: Szenario aus Abschnitt 6.2.2 mit unterschiedlich großen vFPGAs, deren Auslastung (Auslastung / größte verfügbare Region) nach Abschnitt 5.2.4 sowie Migration unter Verwendung der Demonstratoren aus Abschnitt 6.2.1. Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 von Xilinx [Xil16g].

6.4 Abschließende Bewertung und weiterführende Betrachtungen

Die prototypische Virtualisierung von FPGAs und deren Verwaltung innerhalb der prototypischen RC3E-Cloud, wurde in den vorherigen Abschnitten erläutert, umgesetzt und evaluiert. Im Folgenden wird der RC2F-Prototyp in Abschnitt 6.4.1 abschließend bewertet. In Abschnitt 6.4.2 wird eine Abschätzung zum Bedarf an FPGA-Ressourcen des RC2F auf einem Virtex-7 UltraScale+ FPGA gegeben. In Abschnitt 6.1.3 erfolgt eine Bewertung hinsichtlich der Ergebnisse der RC3E-Simulation. Zusammenfassend wird in Abschnitt 6.4.4 auf Basis der Ergebnisse ein Vergleich zu anderen Arbeiten auf dem Gebiet gezogen.

6.4.1 Erkenntnisse der FPGA-Virtualisierung innerhalb des Cloud-Prototypen

Die in Kapitel 4 vorgestellte FPGA-Virtualisierung RC2F, welche dort prototypisch auf einem handelsüblichen FPGA implementiert wird, hat gezeigt, dass der Ansatz der Virtualisierung umsetzbar ist. Die Auslastung der physischen Hardware kann insbesondere bei kleinen Hardwaredesigns erhöht werden, wie in der RC2F-Simulation in Abschnitt 6.3.2 gezeigt. Die Anpassung der vFPGAs an die Anforderungen der Nutzerdesigns (vFPGA-Designs) wurde ebenso gezeigt wie die Beschreibung von vFPGA-Instanzen (siehe Abschnitt 5.3) und die notwendigen Schritte zur Verwaltung der Hardware in einem Cloud-System (siehe Abschnitt 5.2). Somit ist eine Abstraktion der FPGAs von Ressourcen, Bereich und Schnittstellen möglich. Über die RC2F-API wird des Weiteren eine einfache Interaktion zwischen den Host-VMs und den vFPGAs etabliert.

Weitere wichtige Erkenntnisse und Anforderungen zur Verbesserung zukünftiger FPGA-Architekturen für den speziellen Einsatz der virtualisierten rekonfigurierbaren Hardware in Cloud-Architekturen werden im Folgenden aufgezeigt:

- Statische RC2F-Infrastruktur, Bereiche und Taktregionen,
- Homogenität und Migration und
- Sicherheit.

6.4.1.1 Statische RC2F-Infrastruktur, Bereiche und Taktregionen

Als eine Schwierigkeit hat sich die Einteilung des physischen FPGAs in statische und dynamische Regionen für die Rekonfiguration herausgestellt. Im aktuellen RC2F-Prototypen nimmt der statische Bereich 31,07 % der FPGA-Ressourcen ein (siehe Abbildung 6.7), lastet davon aber lediglich 42,30 % der Slice LUTs aus (siehe Abbildung 6.8). Eine optimalere Aufteilung der Regionen ist aufgrund der Platzierung der statischen Komponenten auf dem FPGA und dessen Architektur derzeit nicht möglich. Eine spezielle FPGA-Architektur mit angepassten Bereichen und der statischen RC2F-Infrastruktur in Form eines SoC-FPGAs, wie in Abbildung 6.26 skizziert, für den Cloud-Einsatz ein notwendiger Schritt.

6.4.1.2 Homogenität und Migration

Ein bedeutender Aspekt zur Abstraktion vom physischen Ort von vFPGA-Image/-Instanz auf dem physischen FPGA ist innerhalb des RC2F-Prototypen durch die Etablierung homogener Regionen und ein zusätzliches Routing der vFPGA-Designs realisiert worden. Derartige homogene vFPGAs und ein angepasster Entwurfsablauf ermöglichen eine komplett Migration einer vFPGA-Instanz innerhalb und zwischen physischen FPGAs. Es wird deutlich, dass eine Kontextmigration möglich, aber mit zum Teil

hohen Kosten hinsichtlich der nutzbaren Ressourcen auf dem FPGA verbunden ist. Durch die homogenen vFPGAs sind 6,59 % der Fläche des physischen FPGAs nicht nutzbar und durch statische Leitungen, welche im RC2F-Prototypen nicht vollständig vermeidbar sind, ist zudem ein separates Routing erforderlich. Die dabei entstehenden unterschiedlichen partiellen Bitstreams, welche notwendig sind, um die Flexibilität bezüglich der konkreten Position der vFPGAs zu gewährleisten, führen zu einem deutlichen Zuwachs der Datenmengen in Form von vRAI-Paketen (siehe Abschnitt 6.13).

Die vFPGAs selbst sollten dabei in einer gemeinsamen FPGA-Struktur ohne räumliche Trennung voneinander realisiert werden, um weiterhin wie beschrieben unterschiedliche Größen der vFPGAs zu ermöglichen, wie in Abbildung 6.26 am Beispiel einer zukünftigen konzeptionellen SoC-FPGA Architektur gezeigt. Da die Regionen in diesem Fall vollständig homogen sind und keine statischen Leitungen benötigt werden, ist eine Verschiebung der vFPGA-Designs deutlich einfacher als bisher möglich und der in Abschnitt 6.1.4.7 skizzierte modifizierte Entwurfsablauf entfällt. Ebenso sollte die Möglichkeit bestehen die internen Pipelineregister innerhalb der DSPs auszulesen, um einen vollständigen Kontext ohne weitere Randbedingungen zwischen vFPGAs migrieren zu können.

6.4.1.3 Sicherheit

Die Gewährleistung der Sicherheit ist bei der aktuellen Implementierung aufgrund der statischen Leitungen und der daher notwendigen Validierung der Bitstreams eine weitere Schwierigkeit, kann aber durch spezielle FPGA-Architekturen, beziehungsweise ein entsprechendes Layout des FPGA-Boards für einen produktiven Cloud-Einsatz vermieden werden. Entsprechend sind dabei auch I/O-Ports ausschließlich innerhalb des statischen Bereiches erforderlich, um einerseits Sicherheit zu garantieren und andererseits statische Leitungen innerhalb der vFPGA-Slots zu vermeiden. Ebenso sind separate Taktregionen für die vFPGA-Slots erforderlich, um Konflikte zwischen den unterschiedlichen vFPGAs zu vermeiden. Da die Virtualisierung von FPGAs für den Cloud-Einsatz einen zukunftsweisender Ansatz auch für andere Anwendungsfälle darstellt. Dabei ist möglichst die gesamte RC2F-Infrastruktur, wie sie in Form des in Abschnitt 6.1.4 vorgestellten RC2F-Prototypen realisiert wurde, als fester Bestandteil mit Kommunikations- und Speichercontrollern auf dem Chip mit festen Frontends zu vertikal angeordneten Taktregionen für die vFPGAs umzusetzen, wie in Abbildung 6.26 aufgezeigt und bereits in [KGS17] motiviert.

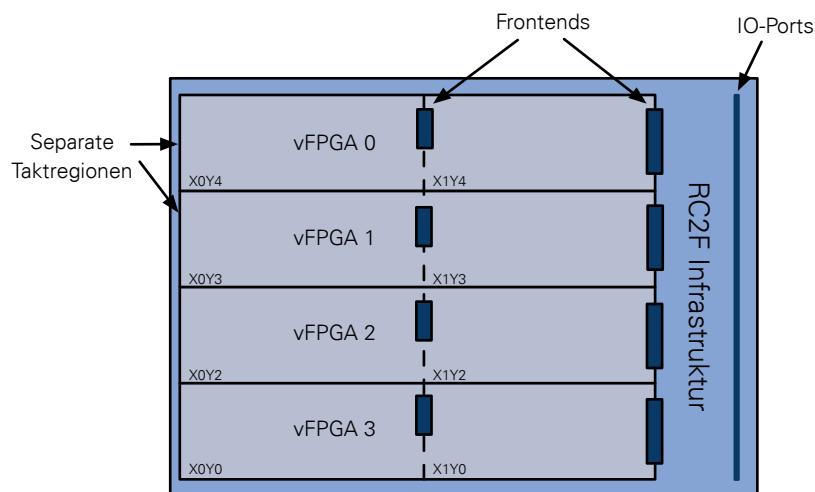


Abbildung 6.26: Aufbau eines konzeptionellen SoC-FPGAs zur Bereitstellung von beispielsweise vier vFPGAs mit eigenen, abgeschlossenen Taktregionen.

6.4.2 Erkenntnisse der RC3E-Simulation für einen Einsatz in einer Cloud

Die in Abschnitt 6.3.1 aufgezeigten prognostischen Ergebnisse des RC3E-Simulators basieren auf den zuvor in Abschnitt 6.2 eingeführten Demonstratoren. Da diese die Vorteile der Virtualisierung der FPGAs zeigen sollen und diese entsprechend nicht vollständig auslasten. Basierend auf dieser Annahme konnte entsprechend mittels der RC3E-Simulation gezeigt werden, dass durch die RC2F-Virtualisierung die Auslastung der FPGA-Ressourcen deutlich erhöht werden konnte. Durch diese Erhöhung der Auslastung der FPGAs konnte des Weiteren der Energieverbrauch in einem realen Lastszenario (siehe Szenario (II) in Tabelle 6.7) konnten des Weiteren der Energiebedarf um 60,25 % gesenkt werden. Gerade bei dem Einsatz von größeren FPGAs in einer Cloud zur Hintergrundbeschleunigung von Anwendungen (BAaaS), wie nachfolgend in Abschnitt 6.4.3 gezeigt, bekommt die Optimierung der Auslastung der Hardware einen deutlich höheren Stellenwert.

Eine zusätzliche Migration von vFPGAs bringt nur geringe Einsparungen von weiteren 2,85 %. Aufgrund einer Verschlechterung des SLAs in Extremwertsituationen wie Szenario (I) ist der Einsatz der Migration unter Umständen nicht zur Optimierung der Auslastung geeignet. Der wesentliche Vorteil der Migration ist hingegen eine örtliche Verlagerung der Arbeitspakete auf andere physische Rechenknoten zur Wartung der Hardware.

6.4.3 Weitere Entwicklungen und Ausblick – Xilinx UltraScale+ FPGA

Der in dieser Arbeit verwendete Virtex-7 XC7VX485T wird als Hardwarebeschleuniger für zahlreiche Anwendungen eingesetzt und ebenso für die prototypische RC2F-Virtualisierung. Wie bereits zuvor in Abschnitt 6.1.4.3 gezeigt ist die Aufteilung der FPGA-Ressourcen auf die unterschiedlichen Bereiche auf dem physischen FPGA für eine Virtualisierung problematisch.

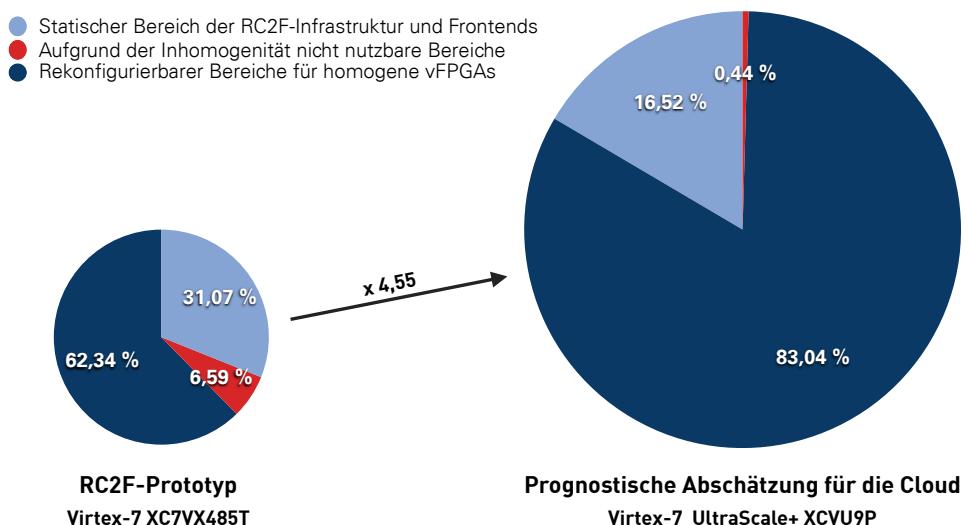


Abbildung 6.27: Anteile der Bereiche auf dem FPGA für eine Bereitstellung homogener vFPGAs innerhalb des RC2F-Prototypen und einer prognostischen RC2F-Virtualisierung auf einem Xilinx UltraScale+ FPGA.

Die von Amazon in den EC2-F1 Instanzen eingesetzten Hardwarebeschleuniger sind Virtex-7 UltraScale+ FPGAs [Ama17a] auf einem VCU1525 Acceleration Development Kit mit einem XCVU9P [Xil17d]. Eine prognostische Übertragung der RC2F-Virtualisierung auf eine UltraScale+ (XCVU9P) FPGA liefert die in Abbildung 6.27 aufgezeigte Aufteilung der FPGA-Ressourcen auf die unterschiedlichen Bereiche (siehe

auch Abbildung C.2 und Tabelle C.2). Der für die vFPGAs nutzbare Bereich liegt demnach bei 83,04 % und skaliert nicht linear mit der Größe des FPGAs, welcher im Vergleich zum Virtex-7 XC7VX485T 4,55 mal so groß ist. Durch die homogene Struktur der UltraScale+ FPGAs ist der nicht nutzbare Bereich auf 0,44 % gesunken aber dennoch vorhanden, sodass eine Homogenisierung weiterhin erforderlich ist. Die entsprechenden Forderungen an die Struktur eines SoC-FPGAs aus Abschnitt 6.4.1 werden daher nicht erfüllt, auch wenn statische RC2F-Infrastruktur, Bereiche und Taktregionen deutlich besser umgesetzt werden können.

6.4.4 Vergleich mit anderen Arbeiten

Der Vergleich zu anderen Arbeiten aus dem Gebiet sowohl der Integration von FPGAs in eine Cloud, wie in Abschnitt 2.4.4.7 dargestellt, als auch der Virtualisierung in Abschnitt 2.4.1.6 wird an dieser Stelle auf Basis der umgesetzten Konzepte und aufgezeigten Möglichkeiten vorgenommen. Eine zusätzliche Klassifizierung relevanter Arbeiten ist im Anhang in Abschnitt B zu finden. Direkte Leistungsvergleiche wurden, wenn möglich, in den Kapiteln 4, 5 und 6 angebracht. Eine Auswahl von Arbeiten mit dem Schwerpunkt der Virtualisierung von FPGAs in der Cloud sind überblicksweise in Tabelle 6.10 dargestellt. Die in Abschnitt 2.4.1.6 vorgestellten Arbeiten zur Virtualisierung ohne Cloud-Bezug sind in der Tabelle nicht aufgeführt, da deren Fokus sich in der Regel nur auf einen Aspekt der Virtualisierung richtet und sie daher für den Vergleich an dieser Stelle ungeeignet sind.

Tabelle 6.10: Qualitativer Vergleich mit einer Auswahl von Arbeiten mit Fokus auf dem Einsatz von FPGAs in einer Cloud und entsprechendem Virtualisierungskonzept. Der Grad der Virtualisierung ist die Summe der Bewertung der nachfolgenden Kriterien, wobei realisiert (✓) dem Wert 1, eingeschränkt (!) dem Wert 0,5 und nicht realisiert (✗) dem Wert 0 entspricht.

Arbeit	Cloud-	Grad der	Host	Migration	Homogene	Mehrere	Skalierbarkeit	
	Integration	Virtualisierung	VM	(vFPGAs)	(vFPGAs)	Nutzer ^a	FPGA	Cluster
Fahmy et al. [Asi+16; FVS15]	!	1,0	✗	✗	✗	✓	✗	✗
Byma et al. [Bym+14]	✓	1,5	✗	!	✗	✓	✗	✗
Kachris et al. [Kac+16b]	✓	1,0	✓	✗	✗	✗	✗	✗
Gazzano et al. [Don+15]	✓	2,0	✗	✗	✗	✓	✗	✓
Iordache et al. [Ior+16]	!	2,5	✓	!	✗	✗	✗	✓
Chen et al. [Che+14]	✓	3,5	✓	!	✗	✓	✓	✗
RC3E & RC2F	✓	5,5	✓	✓	✓	✓	✓	!

✓: realisiert (1) !: eingeschränkt nutzbar (0,5) ✗: nicht realisiert (0) ^a: nebenläufig auf demselben FPGA

Um eine Aussage über die Qualität, beziehungsweise die Facetten der Virtualisierung treffen zu können, werden die unterschiedlichen Kennwerte (abgesehen von der Cloud-Integration) zum Parameter *Grad der Virtualisierung* zusammengefasst, welcher die Summe der nachfolgenden Werte bildet. Die Spalte *Cloud-Integration* zeigt lediglich, wie stark die Rolle der Cloud-Integration in der jeweiligen Arbeit ist. Viele der Arbeiten erfüllen mehrere Kriterien einer Virtualisierung, wie sie in der vorliegenden Arbeit umgesetzt ist. Da die vorliegende Arbeit in großem Umfang die Methoden einer klassischen virtuellen Maschine auf FPGAs überträgt und eine dynamische Verteilung der Arbeitspakete, sowie eine Abstraktion von der konkreten Position auf dem physischen FPGA umsetzt, wird ein hoher Grad der Virtualisierung erreicht. Die Kriterien der Virtualisierung nach Abschnitt 2.2 sind damit weitgehend erfüllt, bis auf eine vollkommenen örtliche Unabhängigkeit, aufgrund der fehlenden Homogenität moderner FPGAs. Ein Ansatz die Homogenität auf aktuellen FPGA-Architekturen dennoch zu ermöglichen und die

6.4 Abschließende Bewertung und weiterführende Betrachtungen

damit Verbundenen Einbußen in FPGA-Ressourcen und Speicherplatz wurden in Abschnitt 6.1.4.2 und Abschnitt 6.1.4.8 aufgezeigt.

Die starke Integration und Kopplung zwischen FPGA und VMs, sowie die Abstraktion von der Größe und der konkreten Position des vFPGAs auf dem physischen FPGA, sowie die eingeführte Homogenität zum Migrieren des Kontextes ist in der Literatur bislang nicht untersucht. Oftmals werden nur Teilaspekte betrachtet, nicht aber das Zusammenspiel von FPGA-Virtualisierung, Host-Hypervisor und Cloud-Verwaltung. Beispielsweise kann eine Virtualisierung eines FPGAs nicht durchgeführt werden, ohne die Kopplung mit einer VM auf dem Host zu berücksichtigen, sowie eine detaillierte Betrachtung der verschiedenen Nutzer und deren Interaktion untereinander und mit den Ressourcen.

Die Arbeiten von Fahmy et al. [Asi+16; FVS15] und Byma et al. [Bym+14] verfolgen im Ansatz ebenfalls die Bereitstellung einer ähnlichen Kapselung der Nutzer in vFPGAs, die aber in Größe und Ressourcen nicht flexibel sind. Die Arbeiten von Kachris et al. [Kac+16b] und Gazzano et al. [Don+15] legen mehr Wert auf die Untersuchung einer Integration der FPGAs in eine Cloud. Einen vielversprechenden Ansatz stellt die Arbeit von Iordache et al. [Ior+16] dar, da das Konzept des Zusammenschlusses von physischen FPGAs dem Cloud-Konzept der Skalierung sehr entgegenkommt. Die Arbeit von Chen et al. [Che+14] verfolgt eine tiefgreifendere Integration von FPGAs in ein Cloud-System, wobei die Flexibilität der virtuellen FPGAs deutlich geringer ist, als in der RC2F-Virtualisierung der hier vorgelegten Arbeit. Die Virtualisierung in Chen et al. [Che+14] ist reduziert auf für den Nutzer vordefinierte Bereiche, was eine dynamische und flexible Lastverteilung deutlich erschwert. Bei der RC3E-Simulation zur Evaluation des Einsatzes von FPGAs in der Cloud wird sowohl die Virtualisierung der FPGAs als auch die Migration der FPGA-Ressourcen im Gegensatz zu anderen Arbeiten berücksichtigt. Die Lastverteilung in der Arbeit von Proaño [PCC16] ist dagegen nur auf volle und vor allem einzelne FPGAs beschränkt und es wird primär ein zweistufiges Schedulingverfahren ähnlich wie dem innerhalb des RC3E-Simulators untersucht.

7 Zusammenfassung und Ausblick

Die vorliegende Arbeit hat die Einsatzmöglichkeiten von rekonfigurierbarer Hardware im Cloud-Kontext untersucht. Die Notwendigkeit einer Virtualisierung der Ressource FPGA für einen effizienten Einsatz in der Cloud ist eine der Hauptkenntnisse der Arbeit. Es wurde ein Ausblick auf mögliche Einsatzgebiete gegeben, um die Virtualisierung der FPGAs zu motivieren. Im Folgenden werden zunächst die Ergebnisse zusammengefasst und in Hinblick auf die in Kapitel 2.5.2 formulierten Ziele bewertet. Das Kapitel schließt mit einem Ausblick auf weitere offene und sich daraus ergebende Forschungsfragen, sowie auf die Anforderungen an zukünftige FPGA-Architekturen für einen Einsatz in der Cloud.

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde zunächst in Kapitel 2 gezeigt, wie FPGAs im Kontext von Cloud-Architekturen eingesetzt werden können. Die Integration der FPGAs in handhabbarer und effizienter Form auf der Anwendungsebene wird untersucht, um die Ressource FPGA den Anbietern von Diensten bereitzustellen. Im Vordergrund stehen Nutzbarkeit, Effizienz, Leistungssteigerung, Sicherheit, Ergonomie und eine Erhöhung der Auslastung der rekonfigurierbaren Hardware, welche sich an die Bedürfnisse der Nutzer anpasst. Ebenso spielt die Einbettung der FPGAs in eine Verwaltungsstruktur mit unterschiedlichen Servicemodellen eine besondere Rolle. Die Untersuchungen zeigten weiterhin mögliche Entwurfsräume mit vielfältigen Lösungsvarianten auf, wie eine Integration der FPGAs auf der Anwendungsebene möglich ist. Ein Teilgebiet aktueller Forschungen, welches seit circa 2014 stetig wächst, widmet sich dabei dem Aspekt mehrere Nutzer mit ihren Hardwaredesigns auf dem selben physischen FPGA gleichzeitig interagieren zu lassen, um diesen möglichst effizient auszulasten.

Für einen effizienten Einsatz von FPGAs in einer Cloud hat sich auch bei der Analyse anderer Arbeiten der eigene Ansatz der Virtualisierung als eine zentrale Zielstellung herausgebildet, da eine Cloud-Integration zwangsläufig virtualisierte Ressourcen benötigt. Die Kopplung zwischen virtualisierten FPGAs und Virtuelle Maschinen auf dem Host-System wird dabei ebenso berücksichtigt, wie die Integration in die Verwaltungsebene der Cloud. In Kapitel 3 wurden aufgrund dessen zunächst drei Dienste entwickelt, welche eine Abstraktion von der Hardware auf unterschiedlichen Ebenen ermöglichen. Somit wurde eine Basis für weitere Dienste von der Bereitstellung kompletter physischer Hardware bis zu virtualisierten FPGAs adaptiver Größe geschaffen. Die möglichen, für die Nutzer relevanten Systemtopologien wie ein Zugang zu einer VM über einen (sicheren) FPGA oder der Einsatz als einfacher Hardwarebeschleuniger werden aufgezeigt. Hierbei spielt die Sicherheit der Nutzerdaten eine entscheidende Rolle, da sich durch die Virtualisierung unterschiedliche virtualisierten FPGAs auf demselben physischen FPGA befinden. Entsprechend wurden die Anforderungen an eine FPGA-Virtualisierung aufgezeigt und abgegrenzt. Des Weiteren wurden ebenso Anforderungen an sowohl eine Verwaltungsstruktur für die FPGA-Ressourcen, als auch an deren Kopplung mit dazugehörigen VMs auf dem Host-System aufgezeigt.

Der gewählte Ansatz für die Virtualisierung der FPGAs, wie er in Kapitel 4 vorgestellt wurde, besteht im Gegensatz zur Literatur darin, die rekonfigurierbare Hardware in der Cloud zu nutzen wie andere vir-

7 Zusammenfassung und Ausblick

tualisierte Komponenten (Virtuelle Maschinen, Betriebssysteme etc.). Da eine die Abstraktion vom physischen FPGA erfolgen muss, ohne diesen durch eine Zwischenschicht, welche zu Leistungseinbußen führen würde, zu ersetzen, erfolgt die FPGA-Virtualisierung analog zu einer native Typ 1 System-/ Bare-Metal-Virtualisierung bei gleicher Architektur (ISA). Der Zugriff auf externe Geräte wird dabei mittels Paravirtualisierung innerhalb des Hypervisors (VMM) ermöglicht. Durch die Bereitstellung homogener Bereiche wird eine einheitliche Plattform geschaffen, um die Migration zwischen Regionen auf demselben FPGA, aber auch zwischen unterschiedlichen FPGAs zu gewährleisten. Das entwickelte Reconfigurable Common Computing Frame(work) (RC2F) ermöglicht somit eine Etablierung von virtuellen FPGAs (vFPGAs), welche ähnlich wie klassische VMs einsetzbar sind.

Um eine Bereitstellung der virtualisierten FPGAs in eine Cloud-Architektur zu ermöglichen wurden in Kapitel 5 Möglichkeiten und erforderliche Systemkomponenten aufgezeigt. Dabei nehmen sowohl der Hypervisor auf dem Host-System, als auch dessen Zusammenspiel mit den Hypervisor auf dem FPGA entscheidende Rollen einnehmen. Ebenso wurden eine Lastverteilung und die Beschreibung der virtuellen FPGAs in der Cloud erarbeitet. Das somit entstandene Reconfigurable Common Cloud Computing Environment (RC3E) bietet Nutzern die Möglichkeit, unterschiedlichste virtuelle Konfigurationen für das RC2F zu beschreiben und zu verwalten.

In Kapitel 6 wurden prototypische Implementierungen von sowohl RC2F als auch RC3E vorgestellt, mittels derer nachgewiesen wird, dass die Bereitstellung von vFPGAs dynamischer Größe und losgelöst vom physischen Ort möglich ist. Innerhalb des entwickelten RC2F-Prototypen konnte exemplarisch gezeigt werden, wie vFPGA-Designs flexibel angeordnet werden können und wie eine Migration von vFPGAs realisiert werden kann. Die Etablierung homogener Bereiche – welche für eine Migration der vFPGAs notwendig sind – benötigen 6,59 % der FPGA-Ressourcen des genutzten FPGAs. Die statische RC2F-Infrastruktur erfordert 31,07 % der Fläche, ist aber lediglich zu 42,30 % (Slice LUTs) ausgelastet. Für die eigentlichen vFPGAs stehen somit insgesamt 62,34 % der FPGA-Ressourcen zur Verfügung. Der praktikable Einsatz der prototypischen RC2F-Virtualisierung wurde anhand von Demonstratoren gezeigt und im Rahmen mehrerer Szenarios untersucht.

Eine Abschätzung zur Übertragung des RC2F auf neuartige FPGAs hat gezeigt, dass bis zu 83,04 % der FPGA-Ressourcen innerhalb von vFPGAs den Nutzern zur Verfügung gestellt werden können. Somit kann eine Steigerung zu den 62,34 % der nutzbaren FPGA-Ressourcen innerhalb des RC2F-Prototypen erreicht werden. Die Größe der Bereiche skalieren somit nicht linear mit der Größe des physischen FPGAs und mit größeren FPGAs können somit deutlich bessere Werte erzielt werden.

Mithilfe der RC3E-Simulation konnte ebenfalls verdeutlicht werden, dass für spezielle Szenarien durch die RC2F-Virtualisierung der Energiebedarf des aufgezeigten Cloud-Systems um 60,25 % reduziert werden kann. Eine zusätzliche Migration der vFPGAs ergab zusätzliche 2,85 % Energieeinsparungen und ist des Weiteren zum Beispiel für die Wartung der Hardware im Rechenzentrums unerlässlich.

Ein Vergleich mit anderen Arbeiten zu den Teilbereichen der Virtualisierung der FPGAs sowie deren Integration in eine Cloud wurde durchgeführt. Ein wichtiges Kriterium stellt neben dem Ressourcenbedarf die Flexibilität und insbesondere die dynamische Änderung der vFPGAs in ihrer Größe und Position innerhalb der physischen Systeme dar. Die Analyse hat gezeigt, dass die vorliegende Arbeit im Vergleich zu anderen Arbeiten alle wesentlichen Teileaspekte einer Virtualisierung abdeckt und des Weiteren auch die Integration in moderne Cloud-Architekturen ermöglicht. Damit sind die eingangs in Abschnitt 2.5.2 formulierten Ziele weitestgehend erreicht. Die starke Integration und Kopplung zwischen virtualisierten FPGAs und VMs, die Abstraktion von der Größe und der konkreten Position der vFPGAs auf den physischen FPGAs ebenso wie die eingeführte Homogenität zum Migrieren des Kontextes sind in der Literatur bislang nicht in diesem Umfang untersucht worden.

7.2 Ausblick

Aus den im vorherigen Abschnitt dargestellten Ergebnissen resultieren Fragestellungen, welche sich für die Bearbeitung im Rahmen zukünftiger Arbeiten anbieten. Insbesondere kann diese Arbeit als eine Grundlage für weitere Forschungen auf Basis der beiden Systeme *RC2F* und *RC3E* dienen:

Übertragung des RC2F auf einen UltraScale+ FPGA: Die Übertragung des RC2F auf einen UltraScale+ FPGA innerhalb eines Cloud-Systems ist mit den entsprechenden homogenen Bereichen und der Migration zu evaluieren.

Weiterentwicklung des RC3E-Simulators: eine detaillierte Abbildung des physischen Systems ist erforderlich, um weitere Untersuchungen für unterschiedlichste Systemkonfigurationen und Lastszenarien durchführen zu können.

Migration zur Defragmentierung und Wartbarkeit von FPGAs in der Cloud: Die Migration zur Erhöhung der Wartbarkeit der Hardwarekomponenten im Rechenzentrum und deren Beitrag zur Optimierung der Auslastung der FPGAs durch Defragmentierung sollte ebenso betrachtet werden.

Detailliertes Scheduling von FPGA-Ressourcen unter Nutzung der Migration: Im Kontext des RC3E-Simulators können weiterführend im Detail untersucht werden, für welche Arten von Arbeitspaketen und in welchem Kontext eine Migration sinnvoll zur Maximierung der Auslastung und zur Einsparung von Ressourcen und Energie eingesetzt werden kann.

Weitere Anwendungsszenarien und konkrete Anwendungen: Neben den gezeigten beispielhaften Anwendungen können alternative Einsatzgebiete (Datenbank, Big Data etc.) innerhalb der vFPGAs realisiert werden, um die Flexibilität der entwickelten RC3E-Architektur nachzuweisen.

Verknüpfung von RC3E mit OpenStack oder OpenNebula: Für eine Weiterentwicklung der entstandenen Ressourcenverwaltung ist die Integration, beziehungsweise Verknüpfung mit einem etablierten Cloud-Management System wie OpenStack oder OpenNebula zu untersuchen.

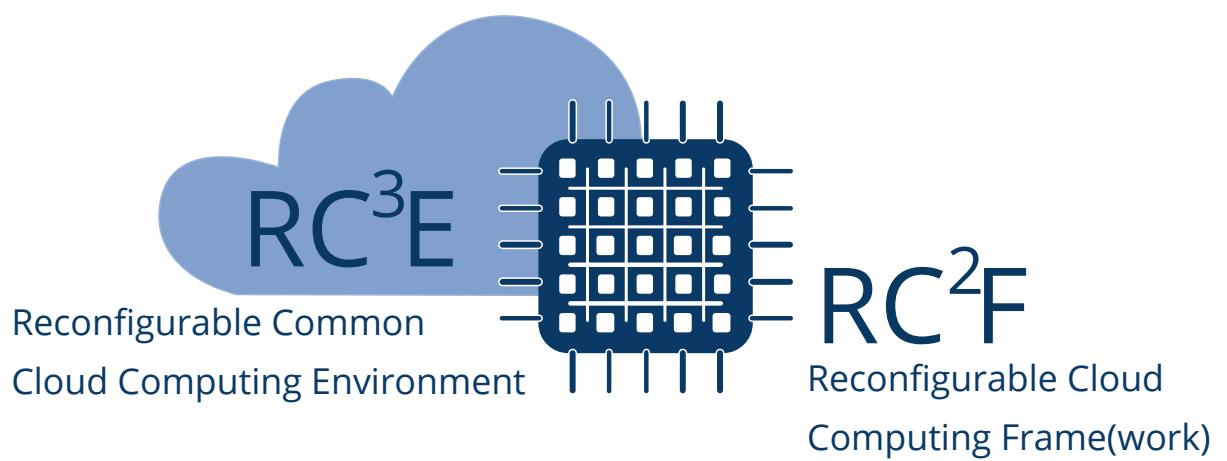
7.3 Aspekte für zukünftige FPGA-Architekturen

Die erarbeiteten Ansätze zur Virtualisierung von FPGA-Ressourcen innerhalb des RC2F sollten des Weiteren in kommerzielle virtualisierte FPGA-Architekturen eingebettet werden. Dazu sind unterschiedliche Aspekte und strukturelle Änderungen der FPGA-Architekturen zur Erleichterung einer Bereitstellung von vFPGA-Ressourcen erforderlich:

SoC-FPGA Architektur für die statische Infrastruktur: Strukturierung eines statischen Bereiches speziell für die Verwaltung (RC2F-Infrastruktur mit FPGA-Hypervisor) mit sämtlichen Schnittstellen zur Außenwelt, um statische Leitungen der Infrastruktur vollständig vom rekonfigurierbaren Bereich der vFPGAs zu separieren.

Strukturelle Änderungen innerhalb der FPGA-Architektur: Etablierung homogener und abgeschlossener Bereiche mit eigenen Taktregionen auf den FPGAs, welche keine statische Leitungen innerhalb der vFPGAs enthalten. Ebenso ist ein effizientes Auslesen sämtlicher Register- und Speicherinhalte (ebenso der Registerinhalte innerhalb der DSPs) für eine vollständige Migration erforderlich.

Umsetzung des Sicherheitskonzeptes: Sicherheitskonzept auf Basis einer Trusted Authority zur Bereitstellung von verifizierbarer Infrastruktur mit FPGA-Hypervisor, in welche sich die dynamisch rekonfigurierbaren vFPGAs integrieren lassen.



Literaturverzeichnis

- [AD11] R. Ajami und A. Dinh. „Embedded Network Firewall on FPGA“. In: *2011 Eighth Int'l Conf. on Information Technology: New Generations*. Apr. 2011, S. 1041–1043. DOI: 10.1109/ITNG.2011.178.
- [AF11] Aderemi A Atayero und Oluwaseyi Feyisetan. „Security issues in cloud computing: The potentials of homomorphic encryption“. In: *Journal of Emerging Trends in Computing and Information Sciences* 2.10 (2011), S. 546–552. ISSN: 2079-8407.
- [Agn+14] Andreas Agne u. a. „ReconOS: An Operating System Approach for Reconfigurable Computing“. In: *IEEE Micro* 34.1 (2014), S. 60–71. ISSN: 0272-1732. DOI: 10.1109/MM.2013.110.
- [Ahm+02] Ardsher Ahmed, Pat Conway, Bill Hughes und Fred Weber. „AMD opteron shared memory mp systems“. In: *Proceedings of the 14th HotChips Symp.* 2002.
- [Al 13] Waleed Al Shehri. „Cloud Database Database as a Service“. In: *Int'l Journal of Database Management Systems* 5.2 (2013), S. 1.
- [Alt+90] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers und David J Lipman. „Basic local alignment search tool“. In: *Journal of molecular biology* 215.3 (1990), S. 403–410.
- [Alt13] Altera Corporation. *Implementing FPGA Design with the OpenCL Standard*. White Paper: WP-01173-3.0. 2013.
- [Ama16a] Amazon Inc. *Amazon EC2 Instance Types*. Website. Online unter: <https://aws.amazon.com/ec2/instance-types/>; abgerufen am 19. Oktober. 2016.
- [Ama16b] Amazon Inc. *Amazon Elastic Compute Cloud - Elastic Load Balancing Documentation*. Website. Online unter: <https://aws.amazon.com/documentation/elastic-load-balancing/>; abgerufen am 25. Oktober. 2016.
- [Ama17a] Amazon Inc. *Amazon EC2 F1 Instances – Run Custom FPGAs in the AWS Cloud*. Website. Online unter: <https://aws.amazon.com/ec2/instance-types/f1/>; abgerufen am 25. Januar. 2017.
- [Ama17b] Amazon Inc. *Amazon Elastic Compute Cloud (Amazon EC2)*. Website. Online unter: <https://aws.amazon.com/de/ec2/>; abgerufen am 6. Oktober. 2017.
- [Ama17c] Amazon Inc. *Amazon Elastic Compute Cloud (Amazon S3)*. Website. Online unter: <https://aws.amazon.com/de/s3/>; abgerufen am 6. Oktober. 2017.
- [Amd67] Gene M Amdahl. „Validity of the single processor approach to achieving large scale computing capabilities“. In: *Proc. of the April 18-20, 1967, spring joint computer Conf.* ACM. 1967, S. 483–485. DOI: 10.1145/1465482.1465560.
- [Apa16a] Apache Cassandra. *Manage massive amounts of data, fast, without losing sleep*. Online unter: <http://cassandra.apache.org>; abgerufen am 13. Oktober. 2016.
- [Apa16b] Apache Mahout. *What is Apache Mahout?* Online unter: <http://mahout.apache.org>; abgerufen am 13. Oktober. 2016.
- [Apa16c] Apache Spark. *Apache Spark – GraphX*. Online unter: <http://spark.apache.org/graphx/>; abgerufen am 13. Oktober. 2016.
- [Apa16d] Apache Spark. *Machine Learning Library (MLlib) Guide*. Online unter: <http://spark.apache.org/docs/latest/ml-guide.html>; abgerufen am 13. Oktober. 2016.
- [Ara+13] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy und Rathnam Venkatesan. „A secure coprocessor for database applications“. In: *2013 23rd Int'l Conf. on Field programmable Logic and Applications*. IEEE. 2013, S. 1–8. DOI: 10.1109/FPL.2013.6645524.
- [Arm+10] Michael Armbrust u. a. „A view of cloud computing“. In: *Communications of the ACM* 53.4 (2010), S. 50–58. DOI: 10.1145/1721654.1721672.
- [Asi+16] Mikhail Asiatici, Nithin George, Kizheppatt Vipin, Suhaib A Fahmy und Paolo lenne. „Designing a virtual runtime for FPGA accelerators in the cloud“. In: *Field Programmable Logic*

- and Applications (FPL), 2016 26th Int'l Conf. on.* EPFL. 2016, S. 1–2. DOI: 10.1109/FPL.2016.7577389.
- [BAB12] Anton Beloglazov, Jemal Abawajy und Rajkumar Buyya. „Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing“. In: *Future generation computer systems* 28.5 (2012), S. 755–768. DOI: 10.1016/j.future.2011.04.017.
- [Bac+14] Rico Backasch, Gerald Hempel, Stefan Werner, Sven Groppe und Thilo Pionteck. „Identifying homogenous reconfigurable regions in heterogeneous FPGAs for module relocation“. In: *ReConfigurable Computing and FPGAs (ReConFig), Int'l Conf. on.* IEEE. 2014, S. 1–6. DOI: 10.1109/ReConFig.2014.7032533.
- [Bak10] J.D. Bakos. „High-performance heterogeneous computing with the convey HC-1“. In: *Computing in Science & Engineering* 12.6 (2010). DOI: 10.1109/MCSE.2010.135.
- [Bal+11] Jayant Baliga, Robert WA Ayre, Kerry Hinton und Rodney S Tucker. „Green cloud computing: Balancing energy in processing, storage, and transport“. In: *Proc. of the IEEE* 99.1 (2011), S. 149–167. DOI: 10.1109/JPROC.2010.2060451.
- [Bal98] H Balzert. *Lehrbuch der Software-Technik. Bd. 2. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum-Verlag. Techn. Ber. ISBN 3-8274-0065-1, 1998.
- [Ban+11] Prith Banerjee u. a. „Everything as a service: Powering the new information economy“. In: *Computer* 44.3 (2011), S. 36–43.
- [Bar+03] Paul Barham u. a. „Xen and the art of virtualization.“ In: *SOSP* 37.5 (2003), S. 164–177.
- [Bar+08] Kevin J Barker u. a. „Entering the petaflop era: the architecture and performance of Roadrunner“. In: *Proceedings of the 2008 ACM/IEEE Conf. on Supercomputing.* IEEE Press. 2008, S. 1. ISBN: 978-1-4244-2835-9.
- [Bax07] R. Baxter et al. „Maxwell-a 64 FPGA supercomputer“. In: *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conf. on.* IEEE. 2007. DOI: 10.1109/AHS.2007.71.
- [BBA10] Rajkumar Buyya, Anton Beloglazov und Jemal Abawajy. „Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges“. In: *arXiv preprint arXiv:1006.0308* (2010).
- [BC11] Anju Bala und Inderveer Chana. „A survey of various workflow scheduling algorithms in cloud environment“. In: *2nd National Conf. on Information and Communication Technology (NCICT)*. sn. 2011, S. 26–30.
- [BDB00] Vasanth Bala, Evelyn Duesterwald und Sanjeev Banerjia. „Dynamo: a transparent dynamic optimization system“. In: *ACM SIGPLAN Notices* 35.5 (2000), S. 1–12. DOI: 10.1145/349299.349303.
- [Ber16] Berten DSP S.L. *GPU vs FPGA Performance Comparison*. Online unter: http://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf; abgerufen am 7. Dezember 2016. 2016.
- [BF16] Boudewijn de Bruin und Luciano Floridi. „The Ethics of Cloud Computing“. In: *Science and engineering ethics* (2016), S. 1–19.
- [Bha+09] Sheetal U Bhandari, Shaila Subbaraman, SS Pujari und Rashmi Mahajan. „Real time video processing on FPGA using on the fly partial reconfiguration“. In: *Signal Processing Systems, Int'l Conf. on.* IEEE. 2009, S. 244–247. DOI: 10.1109/ICSPS.2009.32.
- [BKT11] Christian Beckhoff, Dirk Koch und Jim Torresen. „The Xilinx Design Language (XDL): Tutorial and use cases“. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th Int'l Workshop on.* IEEE. 2011, S. 1–8. DOI: 10.1109/ReCoSoC.2011.5981545.
- [BKT12] Christian Beckhoff, Dirk Koch und Jim Torresen. „Go ahead: A partial reconfiguration framework“. In: *Field-Programmable Custom Computing Machines (FCCM), 20th Annual Int'l Symp. on.* IEEE. 2012, S. 37–44. DOI: 10.1109/FCCM.2012.17.
- [BKT14] Christian Beckhoff, Dirk Koch und Jim Torresen. „Portable module relocation and bitstream compression for Xilinx FPGAs“. In: *Field Programmable Logic and Applications (FPL), 2014 24th Int'l Conf. on.* IEEE. 2014, S. 1–8. DOI: 10.1109/FPL.2014.6927480.

- [BL12] Alexander Brant und Guy GF Lemieux. „Zuma: An open fpga overlay architecture“. In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual Int'l Symp. on*. IEEE. 2012, S. 93–96. DOI: 10.1109/FCCM.2012.25.
- [Blo+15] Michaela Blott, Ling Liu, Kimon Karras und Kees Vissers. „Scaling out to a single-node 80Gbps memcached server with 40terabytes of memory“. In: *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*. 2015.
- [Blu17] Bluespec, Inc. *Bluespec – Best-in-Class, General Purpose High-Level Synthesis (HLS) Tools*. Online unter: <http://www.bluespec.com/high-level-synthesis-tools.html> ; abgerufen am 26. Januar 2017. 2017.
- [BML03] Brandon Blodget, Scott McMillan und Patrick Lysaght. „A lightweight approach for embedded reconfiguration of FPGAs“. In: *Design, Automation and Test in Europe Conf. and Exhibition, 2003*. IEEE. 2003, S. 399–400. DOI: 10.1109/DAT.2003.1253642.
- [Bob07] Christophe Bobda. *Introduction to reconfigurable computing: architectures, algorithms, and applications*. Springer Science & Business Media, 2007.
- [Boh+11] Robert B Bohn, John Messina, Fang Liu, Jin Tong und Jian Mao. „NIST cloud computing reference architecture“. In: *2011 IEEE World Congress on Services*. IEEE. 2011, S. 594–596.
- [Brä+12] Michael Bräuninger, Justus Haucap, Katharina Stepping, Torben Stühmeier u. a. *Cloud Computing als Instrument für effiziente IT-Lösungen*. Techn. Ber. Hamburg Institute of Int'l Economics (HWI), 2012.
- [Bre10] Tony M Brewer. „Instruction set innovations for the Convey HC-1 computer“. In: *IEEE micro* 30.2 (2010). DOI: 10.1109/MM.2010.36.
- [Bri16] Encyclopædia Britannica. *Coprocessor*. Online unter: <http://www.britannica.com/EBchecked/topic/1366763/coprocessor> ; abgerufen am 19. Oktober. Mai 2016.
- [Bro+08] Richard Brown u. a. „Report to congress on server and data center energy efficiency: Public law 109-431“. In: *Lawrence Berkeley National Laboratory* (2008).
- [Bru+98] Peter Brucker u. a. „Scheduling a batching machine“. In: *Journal of scheduling* 1.1 (1998), S. 31–54.
- [Bru01] Peter Brucker. *Scheduling Algorithms*. 3rd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001. ISBN: 3540415106.
- [Bug+12] Edouard Bugnion, Scott Devine, Mendel Rosenblum, Jeremy Sugerman und Edward Y Wang. „Bringing virtualization to the x86 architecture with the original vmware workstation“. In: *ACM Transactions on Computer Systems (TOCS)* 30.4 (2012), S. 12. DOI: 10.1145/2382553.2382554.
- [Bym+14] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia und Paul 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, S. 109–116. DOI: 10.1109/FCCM.2014.42.
- [Cal+11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose und Rajkumar Buyya. „CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms“. In: *Software: Practice and Experience* 41.1 (2011), S. 23–50. DOI: 10.1002/spe.995.
- [Cam+16] M Blanca Caminero u. a. „Towards a green, QoS-enabled heterogeneous cloud infrastructure“. In: *Parallel and Distributed Processing Symp. Workshops, 2016 IEEE*. IEEE. 2016, S. 7–16. DOI: 10.1109/IPDPSW.2016.12.
- [Can+13] Andrew Canis u. a. „LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems“. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.2 (2013), S. 24. DOI: 10.1145/2514740.
- [Can17] Canonical Ltd. *Linux Containers*. Online unter: <https://linuxcontainers.org> ; abgerufen am 9. März 2017. 2017.

Literaturverzeichnis

- [Cao+14] Xiaolin Cao, Ciara Moore, Máire O'Neill, Neil Hanley und Elizabeth O'Sullivan. „High-speed fully homomorphic encryption over the integers“. In: *Int'l Conf. on Financial Cryptography and Data Security*. Springer. 2014, S. 169–180. DOI: 10.1007/978-3-662-44774-1_14.
- [Cau+16] A. M. Caulfield u. a. „A cloud-scale acceleration architecture“. In: *2016 49th Annual IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*. 2016, S. 1–13. DOI: 10.1109/MICRO.2016.7783710.
- [CD94] Jason Cong und Yuzheng Ding. „FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.1 (1994), S. 1–12. DOI: 10.1109/43.273754.
- [CH02] Katherine Compton und Scott Hauck. „Reconfigurable computing: a survey of systems and software“. In: *ACM Computing Surveys (csUR)* 34.2 (2002), S. 171–210. DOI: 10.1145/508352.508353.
- [Che+08] Shuai Che, Jie Li, Jeremy W Sheaffer, Kevin Skadron und John Lach. „Accelerating compute-intensive applications with GPUs and FPGAs“. In: *Application Specific Processors, 2008. SASP 2008. Symp. on*. IEEE. 2008, S. 101–107. DOI: 10.1109/SASP.2008.4570793.
- [Che+14] Fei Chen u. a. „Enabling FPGAs in the cloud“. In: *Computing Frontiers, Proc. of the 11th ACM Conf. on*. ACM. 2014, S. 3. DOI: 10.1145/2597917.2597929.
- [Cho16] Paul Chow. „An Open Ecosystem for Software Programmers to Compute on FPGAs“. In: *FSP 2016; Third Int'l Workshop on FPGAs for Software Programmers; Proc. of*. VDE VERLAG GmbH. 2016, S. 1–11.
- [Cla+05] Christopher Clark u. a. „Live migration of virtual machines“. In: *Proceedings of the 2nd Conf. on Symp. on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, S. 273–286.
- [Cla+08] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner und J. Becker. „A multi-platform controller allowing for maximum Dynamic Partial Reconfiguration throughput“. In: *2008 Int'l Conf. on Field Programmable Logic and Applications*. Sep. 2008, S. 535–538. DOI: 10.1109/FPL.2008.4630002.
- [CM08] Philippe Coussy und Adam Morawiec. *High-level synthesis: from algorithm to digital circuit*. Springer Science & Business Media, 2008.
- [Cor+02] Srdjan Coric, Miriam Leeser, Eric Miller und Marc Trepanier. „Parallel-beam backprojection: an FPGA implementation optimized for medical imaging“. In: *Proceedings of the 2002 ACM/SIGDA tenth Int'l Symp. on Field-programmable gate arrays*. ACM. 2002, S. 217–226. DOI: 10.1145/503048.503080.
- [Cor96] Compaq Computer Corporation. *Internet Solutions Division Strategy for Cloud Computing*. Website. Online unter: http://www.technologyreview.com/sites/default/files/legacy/compaq_cst_1996_0.pdf; abgerufen am 23. November 2016. 1996.
- [Cra+11] Steve Crago u. a. „Heterogeneous cloud computing“. In: *Cluster Computing (CLUSTER), 2011 IEEE Int'l Conf. on*. IEEE. 2011, S. 378–385. DOI: 10.1109/CLUSTER.2011.49.
- [Cra16] Cray Inc. *Cray XD1 - FPGA Development (S-6400-14)*. Online unter: <https://github.com/VLSI-EDA/PoC>; abgerufen am 28. Oktober. 2016.
- [Cre81] Robert J. Creasy. „The origin of the VM/370 time-sharing system“. In: *IBM Journal of Research and Development* 25.5 (1981), S. 483–490. DOI: 10.1147/rd.255.0483.
- [Cro+14] Louise H Crockett, Ross A Elliot, Martin A Enderwitz und Robert W Stewart. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014. ISBN: 099297870X.
- [CS10] James Coole und Greg Stitt. „Intermediate fabrics: virtual architectures for circuit portability and fast placement and routing“. In: *Hardware/Software Codesign and System Synthesis (CODES+ISSS), Int'l Conf. on*. IEEE. 2010, S. 13–22. DOI: 10.1145/1878961.1878966.
- [CS14] Yuk-Ming Choi und Hayden Kwok-Hay So. „Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster“. In: *2014 IEEE 25th Int'l Conf. on Application-*

- Specific Systems, Architectures and Processors*. IEEE. 2014, S. 9–16. DOI: 10.1109/ASAP.2014.6868624.
- [CW16] João MP Cardoso und Markus Weinhardt. „High-Level Synthesis“. In: *FPGAs for Software Programmers*. Springer, 2016, S. 23–47. DOI: 10.1007/978-3-319-26408-0_2.
- [CW97] A Colin Cameron und Frank AG Windmeijer. „An R-squared measure of goodness of fit for some common nonlinear regression models“. In: *Journal of Econometrics* 77.2 (1997), S. 329–342.
- [Dag98] Leonardo Dagum et al. „OpenMP: an industry standard API for shared-memory programming“. In: *Computational Science & Engineering, IEEE* 5.1 (1998). DOI: 10.1109/99.660313.
- [DG08] Jeffrey Dean und Sanjay Ghemawat. „MapReduce: simplified data processing on large clusters“. In: *Communications of the ACM* 51.1 (2008), S. 107–113. DOI: 10.1145/1327452.1327492.
- [DK09] Daniel J Duffy und Joerg Kienitz. *Monte Carlo Frameworks: Building Customisable High-performance C++ Applications*. Bd. 406. John Wiley & Sons, 2009.
- [DLP03] Jack J Dongarra, Piotr Luszczek und Antoine Petitet. „The LINPACK Benchmark: past, present and future“. In: *Concurrency and Computation: practice and experience* 15.9 (2003), S. 803–820.
- [DMH16] Brian Degnan, Bo Marr und Jennifer Hasler. „Assessing Trends in Performance per Watt for Signal Processing Applications“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.1 (2016), S. 58–66. DOI: 10.1109/TVLSI.2015.2392942.
- [Don+15] Julio Dondo Gazzano, Francisco Sanchez Molina, Fernando Rincon und Juan Carlos López. „Integrating reconfigurable hardware-based grid for high performance computing“. In: *The Scientific World Journal* 2015 (2015). DOI: 10.1155/2015/272536.
- [DOW92] Whitfield Diffie, Paul C Oorschot und Michael J Wiener. „Authentication and authenticated key exchanges“. In: *Designs, Codes and cryptography* 2.2 (1992), S. 107–125.
- [DRN13] Tomáš Drahonovský, Martin Rozkovec und Ondrej Novák. „Relocation of reconfigurable modules on Xilinx FPGA“. In: *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2013 IEEE 16th Int'l Symp. on*. IEEE. 2013, S. 175–180. DOI: 10.1109/DDECS.2013.6549812.
- [DS07] Kushal Datta und Ron Sass. „Rboot: Software infrastructure for a remote FPGA laboratory“. In: *Field-Programmable Custom Computing Machines (FCCM), 15th Annual IEEE Symp. on*. IEEE. 2007, S. 343–344. DOI: 10.1109/FCCM.2007.53.
- [DTB10] Florian Devic, Lionel Torres und Benoit Badrignans. „Secure protocol implementation for remote bitstream update preventing replay attacks on FPGA“. In: *Field Programmable Logic and Applications (FPL), 2010 Int'l Conf. on*. IEEE. 2010, S. 179–182. DOI: 10.1109/FPL.2010.44.
- [Dua12] Yucong Duan. „Value modeling and calculation for everything as a service (XaaS) based on reuse“. In: *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS Int'l Conf. on*. IEEE. 2012, S. 162–167. DOI: 10.1109/SNPD.2012.30.
- [Eck14] Marcel Eckert. „FPGA-based system virtual machines“. Dissertation. Helmut Schmidt Universität, Hamburg, 2014.
- [EGE08] Esam El-Araby, Ivan Gonzalez und Tarek El-Ghazawi. „Virtualizing and sharing reconfigurable resources in High-Performance Reconfigurable Computing systems“. In: *High-Performance Reconfigurable Computing Technology and Applications (HPRCTA), Second Int'l Workshop on*. IEEE. 2008, S. 1–8. DOI: 10.1109/HPRCTA.2008.4745683.
- [EGE09] Esam El-Araby, Ivan Gonzalez und Tarek El-Ghazawi. „Exploiting partial runtime reconfiguration for high-performance reconfigurable computing“. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 1.4 (2009), S. 21. DOI: 10.1145/1462586.1462590.
- [Egu10] Ken Eguro. „SIRC: An extensible reconfigurable computing communication API“. In: *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual Int'l Symp. on*. IEEE. 2010, S. 135–138. DOI: 10.1109/FCCM.2010.29.

Literaturverzeichnis

- [Eis+07] Thomas Eisenbarth, Tim Güneysu, Christof Paar, Ahmad-Reza Sadeghi, Dries Schellekens und Marko Wolf. „Reconfigurable trusted computing in hardware“. In: *Proc. of the 2007 ACM workshop on Scalable trusted computing*. ACM. 2007, S. 15–20. DOI: 10.1145/1314354.1314360.
- [EPM13] Thomas Erl, Ricardo Puttini und Zaigham Mahmood. *Cloud computing: concepts, technology, & architecture*. Pearson Education, 2013.
- [Est60] Gerald Estrin. „Organization of computer systems: the fixed plus variable structure computer“. In: *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer Conf.* ACM. 1960, S. 33–40. DOI: 10.1145/1460361.1460365.
- [EV12] Ken Eguro und Ramarathnam Venkatesan. „FPGAs for trusted cloud computing“. In: *Field Programmable Logic and Applications (FPL), 22nd Int'l Conf. on*. IEEE. 2012, S. 63–70. DOI: 10.1109/FPL.2012.6339242.
- [Fen+11] Deng-Guo Feng, Min Zhang, Yan Zhang und Zhen Xu. „Study on cloud computing security“. In: *Journal of software* 22.1 (2011), S. 71–83.
- [Fer+12] Michael Ferdman u. a. „Clearing the clouds: a study of emerging scale-out workloads on modern hardware“. In: *ACM SIGPLAN Notices*. Bd. 47. 4. ACM. 2012, S. 37–48. DOI: 10.1145/2248487.2150982.
- [Fer+14] Diogo AB Fernandes, Liliana FB Soares, João V Gomes, Mário M Freire und Pedro RM Inácio. „Security issues in cloud environments: a survey“. In: *Int'l Journal of Information Security* 13.2 (2014), S. 113–170. DOI: 10.1007/s10207-013-0208-7.
- [Fig+11] Peter Figuli u. a. „A heterogeneous SoC architecture with embedded virtual FPGA cores and runtime Core Fusion“. In: *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conf. on*. IEEE. 2011, S. 96–103. DOI: 10.1109/AHS.2011.5963922.
- [FKL07] Ludwig Fahrmeir, Thomas Kneib und Stefan Lang. „Lineare Regressionsmodelle“. In: *Regression: Modelle, Methoden und Anwendungen* (2007), S. 59–188.
- [Fly72] Michael J Flynn. „Some computer organizations and their effectiveness“. In: *IEEE transactions on computers* 100.9 (1972), S. 948–960. DOI: 10.1109/TC.1972.5009071.
- [Fos+08] Ian Foster, Yong Zhao, Ioan Raicu und Shiyong Lu. „Cloud computing and grid computing 360-degree compared“. In: *2008 Grid Computing Environments Workshop*. IEEE. 2008, S. 1–10. DOI: 10.1109/GCE.2008.4738445.
- [Fou17] Python Software Foundation. *Python 3.5.4 documentation*. Website. Online unter: <https://docs.python.org/3.5/>; abgerufen am 23. März 2017. 2017.
- [Fow+13] Jeremy Fowers, Greg Brown, John Wernsing und Greg Stitt. „A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors“. In: *Transactions on Architecture and Code Optimization (TACO)* 9.4 (Jan. 2013), S. 1–21. DOI: 10.1145/2400682.2400684.
- [FP98a] William Fornaciari und Vincenzo Piuri. „General methodologies to virtualize FPGAs in Hw/Sw systems“. In: *Circuits and Systems, 1998. Proceedings. 1998 Midwest Symp. on*. IEEE. 1998, S. 90–93. DOI: 10.1109/MWSCAS.1998.759442.
- [FP98b] William Fornaciari und Vincenzo Piuri. „Virtual FPGAs: Some steps behind the physical barriers“. In: *Parallel and Distributed Processing*. Springer, 1998, S. 7–12. DOI: 10.1007/3-540-64359-1_665.
- [FPG05] Virtex-II Platform FPGA. „Complete Data Sheet“. In: *Xilinx, Inc.*, Mar (2005).
- [FPR00] William Fornaciari, Vincenzo Piuri und Luigi Ripamonti. „Virtualization of FPGA via Segmentation“. In: *Proc. ACM Int. Symp. on FPGA*. 2000. DOI: 10.1145/329166.329226.
- [FVS15] Suhaib A Fahmy, Kizheppatt Vipin und Shanker Shreejith. „Virtualized FPGA accelerators for efficient cloud computing“. In: *Cloud Computing Technology and Science (CloudCom), Int'l Conf. on*. IEEE, 2015, S. 430–435. DOI: 10.1109/CloudCom.2015.60.
- [FWG10] Yiqiu Fang, Fei Wang und Junwei Ge. „A task scheduling algorithm based on load balancing in cloud computing“. In: *Int'l Conf. on Web Information Systems and Mining*. Springer. 2010, S. 271–277. DOI: 10.1007/978-3-642-16515-3_34.

- [Gac+08] Nicolas Gac, Stéphane Mancini, Michel Desvignes und Dominique Houzet. „High speed 3D tomography on CPU, GPU, and FPGA“. In: *EURASIP Journal on Embedded systems* 2008 (2008), S. 5. DOI: 10.1155/2008/930250.
- [Gaj+12] Daniel D Gajski, Nikil D Dutt, Allen CH Wu und Steve YL Lin. *High—Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.
- [Gen+09] Craig Gentry u. a. „Fully homomorphic encryption using ideal lattices“. In: *Proc. of the 41st annual ACM symposium on Symp. on theory of computing - STOC*. Bd. 9. 2009, S. 169–178. DOI: 10.1145/1536414.1536440.
- [Gen15] Paul Richard Genßler. „Virtualisation of FPGA-Resources for Concurrent User Designs Employing Partial Dynamic Reconfiguration“. In: *Study's Thesis (Belegarbeit)*, Technische Universität Dresden (2015). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [Gen17] Paul Richard Genßler. „Virtualized Reconfigurable Resources and Their Secured Provision in an Unstructured Cloud Environment“. In: *Diploma Thesis (Diplomarbeit)*, Technische Universität Dresden (2017). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [GG13] Sean Gulley und Vinodh Gopal. „Haswell cryptographic performance“. In: *Intel Corporation* (2013).
- [GJ79] Michael R Garey und David S Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. 1979.
- [GKS17] Paul R. Genßler, Oliver Knodel und Rainer G. Spallek. „A New Level of Trusted Cloud Computing - Virtualized Reconfigurable Resources in a Security-First Architecture“. In: *47. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2017, Lecture Notes in Informatics (LNI), Chemnitz, Germany, September 25-29*. Gesellschaft für Informatik, Bonn. 2017, S. 531–542. DOI: 10.18420/in2017_48. URL: https://doi.org/10.18420/in2017_48.
- [Gla+08] Benjamin Glas, Alexander Klimm, Oliver Sander, Klaus Müller-Glaser und Jürgen Becker. „A system architecture for reconfigurable trusted platforms“. In: *Proceedings of the Conf. on Design, automation and test in Europe*. ACM. 2008, S. 541–544. DOI: 10.1145/1403375.1403505.
- [Gol73] Robert P Goldberg. *Architectural principles for virtual computer systems*. Techn. Ber. DTIC Document, HARVARD UNIV CAMBRIDGE MA DIV OF ENGINEERING und APPLIED PHYSICS, 1973.
- [Gol74] Robert P Goldberg. „Survey of virtual machine research“. In: *Computer Journal* 7.6 (1974), S. 34–45. DOI: 10.1109/MC.1974.6323581.
- [Gon+12] Ivan Gonzalez, Sergio Lopez-Buedo, Gustavo Sutter, Diego Sanchez-Roman, Francisco J Gomez-Arribas und Javier Aracil. „Virtualization of reconfigurable coprocessors in HPRC systems with multicore architecture“. In: *Journal of Systems Architecture* 58.6 (2012), S. 247–256.
- [Gra03] Ananth Grama. *Introduction to parallel computing*. Pearson Education, 2003. ISBN: 978-0201648652.
- [Gri+14] Paul Grigoras, Max Tottenham, Xinyu Niu, Jose GF Coutinho und Wayne Luk. „Elastic Management of Reconfigurable Accelerators“. In: *2014 IEEE Int'l Symp. on Parallel and Distributed Processing with Applications*. IEEE. 2014, S. 174–181. DOI: 10.1109/ISPA.2014.31.
- [Gro11] Ian Grout. *Digital systems design with FPGAs and CPLDs*. Newnes, 2011.
- [GS16] Ying Gao und Timothy Sherwood. „Hardware-Assisted Context Management for Accelerator Virtualization: A Case Study with RSA“. In: *Int'l Conf. on Architecture of Computing Systems*. Springer. 2016, S. 72–83. DOI: 10.1007/978-3-319-30695-7_6.
- [Gup+09] Vishakha Gupta u. a. „GVIM: GPU-accelerated virtual machines“. In: *System-level Virtualization for High Performance Computing, Proc. of the 3rd ACM Workshop on*. ACM. 2009, S. 17–24. DOI: 10.1145/1519138.1519141.
- [Gup15] Prabhat K. Gupta. „Intel Xeon+FPGA Platform for the Data Center“. In: *Keynote, Workshop on Reconfigurable Computing for the Masses*. 2015.
- [Gus88] J L Gustafson. „Reevaluating Amdahl's law“. In: *Communications of the ACM* (1988). DOI: 10.1145/42411.42415.

Literaturverzeichnis

- [GVB13] Saurabh Kumar Garg, Steve Versteeg und Rajkumar Buyya. „A framework for ranking of cloud computing services“. In: *Future Generation Computer Systems* 29.4 (2013), S. 1012–1023. DOI: 10.1016/j.future.2012.06.006.
- [Har01] Reiner Hartenstein. „A decade of reconfigurable computing: a visionary retrospective“. In: *Design, automation and test in Europe, Proc. of the Conf. on*. IEEE. 2001, S. 642–649. ISBN: 0-7695-0993-2.
- [Has01] Michael Hasenstein. „The logical volume manager (LVM)“. In: *White paper* (2001).
- [HD08] Scott Hauck und Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation*. Morgan Kaufmann, 2008. ISBN: 978-0-123-70522-8.
- [Hen00] John L Henning. „SPEC CPU2000: Measuring CPU performance in the new millennium“. In: *Computer* 33.7 (2000), S. 28–35. DOI: 10.1109/2.869367.
- [HHW10] Jan Haase, Andreas Hofmann und Klaus Waldschmidt. „A self distributing virtual machine for adaptive multicore environments“. In: Bd. 38. 1. Springer, 2010, S. 19–37. DOI: 10.1007/s10766-009-0119-4.
- [Hig04] Desmond J Higham. „Black Scholes for Scientific Computing Students“. In: *Computing in Science & Engineering* 6.6 (2004), S. 72–79. DOI: 10.1109/MCSE.2004.62.
- [HKR13] Nikolas Roman Herbst, Samuel Kounev und Ralf Reussner. „Elasticity in cloud computing: What it is, and what it is not“. In: *Proceedings of the 10th Int'l Conf. on Autonomic Computing (ICAC 13)*. 2013, S. 23–27.
- [HM04] Göran Herrmann und Dietmar Müller. „ASIC–Entwurf und Test“. In: *Fachbuchverlag Leipzig im Carl-Hanser-Verlag* (2004).
- [Hoe16] James C Hoe. „FPGA compute acceleration is first about energy efficiency: technical perspective“. In: *Communications of the ACM* 59.11 (2016), S. 113–113. DOI: 10.1145/2996866.
- [Hou+16] Ernst Joachim Houtgast, Vlad-Mihai Sima, Giacomo Marchiori, Koen Bertels und Zaid Al-Ars. „Power-efficiency analysis of accelerated BWA-MEM implementations on heterogeneous computing platforms“. In: *ReConfigurable Computing and FPGAs (ReConFig), 2016 Int'l Conf. on*. IEEE. 2016, S. 1–8. DOI: 10.1109/ReConFig.2016.7857181.
- [HTK15] Markus Happe, Andreas Traber und Ariane Keller. „Preemptive Hardware Multitasking in ReconOS“. In: *Applied Reconfigurable Computing*. Springer, 2015, S. 79–90. DOI: 10.1007/978-3-319-16214-0_7.
- [Hua+07] Wei Huang, Matthew J Koop, Qi Gao und Dhabaleswar K Panda. „Virtual machine aware communication libraries for high performance computing“. In: *Proceedings of the 2007 ACM/IEEE Conf. on Supercomputing*. ACM. 2007, S. 9. DOI: 10.1145/1362622.1362635.
- [Hus+11] Hanaa M Hussain, Khaled Benkrid, Ahmet T Erdogan und Huseyin Seker. „Highly parameterized K-means clustering on FPGAs: Comparative results with GPPs and GPUs“. In: *2011 Int'l Conf. on Reconfigurable Computing and FPGAs*. IEEE. 2011, S. 475–480. DOI: 10.1109/ReConFig.2011.49.
- [HV10] Till Haselmann und Gottfried Vossen. *Database-as-a-Service für kleine und mittlere Unternehmen*. Techn. Ber. Working Paper (3), Institut für Wirtschaftsinformatik, Westfälische Wilhelms-Universität Münster, Münster, 2010.
- [IBM15a] IBM China. „New OpenPOWER Cloud Boosts Ecosystem for Innovation and Development“. In: *IBM News Release* (2015).
- [IBM15b] IBM China. *New OpenPOWER Cloud Boosts Ecosystem for Innovation and Development*. Online unter: <http://www-03.ibm.com/press/us/en/pressrelease/47082.wss>; abgerufen am 30. November 2016. 2015.
- [Ich+12] Yoshihiro Ichinomiya, Sadaki Usagawa, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga und Toshinori Sueyoshi. „Designing flexible reconfigurable regions to relocate partial bit-streams“. In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual Int'l Symp. on*. IEEE. 2012, S. 241–241. DOI: 10.1109/FCCM.2012.51.
- [IDC16] IDC (Int'l Data Corporation) Research, Inc. *Worldwide Semiannual Public Cloud Services Spending Guide*. 2016.

- [Int16a] Intel Corporation. *Intel Core i7-5960X Processor*. Online unter: http://ark.intel.com/de/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz; abgerufen am 8. Dezember 2016. 2016.
- [Int16b] Intel Corporation. *Stratix 10 GX/SX Device Family Table*. Online unter: <https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html>; abgerufen am 8. Dezember 2016. 2016.
- [Int16c] Intel Corporation. *The Intel Xeon Phi Product Family*. Online unter: <http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/high-performance-xeon-phi-coprocessor-brief.pdf>; abgerufen am 8. Dezember 2016. 2016.
- [Int16d] Intel Corporation. „Using the Design Security Features in Altera FPGAs“. In: *Application Note AN-556*, 01. Juni 2016. 2016.
- [Int17a] Intel Corporation. *Intel AtomTM Processor E6x5C Series-Based Platform for Embedded Computing*. Online unter: https://newsroom.intel.com/wp-content/uploads/sites/11/2016/01/ProductBrief-IntelAtomProcessor_E600C_series.pdf; abgerufen am 24. Januar 2017. 2017.
- [Int17b] Intel Corporation. *Intel SoCs: When Architecture Matters*. Online unter: <https://www.altera.com/products/soc/overview.html>; abgerufen am 24. Januar 2017. 2017.
- [Int17c] Intel Corporation. *Intel Software Guard Extensions (Intel SGX)*. Online unter: <https://software.intel.com/en-us/sgx/details>; abgerufen am 8. März 2017. 2017.
- [Ior+16] Anca Iordache, Guillaume Pierre, Peter Sanders, Jose Gabriel de F Coutinho und Mark Stillwell. „High performance in the cloud with FPGA groups“. In: *Utility and Cloud Computing (UCC), 2016 IEEE/ACM 9th Int'l Conf. on*. IEEE. 2016, S. 1–10. ISBN: 978-1-4503-4616-0.
- [ITA16] ITA – The Internet Traffic Archive. *EPA-HTTP – A day of HTTP logs from a EPA WWW server*. Website. Online unter: <http://ita.ee.lbl.gov/html/contrib/EPA-HTTPhtml>; abgerufen am 15. Oktober. 2016.
- [Jac+15] Matthew Jacobsen, Dustin Richmond, Matthew Hogains und Ryan Kastner. „RIFFA 2.1: A reusable integration framework for FPGA accelerators“. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 8.4 (2015), S. 22. doi: 10.1145/2815631.
- [JD88] Anil K Jain und Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.
- [JK07] Seul Jung und Sung su Kim. „Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems“. In: *Industrial Electronics, IEEE Transactions on* 54.1 (2007), S. 265–271. doi: 10.1109/TIE.2006.888791.
- [Jon+10] David H Jones, Adam Powell, Christos-Savvas Bouganis und Peter YK Cheung. „GPU versus FPGA for high productivity computing“. In: *Field Programmable Logic and Applications (FPL), 2010 Int'l Conf. on*. IEEE. 2010, S. 119–124. doi: 10.1109/FPL.2010.32.
- [Jon07] L Jones. „Single Event Upset (SEU) Detection and Correction using Virtex-4 Devices“. In: *Xilinx Application Note # 714*. 2007.
- [Jov+07] Slavisa Jovanovic u. a. „A hardware preemptive multitasking mechanism based on scan-path register structure for FPGA-based reconfigurable systems“. In: *Adaptive Hardware and Systems (AHS). NASA/ESA Conf. on*. IEEE. 2007. doi: 10.1109/AHS.2007.4.
- [JRV08] G. S. Jedhe, A. Ramamoorthy und K. Varghese. „A Scalable High Throughput Firewall in FPGA“. In: *2008 16th Int'l Symp. on Field-Programmable Custom Computing Machines*. Apr. 2008, S. 43–52. doi: 10.1109/FCCM.2008.31.
- [JTS05] Kimmo Järvinen, Matti Tommiska und Jorma Skyttä. „Comparative survey of high-performance cryptographic algorithm implementations on FPGAs“. In: *IEE Proceedings-Information Security* 152.1 (2005), S. 3–12. doi: 10.1049/ip-ifs:20055004.
- [Kac+16a] Christoforos Kachris, Dionysios Diamantopoulos, Georgios Ch Sirakoulis und Dimitrios Soudris. „An FPGA-based Integrated MapReduce Accelerator Platform“. In: *Journal of Signal Processing Systems* (2016), S. 1–13. doi: 10.1007/s11265-016-1108-7.
- [Kac+16b] Christoforos Kachris u. a. „The VINEYARD project: Versatile integrated accelerator-based heterogeneous data centres“. In: *Modern Circuits and Systems Technologies (MOCAST), 2016 5th Int'l Conf. on*. IEEE. 2016, S. 1–4. doi: 10.1109/MOCAST.2016.7495121.

Literaturverzeichnis

- [Kah+05] James A Kahle, Michael N Day, H Peter Hofstee, Charles R Johns u. a. „Introduction to the cell multiprocessor“. In: *IBM journal of Research and Development* 49.4/5 (2005), S. 589.
- [Kan15] Valentin Kandetzki. „Evaluation eines Scheduling-Algorithmus für eine elastische Cloud mit rekonfigurierbaren Hardwarebeschleunigern“. In: *Belegarbeit, Technische Universität Dresden* (2015). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [Kao05] Cindy Kao. „Benefits of partial reconfiguration“. In: *Xcell journal* 55 (2005), S. 65–67.
- [Kap+06] Nachiket Kapre u. a. „Packet switched vs. time multiplexed FPGA overlay networks“. In: *Field-Programmable Custom Computing Machines (FCCM), 14th Annual IEEE Symp. on*. IEEE. 2006, S. 205–216. DOI: 10.1109/FCCM.2006.55.
- [Kap+15] Nachiket Kapre, Harnhua Ng, Kirvy Teo und Jaco Naude. „Intime: A machine learning approach for efficient selection of fpga cad tool parameters“. In: *Proc. of the 2015 ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays*. ACM. 2015, S. 23–26. DOI: 10.1145/2684746.2689081.
- [Kav14] Michael J Kavis. *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. John Wiley & Sons, 2014.
- [KB16] Hiliwi Leake Kidane und El-Bay Bourennane. „NoC based virtualized FPGA as cloud Services“. In: *3rd Int'l Conf. on Embedded Systems in Telecommunications and Instrumentation (ICESTI'16)*. 2016.
- [KBL13] Dirk Koch, Christian Beckhoff und Guy GF Lemieux. „An efficient FPGA overlay for portable custom instruction set extensions“. In: *Field Programmable Logic and Applications (FPL), 23rd Int'l Conf. on*. IEEE. 2013, S. 1–8. DOI: 10.1109/FPL.2013.6645517.
- [KBT08] Dirk Koch, Christian Beckhoff und Jurgen Teich. „Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs“. In: *Field Programmable Logic and Applications, 2008. FPL 2008. Int'l Conf. on*. IEEE. 2008, S. 119–124. DOI: 10.1109/FPL.2008.4629918.
- [KCL08] Alexander Kaganov, Paul Chow und Asif Lakhany. „FPGA acceleration of Monte-Carlo based credit derivative pricing“. In: *Field Programmable Logic and Applications, 2008. FPL 2008. Int'l Conf. on*. IEEE. 2008, S. 329–334. DOI: 10.1109/FPL.2008.4629953.
- [KDW10] Srinidhi Kestur, John D Davis und Oliver Williams. „BLAS comparison on FPGA, CPU and GPU“. In: *VLSI (ISVLSI), 2010 IEEE computer society annual Symp. on*. IEEE. 2010, S. 288–293. DOI: 10.1109/ISVLSI.2010.84.
- [Kem15] Alexander Kemnitz. „Realisierung eines Speichermanagements zur Zugriffsvirtualisierung von konkurrierenden Nutzerdesigns auf rekonfigurierbarer Hardware“. In: *Studienarbeit, Technische Universität Dresden* (2015). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [Kep+08] Krzysztof Kepa, Fearghal Morgan, Krzysztof Kosciuszkoiewicz und Tomasz Surmacz. „Serecon: A secure dynamic partial reconfiguration controller“. In: *Symp. on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual*. IEEE. 2008, S. 292–297. DOI: 10.1109/ISVLSI.2008.61.
- [KG16] Lester Kalms und Diana Görninger. „Clustering and Mapping Algorithm for Application Distribution on a Scalable FPGA Cluster“. In: *Parallel and Distributed Processing Symp. Workshops, 2016 IEEE Int'l*. IEEE. 2016, S. 105–113. DOI: 10.1109/IPDPSW.2016.75.
- [KGS16] Oliver Knodel, Paul Genßler und Rainer Spallek. „Migration of long-running Tasks between Reconfigurable Resources using Virtualization“. In: *ACM SIGARCH Computer Architecture News Volume 44, HEART 2016, Hongkong, Hongkong, July 25-27, 2016*. Bd. 44. 4. ACM. 2016, S. 56–61. DOI: 10.1145/3039902.3039913.
- [KGS17] Oliver Knodel, Paul R. Gessler und Rainer G. Spallek. „Virtualizing Reconfigurable Hardware to Provide Scalability in Cloud Architectures“. In: *Reconfigurable Architectures, Tools and Applications, RECATA 2017, Rome, Italy, September 10 - 14, ISBN: 978-1-61208-585-2, CENICS 2017 Best Paper Award*. IARIA. 2017, S. 33–38.
- [KHT07] Dirk Koch, Christian Haubelt und Jürgen Teich. „Efficient hardware checkpointing: concepts, overhead analysis, and implementation“. In: *Proc. of the ACM/SIGDA 15th Int'l Symp. on Field programmable gate arrays*. ACM. 2007. DOI: 10.1145/1216919.1216950.

- [KHZ16] Dirk Koch, Frank Hannig und Daniel Ziener. *FPGAs for Software Programmers*. Springer, 2016. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0.
- [Kim+08] Kangho Kim, Cheiyol Kim, Sung-In Jung, Hyun-Sup Shin und Jin-Soo Kim. „Inter-domain socket communications supporting high performance and full binary compatibility on Xen“. In: *Proceedings of the fourth ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual execution environments*. ACM. 2008, S. 11–20. DOI: 10.1145/1346256.1346259.
- [Kir+12] Robert Kirchgessner, Greg Stitt, Alan George und Herman Lam. „VirtualRC: a virtual FPGA platform for applications and tools portability“. In: *Field Programmable Gate Arrays, Proc. of the ACM/SIGDA Int'l Symp. on*. ACM. 2012, S. 205–208. DOI: 10.1145/2145694.2145728.
- [Kiv+07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin und Anthony Liguori. „KVM: The Linux virtual machine monitor“. In: *Linux Symp, Proc. of the*. Bd. 1. 2007, S. 225–230.
- [Kle69] Leonard Kleinrock. „Models for computer networks“. In: *Proc. of the Int'l Conf. on Communications*. 1969, S. 21–9.
- [KLS16] Oliver Knodel, Patrick Lehmann und Rainer G Spallek. „RC3E: Reconfigurable Accelerators in Data Centres and their Provision by Adapted Service Models“. In: *9th IEEE Int'l Conf. on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2*. IEEE. 2016, S. 19–26. DOI: 10.1109/CLOUD.2016.0013.
- [KM11] Stephen Kaisler und William H Money. „Service migration in a cloud architecture“. In: *System Sciences (HICSS), 44th Hawaii Int'l Conf. on*. IEEE. 2011, S. 1–10. DOI: 10.1109/HICSS.2011.371.
- [Kna14] Thomas Knauth. „Energy Efficient Cloud Computing: Techniques and Tools“. Diss. Saechsische Landesbibliothek-Staats-und Universitaetsbibliothek Dresden, 2014.
- [Kno14] Oliver Knodel. „Vorlesung: Hochparallele Simulationsrechnungen mit CUDA und OpenCL – Architektur“. In: *GPU Center of Excellence Dresden* (2014).
- [Knö17] Martin Knöfel. „Übertragung eines an MapReduce orientierten k-Means-Algorithmus auf einen FPGA-Hardwarebeschleuniger für den Cloud-Einsatz“. In: *Belegarbeit, Technische Universität Dresden* (2017). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [Koc+13] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim und Parthasarathy Ranganathan. „Meet the walkers: Accelerating index traversals for in-memory databases“. In: *Proc. of the 46th Annual IEEE/ACM Int'l Symp. on Microarchitecture*. ACM. 2013, S. 468–479. DOI: 10.1145/2540708.2540748.
- [Koo+09] Jonathan G Koomey, Christian Belady, Michael Patterson, Anthony Santos und Klaus-Dieter Lange. „Assessing trends over time in performance, costs, and energy use for servers“. In: *Lawrence Berkeley National Laboratory, Stanford University, Microsoft Corporation, and Intel Corporation, Tech. Rep* (2009).
- [KPS11] Oliver Knodel, Thomas B. Preußer und Rainer G. Spallek. „Next-generation massively parallel short-read mapping on FPGAs“. In: *22nd IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors, ASAP 2011, Santa Monica, CA, USA, Sept. 11-14*. IEEE. 2011, S. 195–201. DOI: 10.1109/ASAP.2011.6043268.
- [KR07] Ian Kuon und Jonathan Rose. „Measuring the gap between FPGAs and ASICs“. In: *IEEE transactions on computer-aided design of integrated circuits and systems* 26.2 (2007), S. 203–215. DOI: 10.1109/TCAD.2006.884574.
- [Kru09] Hans Günther Kruse. *Leistungsbewertung bei Computersystemen: Praktische Performance-Analyse von Rechnern und ihrer Kommunikation*. Springer Science & Business Media, 2009.
- [KS15a] Simon Kiertscher und Bettina Schnor. „Scalability evaluation of an energy-aware resource management system for clusters of web servers“. In: *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2015 Int'l Symp. on*. IEEE. 2015, S. 1–8. DOI: 10.1109/SPECTS.2015.7285285.
- [KS15b] Oliver Knodel und Rainer G. Spallek. „Computing Framework for Dynamic Integration of Reconfigurable Resources in a Cloud“. In: *2015 Euromicro Conf. on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28*. IEEE. 2015, S. 337–344. DOI: 10.1109/DSD.2015.37.

Literaturverzeichnis

- [KS15c] Oliver Knodel und Rainer G. Spallek. „RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment“. In: *Second Int'l Workshop on FPGAs for Software Programmers, FSP, London, United Kingdom, 1. September 2015*. arXiv preprint. 2015. URL: <http://arxiv.org/abs/1508.06843>.
- [KS16] Christoforos Kachris und Dimitrios Soudris. „A survey on reconfigurable accelerators for cloud computing“. In: *Field Programmable Logic and Applications (FPL), 2016 26th Int'l Conf. on*. EPFL. 2016, S. 1–10. DOI: 10.1109/FPL.2016.7577381.
- [Kun11] Vivek Kundra. „Federal cloud computing strategy“. In: *White House [Chief Information Officers Council]*. 2011.
- [KV10] Ronald L Krutz und Russell Dean Vines. *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing, 2010.
- [KW12] David B Kirk und W Hwu Wen-mei. *Programming Massively Parallel Processors: A hands-on Approach*. Newnes, 2012.
- [KZH16] Dirk Koch, Daniel Ziener und Frank Hannig. „FPGA Versus Software Programming: Why, When, and How?“ In: *FPGAs for Software Programmers*. Springer, 2016, S. 1–21. DOI: 10.1007/978-3-319-26408-0.
- [KZS13] Simon Kiertscher, Jörg Zinke und Bettina Schnor. „CHERUB: power consumption aware cluster resource management“. In: *Cluster computing* 16.1 (2013), S. 55–63. DOI: 10.1007/s10586-011-0176-5.
- [Lac+14] Tiago Pais Pitta de Lacerda Ruivo u. a. „Efficient High-Performance Computing with Infini-band Hardware Virtualization“. In: *Technical Reports – Illinois Institute of Technology, Argonne National Laboratory* (2014).
- [LCK11] Gunho Lee, Byung-Gon Chun und Randy H Katz. „Heterogeneity-aware resource allocation and scheduling in the cloud“. In: *Proc. of HotCloud* (2011), S. 1–5.
- [Leh16] Lehrstuhl für VLSI Design, Diagnostic und Architektur, Fakultät Informatik, Technische Universität Dresden. *PoC - Pile of Cores*. Website. Online unter: <https://github.com/VLSI-EDA/PoC>; abgerufen am 28. Oktober. 2016.
- [Len+09] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai und Thomas Sandholm. „What's inside the Cloud? An architectural map of the Cloud landscape“. In: *Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society. 2009, S. 23–31. DOI: 10.1109/CLOUD.2009.5071529.
- [LGL11] Christian Leber, Benjamin Geib und Heiner Litz. „High frequency trading acceleration using FPGAs“. In: *2011 21st Int'l Conf. on Field Programmable Logic and Applications*. IEEE. 2011, S. 317–322. DOI: 10.1109/FPL.2011.64.
- [Li+09] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li und Liang Zhong. „Enacloud: An energy-saving application live placement approach for cloud computing environments“. In: *2009 IEEE Int'l Conf. on Cloud Computing*. IEEE. 2009, S. 17–24. DOI: 10.1109/CLOUD.2009.72.
- [Lie95] Franz Liebl. „Problemorientierte Einführung. 2“. In: *Auflage, Oldenburg, München/Wien* (1995).
- [Liu+09] Ming Liu, Wolfgang Kuehn, Zhonghai Lu und Axel Jantsch. „Run-time partial reconfiguration speed investigation and architectural design space exploration“. In: *2009 Int'l Conf. on Field Programmable Logic and Applications*. IEEE. 2009, S. 498–502. DOI: 10.1109/FPL.2009.5272463.
- [Liu+13] Haikun Liu, Hai Jin, Cheng-Zhong Xu und Xiaofei Liao. „Performance and energy modeling for live migration of virtual machines“. In: *Cluster computing* 16.2 (2013), S. 249–264. DOI: 10.1007/s10586-011-0194-3.
- [LP07] Enno Lübbbers und Marco Platzner. „ReconOS: An RTOS supporting hard-and software threads“. In: *Field Programmable Logic and Applications (FPL), Int'l Conf. on*. IEEE. 2007, S. 441–446. DOI: 10.1109/FPL.2007.4380686.
- [LSS11] Jian Li, Marinko V Sarunic und Lesley Shannon. „Scalable, high performance Fourier domain optical coherence tomography: Why FPGAs and not GPGPUs“. In: *Field-Programmable*

- Custom Computing Machines (FCCM), 2011 IEEE 19th Annual Int'l Symp. on.* IEEE. 2011, S. 49–56. DOI: 10.1109/FCCM.2011.27.
- [Lüb10] Enno Lübbbers. „Multithreaded programming and execution models for reconfigurable hardware“. Dissertation. University of Paderborn, 2010.
- [Mad13] Thomas Madden. *The BLAST sequence analysis tool*. 2013.
- [McC61] J McCarthy. „Centennial Keynote Address“. In: *Massachusetts Institute of Technology* (1961).
- [McD08] Eric J McDonald. „Runtime FPGA partial reconfiguration“. In: *Aerospace Conf., 2008 IEEE*. IEEE. 2008, S. 1–7.
- [MCZ16] Vaibhawa Mishra, Qianqiao Chen und Georgios Zervas. „REoN: A protocol for reliable software-defined FPGA partial reconfiguration over network“. In: *ReConfigurable Computing and FPGAs (ReConFig), 2016 Int'l Conf. on*. IEEE. 2016, S. 1–7. DOI: 10.1109/ReConFig.2016.7857184.
- [MD97] Jon Meyer und Troy Downing. *Java virtual machine*. O'Reilly & Associates, Inc., 1997.
- [Mem16] Memcached. *What is Memcached?* Online unter: <https://memcached.org> ; abgerufen am 13. Oktober. 2016.
- [Men09] O. Mencer et al. „Cube: A 512-FPGA cluster“. In: *Programmable Logic (SPL), 5th Southern Conf. on*. IEEE, 2009. DOI: 10.1109/SPL.2009.4914907.
- [Mer+08] Saumil G Merchant u. a. „Strategic challenges for application development productivity in reconfigurable computing“. In: *Aerospace and Electronics Conf., 2008. NAECON 2008. IEEE National*. IEEE. 2008, S. 209–218. DOI: 10.1109/NAECON.2008.4806548.
- [Mer14] Dirk Merkel. „Docker: lightweight linux containers for consistent development and deployment“. In: *Linux Journal* 2014.239 (2014), S. 2.
- [Mer76] Robert C Merton. „Option pricing when underlying stock returns are discontinuous“. In: *Journal of financial economics* 3.1 (1976), S. 125–144. DOI: 10.1016/0304-405X(76)90022-2.
- [Mey12] B. Meyer et al. „Convey vector personalities - FPGA acceleration with an openmp-like programming effort?“ In: *2012 22nd Int'l Conf. on Field Programmable Logic and Applications (FPL)*. IEEE, 2012. ISBN: 978-1-4673-2255-3. DOI: 10.1109/FPL.2012.6339259.
- [Mey15] Dominik Meyer. „Multicore Reconfiguration Platform-A Research and Evaluation FPGA Framework for Runtime Reconfigurable Systems“. Dissertation. Helmut Schmidt Universität, Hamburg, 2015.
- [MG09] Peter Mell und Timothy Grance. „The NIST definition of cloud computing“. In: *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg* (2009).
- [MG11] Peter Mell und Timothy Grance. „The NIST definition of cloud computing, Revised“. In: *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg* (2011).
- [MG13] Aurelio Morales-Villanueva und Ann Gordon-Ross. „HTR: on-chip hardware task relocation for partially reconfigurable FPGAs“. In: *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2013. DOI: 10.1007/978-3-642-36812-7_18.
- [Mic16a] Microsoft. *Cloud Security & Cryptography*. Online unter: <https://www.microsoft.com/en-us/research/project/cloud-security-cryptography/> ; abgerufen am 30. November 2016. 2016.
- [Mic16b] Microsoft. *Project Catapult*. Online unter: <https://www.microsoft.com/en-us/research/project/project-catapult/> ; abgerufen am 24. November. 2016.
- [Mic16c] Microsoft Inc. *Microsoft Azure*. Website. Online unter: <https://azure.microsoft.com/> ; abgerufen am 6. Oktober. 2016.
- [Mic16d] Microsoft Inc. *Simple Encrypted Arithmetic Library - SEAL*. Website. Online unter: <https://sealcrypto.codeplex.com> ; abgerufen am 6. Oktober. 2016.
- [Mis+10] Asit K Mishra, Joseph L Hellerstein, Walfrido Cirne und Chita R Das. „Towards characterizing cloud backend workloads: insights from Google compute clusters“. In: *ACM SIGMETRICS Performance Evaluation Review* 37.4 (2010), S. 34–41.

Literaturverzeichnis

- [MK11] Dominik Meyer und Bernd Klauer. „Multicore reconfiguration platform an alternative to rampsoc“. In: *ACM SIGARCH Computer Architecture News* 39.4 (2011), S. 102–103. DOI: 10.1145/2082156.2082185.
- [MKH12] Viktor Mauch, Marcel Kunze und Marius Hillenbrand. „High performance cloud computing“. In: *Future Generation Computer Systems* (2012). DOI: 10.1016/j.future.2012.03.011.
- [MKS13] John Miller, James Kulp und Shepard Siegel. „Open Component Portability Infrastructure (OPENCPI)“. In: MERCURY FEDERAL SYSTEMS INC ARLINGTON VA, 2013.
- [MN06] Jing Ma und Jeffrey V Nickerson. „Hands-on, simulated, and remote laboratories: A comparative literature review“. In: *ACM Computing Surveys* 38.3 (2006), S. 7. DOI: 10.1145/1132960.1132961.
- [Mon11] Joel-Ahmed M. Mondol. „Cloud security solutions using FPGA“. In: *Communications, Computers and Signal Processing (PacRim), Pacific Rim Conf. on*. IEEE. 2011, S. 747–752. DOI: 10.1109/PACRIM.2011.6032987.
- [Moo65] G. E. Moore. „Cramming more components onto integrated circuits“. In: *Electronics. Band 38, Nr. 8* (1965), S. 114–117.
- [MS11] Anil Madhavapeddy und Satnam Singh. „Reconfigurable data processing for clouds“. In: *Field-Programmable Custom Computing Machines (FCCM), 19th Annual Int'l Symp. on*. IEEE. 2011, S. 141–145. DOI: 10.1109/FCCM.2011.35.
- [Mue03] Scott Mueller. *Upgrading and repairing PCs*. Que Publishing, 2003. ISBN: 978-0789-75610-7.
- [Mun+09] Aaftab Munshi u. a. „The opencl specification“. In: *Khronos OpenCL Working Group 1* (2009), S. 11–15.
- [Mus08] Luciano Musa. „FPGAs in high energy physics experiments at CERN“. In: *Field Programmable Logic and Applications (FPL), Int'l Conf. on*. IEEE. 2008, S. 2–2. DOI: 10.1109/FPL.2008.4629896.
- [Nap+14] Christian Napoli, Giuseppe Pappalardo, Emiliano Tramontana und Gaetano Zappalà. „A cloud-distributed GPU architecture for pattern identification in segmented detectors big-data surveys“. In: *The Computer Journal* 59.3 (2014), S. 338–352.
- [NG16] Hamid Nasiri und Maziar Goudarzi. „Dynamic FPGA-accelerator sharing among concurrently running virtual machines“ . In: *East-West Design & Test Symposium (EWDTs), 2016 IEEE*. IEEE. 2016, S. 1–4. ISBN: 978-1-55860-910-5.
- [Ngi16] Nginx. *We are proud to announce NGINX Plus R10!* Online unter: <https://nginx.org/en/> ; abgerufen am 13. Oktober. 2016.
- [Nim16] Nimbix. *Xilinx on the Nimbix Cloud*. Online unter: <https://www.nimbix.net/xilinx/> ; abgerufen am 30. November 2016. 2016.
- [NK14] Tuan DA Nguyen und Ajit Kumar. „PR-HMPSoC: A versatile partially reconfigurable heterogeneous Multiprocessor System-on-Chip for dynamic FPGA-based embedded systems“. In: *Field Programmable Logic and Applications (FPL), 2014 24th Int'l Conf. on*. IEEE. 2014, S. 1–6. DOI: 10.1109/FPL.2014.6927492.
- [NLV11] Michael Naehrig, Kristin Lauter und Vinod Vaikuntanathan. „Can homomorphic encryption be practical?“ In: *Proc. of the 3rd ACM workshop on Cloud computing security workshop*. ACM. 2011, S. 113–124. DOI: 10.1145/2046660.2046682.
- [NVI16] NVIDIA Corporation. *NVIDIA Tesla P100 – The Most Advanced Datacenter Accelerator Ever Built – Featuring Pascal GP100, the World's Fastest GPU*. Online unter: <https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf> ; abgerufen am 13. Dezember 2016. 2016.
- [NWZ13] Yun Niu, Liji Wu und Xiangmin Zhang. „An IPsec Accelerator Design for a 10Gbps In-Line Security Network Processor“. In: *JOURNAL OF COMPUTERS (JCP)* 8.2 (2013), S. 319–325.
- [NZ04] Adronis Niyonkuru und Hans Christoph Zeidler. „Designing a runtime reconfigurable processor for general purpose applications“. In: *Parallel and Distributed Processing Symp., 2004. Proceedings. 18th Int'l*. IEEE. 2004, S. 143. DOI: 10.1109/IPDPS.2004.1303123.

- [OCC16] Julio Proaño Orellana, Carmen Carrión und Blanca Caminero. „FPGA-aware Scheduling Strategies at Hypervisor Level in Cloud Environments“. In: *Scientific Programming*, vol. 2016, Article ID 4670271 (2016). DOI: 10.1155/2016/4670271.
- [OCR08] Diego Ongaro, Alan L Cox und Scott Rixner. „Scheduling I/O in virtual machine monitors“. In: *Proc. of the fourth ACM SIGPLAN/SIGOPS, Int'l Conf. on Virtual execution environments*. ACM. 2008, S. 1–10.
- [OlIs+12] Corey B Olson u.a. „Hardware acceleration of short read mapping“. In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual Int'l Symp. on*. IEEE. 2012, S. 161–168. DOI: 10.1109/FCCM.2012.36.
- [Oom+15] Roel Oomen, Tuan Nguyen, Akash Kumar und Henk Corporaal. „An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs“. In: *Field Programmable Logic and Applications (FPL), 2015 25th Int'l Conf. on*. IEEE. 2015, S. 1–4. DOI: 10.1109/FPL.2015.7293980.
- [Ope16a] OpenNebula. *OpenNebula - Software*. Website. Online unter: <http://opennebula.org/software/>; abgerufen am 6. Oktober. 2016.
- [Ope16b] OpenPOWER Foundation. *OpenPOWER*. Online unter: <https://openpowerfoundation.org>; abgerufen am 30. November 2016. 2016.
- [Ope16c] OpenStack. *OpenStack - Get started with OpenStack*. Website. Online unter: <http://docs.openstack.org/admin-guide/common/get-started-with-openstack.html>; abgerufen am 11. Oktober. 2016.
- [Ope16d] OpenStack. *OpenStack - Open Source Cloud Computing Software*. Website. Online unter: <http://www.openstack.org/software/>; abgerufen am 6. Oktober. 2016.
- [Ora16] Oracle Corporation. *MySQL*. Online unter: <http://www.mysql.com>; abgerufen am 17. Oktober. 2016.
- [Ovt+15] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss und Eric S Chung. „Accelerating deep convolutional neural networks using specialized hardware“. In: *Microsoft Research Whitepaper 2* (2015).
- [Owe10] Dustin Owens. „Securing elasticity in the cloud“. In: *Communications of the ACM* 53.6 (2010). ISSN: 0001-0782.
- [OWH17] Andreas Oeldemann, Thomas Wild und Andreas Herkersdorf. „Data Center Energy Optimization through Dynamic Load Management for Virtualized Network Functions“. In: *30th Int'l Conf. on Architecture of Computing Systems (ARCS)* (2017). DOI: 10.1109/CCGRID.2010.46.
- [Pag16] PageRank.net. *Search Engine Optimization*. Online unter: <https://www.pagerank.net>; abgerufen am 13. Oktober. 2016.
- [Pal16] Paloalto Networks. *What is a service level agreement?* Website. Online unter: <https://www.paloaltonetworks.com/documentation/glossary/what-is-a-service-level-agreement-sla>; abgerufen am 13. Oktober. 2016.
- [Par+10] Angshuman Parashar, Michael Adler, Kermin Fleming, Michael Pellauer und Joel Emer. „LEAP: A virtual platform architecture for FPGAs“. In: *The First Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL 2010)*. 2010.
- [Par66] Douglas F Parkhill. *Challenge of the computer utility*. Addison-Wesley, 1966.
- [PCC14] Julio Proaño, Carmen Carrión und Blanca Caminero. „An Open-Source Framework for Integrating Heterogeneous Resources in Private Clouds“. In: *CLOSER*. 2014, S. 129–134.
- [PCC16] Julio Proaño, Carmen Carrión und M. Blanca Caminero. „Towards a green, QoS-enabled heterogeneous cloud infrastructure“. In: *2016 IEEE Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW)*. IEEE. 2016, S. 7–16. DOI: 10.1109/IPDPSW.2016.12.
- [Pec+06] Wesley Peck, Erik Anderson, Jason Agron, Jim Stevens, Fabrice Baijot und David Andrews. „Hthreads: A computational model for reconfigurable devices“. In: *Field Programmable Logic and Applications (FPL), Int'l Conf. on*. IEEE. 2006, S. 1–4. DOI: 10.1109/FPL.2006.311336.
- [PG74] Gerald J Popek und Robert P Goldberg. „Formal requirements for virtualizable third generation architectures“. In: *Communications of the ACM* 17.7 (1974), S. 412–421. DOI: 10.1145/361011.361073.

Literaturverzeichnis

- [Pic09] Hans-Joachim Picht. *Xen-Kochbuch*. O'Reilly Media, Inc., 2009.
- [PKS12] Thomas B. Preußer, Oliver Knodel und Rainer G. Spallek. „Short-Read Mapping by a Systolic Custom FPGA Computation“. In: *2012 IEEE 20th Annual Int'l Symposium on Field-Programmable Custom Computing Machines, FCCM 2012, Toronto, Ontario, Canada, 29 April - 1 May*. IEEE. 2012, S. 169–176. DOI: 10.1109/FCCM.2012.37.
- [Plu+08] Daryl C Plummer, Thomas J Bittman, Tom Austin, David W Cearley und David Mitchell Smith. „Cloud computing: Defining and describing an emerging phenomenon“. In: *Gartner*, June 17 (2008). ID Number: G00156220.
- [Plu16] Plunify Pte Ltd. *Plunify – Simplify your chip design*. Website. Online unter: <http://www.plunify.com>; abgerufen am 23. November. 2016.
- [Pog+16] Marcello Pogliani, Gianluca C Durelli, Antonio Miele und Tobias Becker. „Quality of Service Driven Runtime Resource Allocation in Reconfigurable HPC Architectures“. In: (2016), S. 16–23. DOI: 10.1109/CSE-EUC-DCABES.2016.156.
- [Pöp+15] Thomas Pöppelmann, Michael Naehrig, Andrew Putnam und Adrian Macias. „Accelerating homomorphic evaluation on reconfigurable hardware“. In: *Int'l Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2015, S. 143–163. DOI: 10.1007/978-3-662-48324-4_8.
- [Pra+05] Ian Pratt u. a. „Xen 3.0 and the art of virtualization“. In: *Linux Symp*. Bd. 2. Ottawa, Ontario, Canada. 2005, S. 65–78.
- [Pre+16] T. B. Preußer, M. Zabel, P. Lehmann und 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, S. 1–6. DOI: 10.1109/ReConFig.2016.7857191.
- [Pro11] CSX700 Floating Point Processor. „Datasheet 06-PD-1425 Rev 1“. In: *ClearSpeed Technology Ltd* (2011).
- [Put+14] Andrew Putnam u. a. „A reconfigurable fabric for accelerating large-scale datacenter services“. In: *Computer Architecture (ISCA), 2014 ACM/IEEE 41st Int'l Symp. on*. IEEE. 2014, S. 13–24. DOI: 10.1109/ISCA.2014.6853195.
- [Put+15] Deepak Puthal, BPS Sahoo, Sambit Mishra und Satyabrata Swain. „Cloud computing features, issues, and challenges: a big picture“. In: *Computational Intelligence and Networks (CINE), 2015 Int'l Conf. on*. IEEE. 2015, S. 116–123. DOI: 10.1109/CINE.2015.31.
- [Put16] Andrew Putnam. „Accelerating Hyperscale Datacenter Services with FPGAs – Microsoft Catapult“. In: *Keynote, Seventh Int'l Symp. on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*. 2016.
- [Raj+08] Yamuna Rajasekhar, William V Kritikos, Andrew G Schmidt und Ron Sass. „Teaching FPGA system design via a remote laboratory facility“. In: *Field Programmable Logic and Applications (FPL), Int'l Conf. on*. IEEE. 2008, S. 687–690. DOI: 10.1109/FPL.2008.4630040.
- [Rao+16] Muzaffar Rao, Thomas Newe, Ian Grout und Avijit Mathur. „An FPGA-based reconfigurable IPsec AH core with efficient implementation of SHA-3 for high speed IoT applications“. In: *Security and Communication Networks* 9.16 (2016), S. 3282–3295. DOI: 10.1002/sec.1533.
- [Rav+11] Vignesh T Ravi, Michela Becchi, Gagan Agrawal und Srimat Chakradhar. „Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework“. In: *High performance distributed computing, Proc. of the 20th Int'l Symp. on*. ACM. 2011, S. 217–228. DOI: 10.1145/1996130.1996160.
- [Raz94] Rahul Razdan. „PRISC: Programmable reduced instruction set computers“. Dissertation. Harvard University, 1994. URL: <http://nrs.harvard.edu/urn-3:HUL.InstRepos:25620498>.
- [Rev+05] Xavier Revés, Vuk Marojevic, Ramon Ferrús und Antoni Gelonch. „FPGA's middleware for software defined radio applications“. In: *Field Programmable Logic and Applications, Int'l Conf. on*. IEEE. 2005, S. 598–601. DOI: 10.1109/FPL.2005.1515794.
- [RFG16] Jens Rettkowski, Konstantin Friesen und Diana Göringer. „RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs“. In: *ReConfigurable Computing and FPGAs (ReConFig), 2016 Int'l Conf. on*. IEEE. 2016, S. 1–8. DOI: 10.1109/ReConFig.2016.7857186.

- [Rog16] Bruce Rogers. „The Coming Data Avalanche – And how we'll handle it“. In: *Forbes Insights*. 2016.
- [Rou+04] Gaël Rovroy, F-X Standaert, J-J Quisquater und J-D Legat. „Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications“. In: *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. Int'l Conf. on*. Bd. 2. IEEE. 2004, S. 583–587. DOI: 10.1109/ITCC.2004.1286716.
- [Rus15] Patrick Russell. „Entwurf und Implementierung eines hochparallelen Black-Scholes Monte-Carlo-Simulators“. In: *Komplexpraktikum Massivparalleles Rechnen (MPR)*, Technische Universität Dresden (2015). Betreuer: Oliver Knodel, Hochschullehrer: Prof. Rainer G. Spallek.
- [Rus17] Ruschival, Thomas. *AES 128/192/256 (ECB) – Opencores*. Website. Online unter: https://opencores.org/project,avs_aes; abgerufen am 25. Julie. 2017.
- [RWW09] Ramaswamy Ramaswamy, Ning Weng und Tilman Wolf. „Analysis of network processing workloads“. In: *Journal of Systems Architecture* 55.10 (2009), S. 421–433.
- [Sal+10] Manuel Saldaña u. a. „MPI as a Programming Model for High-Performance Reconfigurable Computers.“ In: *TRETS* 3.4 (2010), S. 22–29.
- [San+14a] Oliver Sander, Steffen Baehr, Enno Luebbers, Timo Sandmann, Viet Vu Duy und Juergen Becker. „A flexible interface architecture for reconfigurable coprocessors in embedded multicore systems using pcie single-root i/o virtualization“. In: *Field-Programmable Technology (FPT), Int'l Conf. on*. IEEE. 2014, S. 223–226. DOI: 10.1109/FPT.2014.7082780.
- [San+14b] Oliver Sander u. a. „Hardware virtualization support for shared resources in mixed-criticality multicore systems“. In: *Design, Automation and Test in Europe Conf. and Exhibition (DATE), 2014*. IEEE. 2014, S. 1–6. DOI: 10.7873/DATE.2014.081.
- [Sas07] R. Sass et al. „Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing“. In: *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symp. on*. IEEE. 2007. DOI: 10.1109/FCCM.2007.62.
- [SB08] Hayden Kwok-Hay So und Robert Brodersen. „A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH“. In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.2 (2008), S. 14. DOI: 10.1145/1331331.1331338.
- [Sca11] Karen Scarfone. *Guide to security for full virtualization technologies*. DIANE Publishing, 2011.
- [Sch11] Mario Schölzel. „Fine-grained software-based self-repair of VLIW processors“. In: *2011 IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE. 2011, S. 41–49.
- [Sch12] A. G. Schmidt et al. „Investigation into scaling I/O bound streaming applications productively with an all-FPGA cluster“. In: *Parallel Computing* 38.8 (Aug. 2012).
- [Sch14] Mario Schölzel. *Hardware/Software-Codesign – Kapitel 1 - Einführung*. Vorlesung: Universität Potsdam, 2014.
- [SGH15] Fynn Schwiegelshohn, Lars Gierke und Michael Hübner. „FPGA based traffic sign detection for automotive camera systems“. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th Int'l Symp. on*. IEEE. 2015, S. 1–6. DOI: 10.1109/ReCoSoC.2015.7238089.
- [Sha+09] Gilad Shainer, Tong Liu, Jeffrey Layton und Joshua Mora. „Scheduling strategies for HPC as a service (HPCaaS)“. In: *Cluster Computing and Workshops, CLUSTER'09. Int'l Conf. on*. IEEE. 2009, S. 1–6. DOI: 10.1109/CLUSTR.2009.5289158.
- [Sha+10] Yi Shan, Bo Wang, Jing Yan, Yu Wang, Ningyi Xu und Huazhong Yang. „FPMR: MapReduce framework on FPGA“. In: *Proc. of the 18th annual ACM/SIGDA Int'l Symp. on Field programmable gate arrays*. ACM. 2010, S. 93–102. DOI: 10.1145/1723112.1723129.

Literaturverzeichnis

- [Sha+11] Upendra Sharma, Prashant Shenoy, Sambit Sahu und Anees Shaikh. „A cost-aware elasticity provisioning system for the cloud“. In: *Distributed Computing Systems (ICDCS), 2011 31st Int'l Symp. on*. IEEE. 2011, S. 559–570. DOI: 10.1109/ICDCS.2011.59.
- [Shi+12] Lin Shi, Hao Chen, Jianhua Sun und Kenli Li. „vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines“. In: *IEEE Trans. Computers* 61.6 (2012), S. 804–816. DOI: 10.1109/TC.2011.112.
- [Sho09] M. Showerman et al. „QP: A heterogeneous multi-accelerator cluster“. In: *Proc. of 10th LCI Int'l Conf. on High-Performance Clustered Computing*. 2009.
- [SK12] Gurdev Singh und Rakesh Kumar. „Availability metrics for cloud vibrant behaviour with benchmarks influence on diverse facets“. In: *Int'l Journal of Software Engineering & Applications* 3.1 (2012), S. 101.
- [Ska+13] Sam Skalicky, Christopher Wood, Marcin Lukowiak und Matthew Ryan. „High level synthesis: Where are we? A case study on matrix multiplication“. In: *ReConfigurable Computing and FPGAs (ReConFig), Int'l Conf. on* (2013). DOI: 10.1109/ReConFig.2013.6732298.
- [Sku12] Holger Skurk. „Speichervirtualisierung – Leitfaden Version 1“. In: *BITKOM: Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.* (2012).
- [SL05] Haoyu Song und John W Lockwood. „Efficient packet classification for network intrusion detection using FPGA“. In: *Proc. of the 2005 ACM/SIGDA 13th Int'l Symp. on Field-programmable gate arrays*. ACM. 2005, S. 238–245. DOI: 10.1145/1046192.1046223.
- [SN05a] James E Smith und Ravi Nair. „The architecture of virtual machines“. In: *Computer* 38.5 (2005), S. 32–38.
- [SN05b] James E. Smith und Ravi Nair. *Virtual machines - versatile platforms for systems and processes*. Elsevier, 2005. ISBN: 978-1-55860-910-5.
- [Sni98] Marc Snir. *MPI—the Complete Reference: The MPI core*. Bd. 1. MIT press, 1998.
- [Sol+07] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier und Larry Peterson. „Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors“. In: *ACM SIGOPS Operating Systems Review*. Bd. 41. 3. ACM. 2007, S. 275–287. DOI: 10.1145/1272996.1273025.
- [Sot+09] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente und Ian Foster. „Virtual infrastructure management in private and hybrid clouds“. In: *IEEE Internet computing* 13.5 (2009), S. 14–22. DOI: 10.1109/MIC.2009.119.
- [Spe05] Ralf Spenneberg. *Intrusion Detection und Prevention mit Snort 2 & Co: Einbrüche auf Linux-servern erkennen und verhindern*. Pearson Deutschland GmbH, 2005.
- [Spr+07] Henning Sprang, Timo Benk, Jaroslaw Zdrzalek und Ralph Dehner. *Xen: Virtualisierung unter Linux*. Open source press, 2007. ISBN: 978-3-937514-29-1.
- [SPV06] Dries Schellekens, Bart Preneel und Ingrid Verbauwhede. „FPGA vendor agnostic true random number generator“. In: *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE. 2006, S. 1–6. DOI: 10.1109/FPL.2006.311206.
- [SRC85] John A Stankovic, Krishivasan Ramamirtham und Shengchang Cheng. „Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems“. In: *IEEE Transactions on computers* 100.12 (1985), S. 1130–1143. DOI: 10.1109/TC.1985.6312211.
- [Sta+08] James Staten, Simon Yates, Frank E Gillett, Walid Saleh und Rachel A Dines. „Is cloud computing ready for the enterprise“. In: *Forrester Research, March 7* (2008).
- [Sto10] John E Stone et al. „OpenCL: A parallel programming standard for heterogeneous computing systems“. In: *Computing in science & engineering* 12.3 (2010).
- [Stu+15] Jeffrey Stuecheli, Bart Blaner, CR Johns und MS Siegel. „CAPI: A coherent accelerator processor interface“. In: *IBM Journal of Research and Development* 59.1 (2015), S. 7–1.
- [Sur13] Vivek Surabhi. „Designing with SR-IOV Capability of Xilinx Virtex-7 PCI Express Gen3 Integrated Block“. In: XAPP1177 (v1.0) November 15. 2013.
- [Suz+10] Jun Suzuki, Yoichi Hidaka, Junichi Higuchi, Teruyuki Baba, Nobuharu Kami und Takashi Yoshikawa. „Multi-root share of single-root I/O virtualization (SR-IOV) compliant PCI Express

- device". In: *High Performance Interconnects (HOTI), IEEE 18th Annual Symp. on*. IEEE. 2010, S. 25–31. DOI: 10.1109/HOTI.2010.21.
- [SW04] Uwe Schneider und Dieter Werner. *Taschenbuch der Informatik*. Fachbuchverlag, 2004.
- [Tan06] Andrew S. Tanenbaum. *Computerarchitektur - Strukturen, Konzepte, Grundlagen* (5. ed.) Pearson Education, 2006. ISBN: 978-3-8273-7151-5.
- [TB14] Andrew S Tanenbaum und Herbert Bos. *Modern operating systems*. Prentice Hall Press, 2014. ISBN: 0-13-359162-X.
- [TBG08] Xiang Tian, Khaled Benkrid und Xiaochen Gu. „High Performance Monte-Carlo Based Option Pricing on FPGAs“. In: *Engineering Letters* 16.3 (2008).
- [TEE12] Maha Tebaa, Saïd El Hajji und Abdellatif El Ghazi. „Homomorphic encryption applied to the cloud computing security“. In: *Proc. of the World Congress on Engineering*. Bd. 1. 2012, S. 4–6.
- [Tei13] Jürgen Teich. *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. Springer-Verlag, 2013.
- [The+11] Kostas Theoharoulis, Charalambos Antoniadis, Nikolaos Bellas und Christos D Antonopoulos. „Implementation and performance analysis of seal encryption on FPGA, GPU and multi-core processors“. In: *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual Int'l Symp. on*. IEEE. 2011, S. 65–68. DOI: 10.1109/FCCM.2011.33.
- [TM14] Stephen M Trimberger und Jason J Moore. „FPGA security: Motivations, features, and applications“. In: *Proc. of the IEEE* 102.8 (2014), S. 1248–1265. DOI: 10.1109/JPROC.2014.2331672.
- [TMT14] Qingshan Tang, Habib Mehrez und Matthieu Tuna. „Multi-fpga prototyping board issue: the fpga i/o bottleneck“. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 Int'l Conf. on*. IEEE. 2014, S. 207–214. DOI: 10.1109/SAMOS.2014.6893213.
- [Tod+05] Timothy J Todman, George A Constantinides, Steven JE Wilton, Oscar Mencer, Wayne Luk und Peter YK Cheung. „Reconfigurable computing: architectures and design methods“. In: *IEE Proceedings-Computers and Digital Techniques* 152.2 (2005), S. 193–207. DOI: 10.1049/ip-cdt:20045086.
- [Tre96] Nick Tredennick. „The Case for Reconfigurable Computing“. In: *Microprocessor Report* 10.10 (1996), S. 5.
- [Tri15] Stephen M Trimberger. „Three ages of FPGAs: a retrospective on the first thirty years of FPGA technology“. In: *Proc. of the IEEE* 103.3 (2015), S. 318–331. DOI: 10.1109/JPROC.2015.2392104.
- [Tru16] Trusted Computing Group. *Trusted Computing Group*. Online unter: <http://www.trustedcomputinggroup.org>; abgerufen am 5. Dezember 2016. 2016.
- [TS09] Cole Trapnell und Steven L Salzberg. „How to map billions of short reads onto genomes“. In: *Nature biotechnology* 27.5 (2009), S. 455–457.
- [Tso10] K.H. Tsoi et al. „Axel: a heterogeneous cluster with FPGAs and GPUs“. In: *Proc. of the 18th annual ACM/SIGDA Int'l Symp. on Field programmable gate arrays*. ACM. 2010. DOI: 10.1145/1723112.1723134.
- [Unn+10] Deepak Unnikrishnan u. a. „Scalable network virtualization using FPGAs“. In: *Proc. of the 18th annual ACM/SIGDA Int'l Symp. on Field programmable Gate Arrays*. ACM. 2010, S. 219–228. DOI: 10.1145/1723112.1723150.
- [V+11] Kizheppatt Vipin, Suhaib Fahmy u. a. „Efficient region allocation for adaptive partial reconfiguration“. In: *Field-Programmable Technology (FPT), Int'l Conf. on*. IEEE. 2011, S. 1–6. DOI: 10.1109/FPT.2011.6133248.
- [V+12] Kizheppatt Vipin, Suhaib Fahmy u. a. „A high speed open source controller for FPGA partial reconfiguration“. In: *Field-Programmable Technology (FPT), 2012 Int'l Conf. on*. IEEE. 2012, S. 61–66. DOI: 10.1109/FPT.2012.6412113.
- [Van+13] Wim Vanderbauwheide, Anton Frolov, Sai Rahul Chalamalasetti und Martin Margala. „A hybrid CPU-FPGA system for high throughput (10Gb/s) streaming document classification“.

- In: *SIGARCH Computer Architecture News* 41.5 (2013), S. 53–58. DOI: 10.1145/2641361.2641370.
- [VB13] Wim Vanderbauwhede und Khaled Benkrid. *High-Performance Computing Using FPGAs*. Springer, 2013.
- [Vip+13] Kizheppatt Vipin, Shanker Shreejith, Dulitha Gunasekera, Suhaib A Fahmy und Nachiket Kapre. „System-level FPGA device driver with high-level synthesis support“. In: *Field-Programmable Technology (FPT), Int'l Conf. on*. IEEE. 2013, S. 128–135. DOI: 10.1109/FPT.2013.6718342.
- [VJ10] Marten Van Dijk und Ari Juels. „On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing.“ In: *HotSec 10* (2010), S. 1–8.
- [VPB09] Christian Vecchiola, Suraj Pandey und Rajkumar Buyya. „High-performance cloud computing: A view of scientific applications“. In: *2009 10th Int'l Symp. on Pervasive Systems, Algorithms, and Networks*. IEEE. 2009, S. 4–16. DOI: 10.1109/I-SPAN.2009.150.
- [VPP14] Charalampos Vatsalakis, Kyriacos Papadimitriou und Dionisios Pnevmatikatos. „Enabling dynamically reconfigurable technologies in mid range computers through PCI express“. In: *HIPER Workshop on Reconfigurable Computing (WRC)*. 2014.
- [Waw+07] John Wawrzynek u. a. „RAMP: Research Accelerator for Multiple Processors“. In: *IEEE Micro* 27.2 (2007), S. 46–57. DOI: 10.1109/MM.2007.39.
- [WBP13] Wei Wang, Miodrag Bolic und Jonathan Parri. „pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment“. In: *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 Int'l Conf. on* (2013), S. 1–9. DOI: 10.1109/CODES-ISSS.2013.6658997.
- [Wee+15] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner und Andreas Herkersdorf. „Enabling FPGAs in Hyperscale Data Centers“. In: *Cloud and Big Data Computing (CBDCom), Int'l Conf. on*. IEEE. 2015. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-LoP.2015.199.
- [Wee+16] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner und Andreas Herkersdorf. „Disaggregated FPGAs: Network Performance Comparison against Bare-Metal Servers, Virtual Machines and Linux Containers“. In: *Cloud Computing Technology and Science (CloudCom), 2016 IEEE Int'l Conf. on*. IEEE. 2016, S. 9–17. DOI: 10.1109/CloudCom.2016.0018.
- [Wen+12] Xiaolong Wen, Gengjiang Gu, Qingchun Li, Yun Gao und Xuejie Zhang. „Comparison of open-source cloud management platforms: OpenStack and OpenNebula“. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th Int'l Conf. on*. IEEE. 2012, S. 2457–2461. DOI: 10.1109/FSKD.2012.6234218.
- [Wes+11] Stephen Weston, James Spooner, Jean-Tristan Marin, Oliver Pell und Oskar Mencer. „FPGAs speed the computation of complex credit derivatives“. In: *Xcelljournal* (2011), S. 18.
- [WIA13] Louis Woods, Zsolt István und Gustavo Alonso. „Hybrid FPGA-accelerated SQL query processing“. In: *Field Programmable Logic and Applications (FPL), 23rd Int'l Conf. on*. IEEE. 2013, S. 1–1. DOI: 10.1109/FPL.2013.6645619.
- [Wig+02] Andy Wigley, Mark Sutton, Stephen Wheelwright, Robert Burbidge und R Mccloud. *Microsoft. net compact framework: Core reference*. Microsoft Press, 2002.
- [Wil+10] Jason Williams, Chris Massie, Alan D George, Justin Richardson, Kunal Gosrani und Herman Lam. „Characterization of fixed and reconfigurable multi-core devices for application acceleration“. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 3.4 (2010), S. 19. DOI: 10.1145/1862648.1862649.
- [Wol+84] Gil Wolrich, Edward McLellan, Larry Harada, James Montanaro und Robert Yodlowski. „A high performance floating point coprocessor“. In: *IEEE Journal of Solid-State Circuits* 19.5 (1984), S. 690–696.
- [WWG08] Jian Wang, Kwame-Lante Wright und Kartik Gopalan. „XenLoop: a transparent high performance inter-vm network loopback“. In: *High performance distributed computing, Proc. of the 17th Int'l Symp. on*. ACM. 2008, S. 109–118. DOI: 10.1145/1383422.1383437.
- [Xav+13] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange und Cesar AF De Rose. „Performance evaluation of container-based virtualization for high perfor-

- mance computing environments". In: *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro Int'l Conf. on*. IEEE. 2013, S. 233–240. DOI: 10.1109/PDP.2013.41.
- [Xil10a] Xilinx Inc. *Getting Started with the Virtex-6 FPGA ML605 Embedded Kit*. UG730 (v1.1), 14. Juni. 2010.
- [Xil10b] Xilinx Inc. „LogiCORE IP XPS HWICAP - Data Sheet“. In: Xilinx Technical Report DS586 (v5.00a), 23. July. 2010.
- [Xil11] Xilinx Inc. *ML505/ML506/ML507 Evaluation Platform - User Guide*. UG347 (v3.1.2), 16. Mai. 2011.
- [Xil12a] Xilinx Inc. *7 Series FPGAs Memory Interface Solutions – User Guide*. UG586, 18. Januar. 2012.
- [Xil12b] Xilinx Inc. *LogiCORE IP Tri-Mode Ethernet MAC v5.2 – User Guide*. UG777, 18. Januar. 2012.
- [Xil16a] Xilinx Inc. *7 Series FPGAs Configurable Logic Block – User Guide*. UG474 (v1.8), 27. September. 2016.
- [Xil16b] Xilinx Inc. „7 Series FPGAs Configuration – User Guide“. In: UG470 (v1.11), 27. September. 2016.
- [Xil16c] Xilinx Inc. *7 Series FPGAs Data Sheet: Overview*. DS180 (v2.2). 2016.
- [Xil16d] Xilinx Inc. *7 Series FPGAs Memory Resources*. UG473 (v1.12), 27. September. 2016.
- [Xil16e] Xilinx Inc. *KC705 Evaluation Board for the Kintex-7 FPGA - User Guide*. UG810 (v1.7), 8. Julie. 2016.
- [Xil16f] Xilinx Inc. *VC707 Evaluation Board for the Virtex-7 FPGA - User Guide*. UG885 (v1.7.1), 12. August. 2016.
- [Xil16g] Xilinx Inc. *Vivado Design Suite v2016.4 User Guide – Release Notes, Installation, and Licensing*. UG973 (v2016.4), 30. November. 2016.
- [Xil16h] Xilinx Inc. *Xilinx Power Estimator (XPE)*. Online unter: <https://www.xilinx.com/products/technology/power/xpe.html>; abgerufen am 22. Dezember 2016. 2016.
- [Xil16i] Xilinx Inc. *Xilinx products - 7 series FPGAs DSP Solution*. Online unter: <https://www.xilinx.com/products/technology/dsp.html>; abgerufen am 8. Dezember 2016. 2016.
- [Xil16j] Xillybus Ltd., Haifa, Israel. *An FPGA IP core for easy DMA over PCIe with Windows and Linux*. Website. Online unter: <http://xillybus.com>; abgerufen am 28. Oktober. 2016.
- [Xil17a] Xilinx Inc. *7 Series FPGAs Data Sheet: Overview*. DS180 (v2.5), 1. August. 2017.
- [Xil17b] Xilinx Inc. *7 Series FPGAs Integrated Block for PCI Express v3.3 – LogiCORE IP Product Guide*. PG054, 5. April. 2017.
- [Xil17c] Xilinx Inc. *SDAccel Development Environment – Delivering GPU and CPU-like Programming Experience for Data Center Workload Acceleration*. Online unter: <https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html>; abgerufen am 10. April. 2017.
- [Xil17d] Xilinx Inc. *VCU1525 Reconfigurable Acceleration Platform – User Guide*. UG1268 (v1.0), 13. November. 2017.
- [Xil17e] Xilinx Inc. *Vivado Design Suite - HLx Editions*. Online unter: <https://www.xilinx.com/products/design-tools/vivado.html>; abgerufen am 26. Januar 2017. 2017.
- [Xil17f] Xilinx Inc. *Vivado Design Suite User Guide*. UG902 (v2017.1), 5. April. 2017.
- [Xil17g] Xilinx Inc. *Vivado Design Suite User Guide – Partial Reconfiguration*. UG909 (v2017.1), 5. April. 2017.
- [Xil17h] Xilinx Inc., Ed Hallett. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools) – XAPP1222 (v1.3) September 23, 2016*. Online unter: https://www.xilinx.com/support/documentation/application_notes/xapp1222-idf-for-7s-or-zynq-vivado.pdf; abgerufen am 14. Mai 2017. 2017.
- [XSS14] Lei Xu, Weidong Shi und Taeweon Suh. „PFC: Privacy Preserving FPGA Cloud-A Case Study of MapReduce“. In: *Cloud Computing (CLOUD), 7th Int'l Conf. on*. IEEE. 2014, S. 280–287. DOI: 10.1109/CLOUD.2014.46.

- [Y+12] Dong Yin, Ge Li u. a. „Scalable mapreduce framework on fpga accelerated commodity hardware“. In: *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 2012, S. 280–294. DOI: 10.1007/978-3-642-32686-8_26.
- [YKL15] Michael Xi Yue, Dirk Koch und Guy GF Lemieux. „Rapid overlay builder for Xilinx FPGAs“. In: *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual Int'l Symp. on*. IEEE. 2015, S. 17–20. DOI: 10.1109/FCCM.2015.48.
- [Yos+10] Masato Yoshimi, Yuri Nishikawa, Mitsunori Miki, Tomoyuki Hiroyasu, Hideharu Amano und Oskar Mencer. „A performance evaluation of CUBE: one-dimensional 512 FPGA cluster“. In: *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2010, S. 372–381. DOI: 10.1007/978-3-642-12133-3_36.
- [Yos10] M. Yoshimi et al. „A performance evaluation of CUBE: one-dimensional 512 FPGA cluster“. In: *Reconfigurable Computing: Architectures, Tools and Applications* (2010).
- [You+09] Cliff Young, Joseph A Bank, Ron O Dror, J P Grossman, John K Salmon und David E Shaw. „A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton“. In: *SC 2009* (2009). DOI: 10.1145/1654059.1654083.
- [ZCB10] Qi Zhang, Lu Cheng und Raouf Boutaba. „Cloud computing: state-of-the-art and research challenges“. In: *Journal of internet services and applications* 1.1 (2010), S. 7–18.
- [ZGR15] Xiaolan Zhang, John Griffin und Pankaj Rohatgi. *XVMSocket*. GitHub. Online unter: <https://github.com/skranjac/XVMSocket>; abgerufen am 3. Mai. 2015.
- [Zha+07] Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi und John Linwood Griffin. „XenSocket: A high-throughput interdomain transport for virtual machines“. In: *Middleware 2007*. Springer, 2007, S. 184–203.
- [Zha+16] Qian Zhao, Takuya Nakamichi, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga und Toshinori Sueyoshi. „hCODE: An open-source platform for FPGA accelerators“. In: *Field-Programmable Technology (FPT), 2016 Int'l Conf. on*. IEEE. 2016, S. 205–208. DOI: 10.1109/FPT.2016.7929534.
- [Zia+10] Dimitrios Ziakas, Allen Baum, Robert A Maddox und Robert J Safranek. „Intel® quickpath interconnect architectural features supporting scalable system architectures“. In: *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symp. on*. IEEE. 2010, S. 1–6. DOI: 10.1109/HOTI.2010.24.
- [Zim80] H. Zimmermann. „OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection“. In: *IEEE Transactions on Communications* 28.4 (Apr. 1980), S. 425–432. ISSN: 0090-6778. DOI: 10.1109/TCOM.1980.1094702.
- [TZT10] Hai Zhong, Kun Tao und Xuejie Zhang. „An approach to optimized resource scheduling algorithm for open-source cloud systems“. In: *2010 Fifth Annual ChinaGrid Conf.* IEEE. 2010, S. 124–129. DOI: 10.1109/ChinaGrid.2010.37.
- [ZZL13] Zibin Zheng, Jieming Zhu und Michael R Lyu. „Service-generated big data and big data-as-a-service: an overview“. In: *2013 IEEE Int'l Congress on Big Data*. IEEE. 2013. DOI: 10.1109/BigData.Congress.2013.60.

Übersicht eigener Veröffentlichungen im Kontext der Dissertation

- [KGS17] Oliver Knodel, Paul R. Gessler und Rainer G. Spallek. „Virtualizing Reconfigurable Hardware to Provide Scalability in Cloud Architectures“. In: *Reconfigurable Architectures, Tools and Applications, RECATA 2017, Rome, Italy, September 10 - 14, ISBN: 978-1-61208-585-2, CENICS 2017 Best Paper Award*. IARIA. 2017, S. 33–38.
- [Kno16a] Oliver Knodel. „Virtualisierung rekonfigurierbarer Hardware zur Hintergrundbeschleunigung und Erhöhung der Zugangssicherheit von Cloud-Anwendungen“. In: *Dresdner Arbeitstagung Schaltungs- und Systementwurf, DASS 2016, Cottbus, Germany, ISBN 978-3-8396-1046-6*. Fraunhofer Verlag. 2016.
- [Kno16b] Oliver Knodel. „Virtualisierung rekonfigurierbarer Hardware zur Steigerung der Rechenleistung und Sicherheit in einer flexiblen Cloud-Architektur“. In: *46. Jahrestagung der Gesellschaft für Informatik, INFORMATIK 2016, Lecture Notes in Informatics (LNI), Klagenfurt, Austria, September 26-30*. Gesellschaft für Informatik, Bonn. 2016, S. 1975–1980. ISBN: 978-3-88579-653-4.
- [KGS16] Oliver Knodel, Paul Genßler und Rainer Spallek. „Migration of long-running Tasks between Reconfigurable Resources using Virtualization“. In: *ACM SIGARCH Computer Architecture News Volume 44, HEART 2016, Hongkong, Hongkong, July 25-27, 2016*. Bd. 44. 4. ACM. 2016, S. 56–61. DOI: 10.1145/3039902.3039913.
- [KLS16] Oliver Knodel, Patrick Lehmann und Rainer G Spallek. „RC3E: Reconfigurable Accelerators in Data Centres and their Provision by Adapted Service Models“. In: *9th IEEE Int'l Conf. on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2*. IEEE. 2016, S. 19–26. DOI: 10.1109/CLOUD.2016.0013.
- [KS15a] Oliver Knodel und Rainer G. Spallek. „Computing Framework for Dynamic Integration of Reconfigurable Resources in a Cloud“. In: *2015 Euromicro Conf. on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28*. IEEE. 2015, S. 337–344. DOI: 10.1109/DSD.2015.37.
- [KS15b] Oliver Knodel und Rainer G. Spallek. „RC3E: Provision and Management of Reconfigurable Hardware Accelerators in a Cloud Environment“. In: *Second Int'l Workshop on FPGAs for Software Programmers, FSP, London, United Kingdom, 1. September 2015*. arXiv preprint. 2015. URL: <http://arxiv.org/abs/1508.06843>.
- [Kno14] Oliver Knodel. „Integration von FPGA-Ressourcen als Hardwarebeschleuniger und deren Bereitstellung sowie Verwaltung in einem Mehrbenutzersystem“. In: *Dresdner Arbeitstagung Schaltungs- und Systementwurf, DASS 2014, Dresden, Germany, April 29-30*. Fraunhofer Verlag. 2014.
- [Kno+14b] Oliver Knodel, Martin Zabel, Patrick Lehmann und Rainer G Spallek. „Educating hardware design—From boolean equations to massively parallel computing systems“. In: *IX Southern Conf. on Programmable Logic, SPL 2014, Nov. 5-7, Buenos Aires, Argentina*. IEEE. 2014, S. 1–6. DOI: 10.1109/SPL.2014.7002216.
- [Kno+13] Oliver Knodel, Andy Georgi, Patrick Lehmann, Wolfgang E. Nagel und Rainer G. Spallek. „Integration of a Highly Scalable, Multi-FPGA-Based Hardware Accelerator in Common Cluster Infrastructures“. In: *42nd Int'l Conf. on Parallel Processing, ICPP 2013, Lyon, France, October 1-4*. IEEE. 2013, S. 893–900. DOI: 10.1109/ICPP.2013.106.
- [KS13a] Oliver Knodel und Rainer Spallek. „FPGAs in der Cloud: Integration und Bereitstellung von rekonfigurierbaren Hardware-Ressourcen in einer Cloud-Infrastruktur“. In: *5. Workshop für Grid-, Cloud- und Big-Data-Technologien für Systementwurf und -analyse, Grid4Sys 2013, Dresden, Germany, Nov. 27 - 28, PARS-Mitteilungen 2014*. Bd. 31. 1. 2013, S. 103–107.
- [KS13b] Oliver Knodel und Rainer G. Spallek. „Integration of a multi-FPGA system in a common cluster environment“. In: *23rd Int'l Conf. on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4*. IEEE. 2013, S. 1–2. DOI: 10.1109/FPL.2013.6645612.

Anhang

A Grundlagen

A.1 Rekonfigurierbare Hardware

A.1.1 Grundaufbau

Die sich für eine Rekonfiguration ergebenden Grundbausteine eines modernen FPGAs sind in der Regel in einem Gitter angeordnet, wie in Abbildung A.1 dargestellt. Nach [KZH16] und [HM04] sind die Basiselemente, aus denen jeder FPGA aufgebaut ist, im Einzelnen:

Konfigurierbarer Logikblock (CLB): Die CLBs stellen die eigentlichen Basiselemente von FPGAs dar, in denen die logischen Verknüpfungen und die FlipFlops innerhalb der Schaltung realisiert werden. Die eingehenden Datenleitungen werden mit LUTs zu einer logischen Funktionen verknüpft, wie in Abbildung A.1(c) dargestellt. Ein CLB besteht bei modernen FPGAs typischerweise aus mehreren LUTs mit je bis zu sechs Eingängen und FlipFlop, wie in Abbildung A.1(a) dargestellt.

Konfigurationszellen: Um die CLBs zu konfigurieren werden die Inhalte der LUTs (bei sechs Eingängen hat der LUT-Speicher entsprechend 2^6 Zeilen), welche die logische Funktion darstellen sowie die Steuersignale der Multiplexer in SRAM-Zellen¹ gespeichert, wodurch der FPGA entsprechend für eine Anwendung konfiguriert und jederzeit *rekonfiguriert* werden kann. Die Größe einer Konfigurationsdatei (Bitstream) liegt bei modernen FPGAs wie dem Xilinx Virtex-7 XC7VX485T bei 20 MByte.

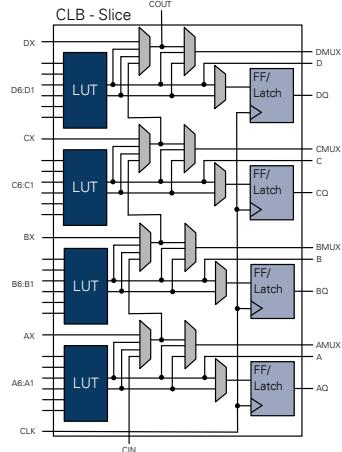
Verbindungsnetzwerk und CB: Das Verbindungsnetzwerk eines FPGAs, wie in Abbildung A.1(b) veranschaulicht, verbindet die gitterförmig angeordneten CLBs untereinander. Die Verbindungsleitungen können dabei über direkte lokale Verbindungen auf benachbarte CLBs beschränkt sein, aber auch global über den ganzen FPGA reichen. Spezielle Carry-Leitungen ermöglichen direkte Verbindungen zwischen den LUTs benachbarter CLBs. Das Verbindungsnetzwerk ist ein wesentlicher Teil, der für den kritischen Pfad in der Schaltung (zwischen zwei FlipFlops) und damit für die Taktrate verantwortlich ist². Das Konfigurieren der Verbindung zwischen den CLBs und dem Netzwerk erfolgt über Connection Block (CB)s, wie sie in Abbildung A.1(d) gezeigt sind. An den Kreuzungspunkten der Leitungen in Abbildung A.1(b) sind Programmable Switch Matrices (PSMs) angeordnet, um die Leitungen untereinander zu verschalten. Obwohl das Verbindungsnetzwerk mit seinen zahlreichen Multiplexern in einem FPGA den größten Teil der Ressourcen beansprucht, ist die Dokumentation verglichen mit der aller anderen Komponenten eher gering. Neben den Verbindungen zwischen den CLBs untereinander stellt das Verbindungsnetzwerk des Weiteren auch die Konnektivität zu den PSMs sicher.

¹Die Hersteller Intel (Altera), Lattice und Xilinx nutzen SRAM-basierte FPGAs und erreichen dadurch eine hohe Logikdichte und Leistungsfähigkeit. Solange der FPGA mit Energie versorgt wird, behält er entsprechend seine Funktion (volatile). Microsemi stellt FPGAs auf Basis von Flash-Electrically Erasable Programmable Read-Only Memory (EEPROM) Speicher her, welche weniger anfällig gegenüber Umwelteinflüssen wie Strahlung sind [KZH16].

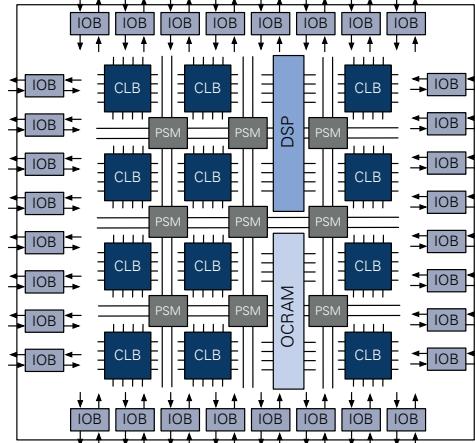
²Bei modernen FPGAs gehen 60-70% der Verzögerungszeiten der Signale auf das Verbindungsnetzwerk zurück [KZH16].

Anhang

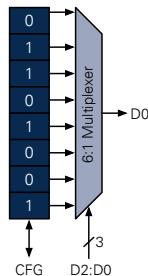
tivität mit der Außenwelt über konfigurierbare I/O Blöcke (IOBs) her. Spezielle I/O Pins zur Spannungsversorgung sowie Systemtakt und Reset sind dabei ebenso als Teil der FPGA-Infrastruktur vorhanden.



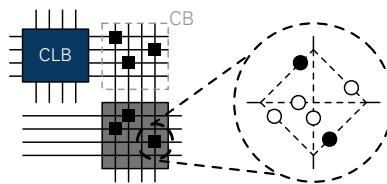
(a) Vereinfachte Darstellung eines modernen Xilinx-Slice mit vier 6er-LUTs und FlipFlops, sowie den dazugehörigen Multiplexern zur Konfiguration und Carry-Leitungen. Nach [Xil16a, S. 20].



(b) Aufbau eines vollständigen FPGAs mit Logik- (CLBs), I/O-, DSP- und dedizierten Speicherblöcken sowie den konfigurierbaren Leitungen mit den Verbindungsblöcken (PSMs), DSPs und OCRAM. Nach [HD08, S. 9].



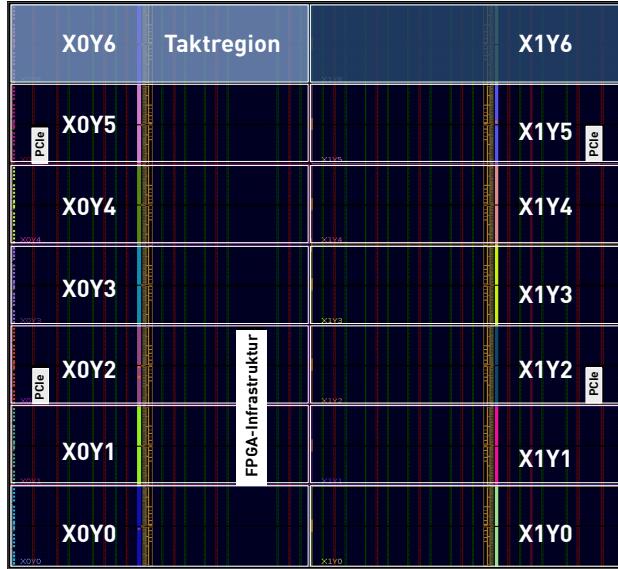
(c) 3er-LUT mit Wahrheitswertetabelle für ein XOR mit drei Eingängen, sowie serieller Leitung zur Konfiguration. Nach [HD08, S. 404].



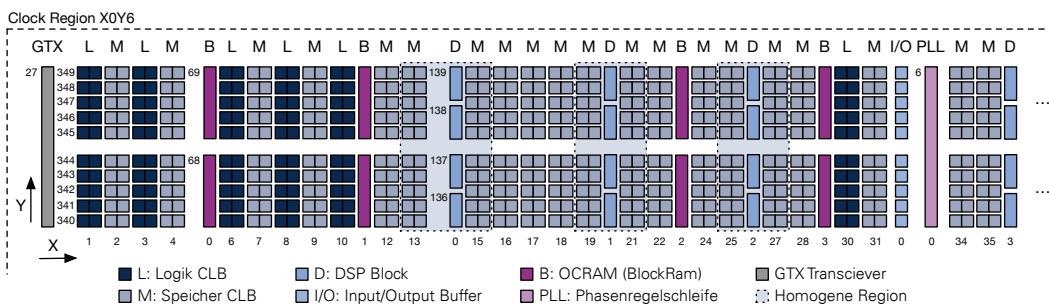
(d) Konfigurierbarer Verbindungsleitungen mit PSM, CLB und CB. Nach [HD08, S. 10].

Abbildung A.1: Bestandteile und Aufbau eines modernen FPGAs.

Den Aufbau eines modernen Xilinx Virtex-7 (XC7VX485T) FPGAs zeigt Abbildung A.2(a). Der FPGA ist in 14 Taktregionen unterteilt, welche die Takte innerhalb der Regionen an die unterschiedlichen Basiselemente über Mixed-Mode Clock Managers (MMCMs) oder Phase Locked Loops (PLLs) verteilen. Eine detaillierte Ansicht über die einzelnen Spalten, welche sich vertikal über den gesamten FPGA erstrecken ist in Abbildung A.2(b) dargestellt. Ein Rekonfigurationsframe ist dabei eine komplette Spalte innerhalb einer Taktregion.



(a) Gesamtübersicht mit 14 Taktröumen, PCIe-Endpunkten und der FPGA-Infrastruktur eines Virtex-7. Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 von Xilinx [Xil16g].



(b) Schematischer Ausschnitt ohne Routingressourcen innerhalb einer Taktregion eines Virtex-7 mit GTX Transceivern und PLL.

Abbildung A.2: Aufbau der Architektur eines Xilinx Virtex-7 XC7VX485T FPGAs [Xil16f].

A.1.2 GMAC-Leistung

Abbildung A.3 zeigt die Entwicklung der Giga Multiply-Accumulate (GMAC)-Leistung pro Watt für unterschiedliche Architekturen nach Degnan et al. [DMH16]. Des Weiteren veranschaulicht Abbildung A.3 den Trend der GMAC-Leistung und die Beobachtung von Koomey et al. [Koo+09], nach welcher sich die Rechenleistung pro Kilowattstunde innerhalb von jeweils 1,5 Jahren verdoppelt. Hierbei zeigt sich deutlich eine höhere Effizienz von sowohl FPGA als auch GPU gegenüber CPU.

A.1.3 Frameworks zur Kopplung von Prozessor und FPGA-Hardwarebeschleuniger

Die HLS-Werkzeuge erzeugen meist nur die reine Software-Funktion für den FPGA, sodass das Gerüst zur Kommunikation mit einem Host mittels PCI oder Ethernet ergänzt werden muss. Tabelle A.1 gibt eine Übersicht zu unterschiedlichen in der Literatur häufig eingesetzten Frameworks, in welche auch über HLS erzeugte Netzlisten eingebunden werden können. Die Systeme bieten dabei in der Regel nur einen Rahmen mit festen Schnittstellen.

Anhang

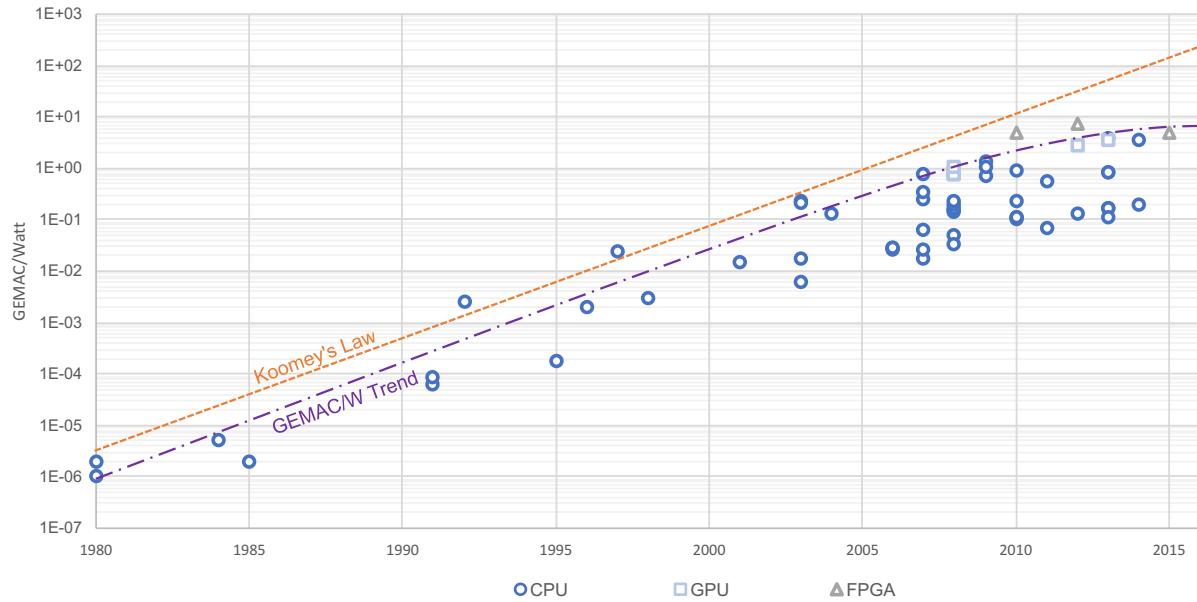


Abbildung A.3: Entwicklung der GEMAC-Leistung von CPU, FPGA und GPU. Nach [DMH16].

Tabelle A.1: Überblick zu FPGA-Beschleuniger Frameworks und deren Schnittstellen.

System	Schnittstellen			PR	Virtualisiert/ Mehrbenutzer	Bandbreite (MByte/s)	
	PCI	Ethernet	DDR			Up	Down
Riffa 2 [Jac+15]	✓	✗	✗	✗	✗	1.600	1.600
Xillybus [Xil16]	✓	✗	✗	✗	✗	3.500	3.500
Leap [Par+10]	✓	✗	✗	✗	✗	680 ^a	339 ^a
OpenCPI [MKS13]	✓	✓	✗	✗	✗	925	925
SIRC [Egu10]	✗	✓	✗	✗	✗	118	118
Vipin et al. [V+11]	✓	✓	✓	✓	✗	1.474	1.452
Zhao et al. [Zha+16]	✓	✗	✗	✓	✗	800	800

✓: vorhanden ?: nicht bekannt ✗: nicht vorhanden

^a Aus [Vip+13].

Viele der Arbeiten liefern auch Konzepte und Ansätze dazu, wie eigene Rechenkerne möglichst flexibel eingebettet werden können, wie beispielsweise die Arbeiten von Miller et al. [MKS13] oder Parashar et al. [Par+10]. Die Arbeit von Zhao et al. [Zha+16] nimmt dabei einen besonderen Stellenwert ein, da ein dynamisches Laden von Rechenkernen aus einer Datenbank in das bereitgestellte Grundgerüst auf dem FPGA erfolgt.

A.2 Gerätевirtualisierung allgemein

Im Folgenden werden unterschiedliche Kategorien oder Klassen von Geräten und deren charakteristische Virtualisierung auf der Basis exklusiver oder gemeinsamer Nutzung durch VMs aufgelistet.

Dedizierte Geräte

Eine besondere Kategorie sind Geräte, welche einer VM exklusiv zugeordnet werden und entsprechend eine längere Zeit benötigen, um an eine ander VM übergeben werden zu können. Der Zugang zu den Geräten erfolgt in der Regel ohne Virtualisierung durch direktes Durchreichen vom VMM, um die Privilegien (Zugriffsart, Speicherbereiche etc.) der neuen VM entsprechend anzupassen. Beispiele sind einfache Eingabegeräte [SN05b, S. 404].

Gemeinsame Geräte

Andere Geräte, wie beispielsweise Netzwerkkarten, können nicht einfach unter den VMs aufgeteilt werden, sondern werden geteilt, wozu der VMM eine Serialisierung der Anfragen nebenläufiger VMs durchführen muss. Bei einer Netzwerkkarte muss der VMM beispielsweise die internen Anfragen der VMs durch die reale, externe Netzwerkadresse ersetzen. Eingehende Pakete müssen wiederum an die virtuelle Netzwerkkarte der Ziel-VM weitergeleitet werden. Es wird also innerhalb des VMM eine Umsetzung der Adressen zum Beispiel durch *Network Address Translation (NAT)* durchgeführt [SN05b, S. 405]. Das Scheduling von Anfragen an I/O-Geräte innerhalb eines VMMs wurde zum Beispiel von Ongaro et al. in [OCR08] untersucht.

Partitionierte Geräte

Einige Geräte wie zum Beispiel Festplatten können in einzelne Partitionen unterteilt werden. Eine so entstandene virtuelle Festplatte wird von der VM behandelt wie eine normale Festplatte, um aber einen Zugriff zu emulieren, setzt der VMM die Adressparameter für die eigentliche reale Hardware, führt die Operation aus und sendet entsprechende Daten oder emulierte Statusinformationen der Festplatte zur VM zurück [SN05b, S. 405].

Nicht existente Geräte

Wie bereits im vorherigen Abschnitt angedeutet, müssen virtuelle Geräte nicht zwangsläufig real existieren. Ein Beispiel sind virtuelle Netzwerkkarten, die über ein virtuelles Netzwerk im VMM direkt eine Kommunikation zwischen den unterschiedlichen VMs aufbauen können. Somit kann der VMM als Router für ein virtuelles internes Netz mit virtuellen Interfaces zu den VMs auftreten [SN05b, S. 407].

A.2.1 Speicher-Virtualisierung

Speicher-Virtualisierung wird eingesetzt um den logischen Speicher vom physischen Speicher und von seinen Eigenschaften wie Größe, Adressierung und Verteilung zu abstrahieren. Für VMs erscheint der Speicher demnach virtuell, da die zugrundeliegenden physischen Geräte nicht mehr ersichtlich sind. Durch Speicher-Virtualisierung ist der Speicher für Nutzer nicht an physische Grenzen gebunden. Eine

Anhang

Erweiterung oder Umstrukturierung des physischen Speichers erfolgt für die Nutzer völlig intransparent. Der Vorteil besteht darin, dass der physische Speicher effektiver auf die vorhandenen Nutzer aufgeteilt und somit die Auslastung verbessert werden kann [Sku12]. Zur Speicher-Virtualisierung existieren unterschiedliche Ansätze, basierend auf der eigentlichen Position des Speichers im Gesamtsystem (Arbeitsspeicher, Festplatte etc.) und speziell im Datenpfad. Die Verfahren für Speicher im Netzwerk sind *Out of Band* mit einer Virtualisierung außerhalb des Datenpfades, *In Band* direkt innerhalb des Datenpfades des Speichergeräts oder im Host-System (VMM), welches die VMs bereitstellt [Sku12]. Die Virtualisierung von Festplatten im Host-System erfolgt über eine Abstraktionsebene im Betriebssystem wie beispielsweise im Logical Volume Management (LVM), welches es ermöglicht, dynamisch veränderbare Bereiche (Logical Volumes) aufzubauen, die sich über mehrere Festplatten erstrecken können [Has01].

Neben der Virtualisierung von Festplattenspeichern ist auch die klassische Virtualisierung von Arbeitsspeicher im Betriebssystem mit dynamischer Zuordnung von virtuellem Speicher zu physischem Speicher möglich. Techniken sind hierbei die statische Einteilung der Bereiche, Zugriffsrechte, Basis- und Grenzregister oder (mehrstufige) Seitentabellen [Tan06, S. 476].

A.2.2 Netzwerk-Virtualisierung

Eine Besonderheit stellt die Virtualisierung des Netzwerkverkehrs dar, die in jedem VMM notwendig ist, um den VMs eine Kommunikation mit der Außenwelt über das Netzwerk zu gestatten. Der VMM agiert dabei wie ein klassischer Router und ermöglicht in der Regel eine Netzwerkkadressübersetzung (NAT) über eine Netzwerkbrücke zwischen der physischen Netzwerkkarte, den entsprechenden virtuellen internen Netzwerken und den virtuellen Netzwerkkarten der Gast-VMs [Pic09, S. 404].

Neben dem Aufbau eines internen virtuellen Netzwerkes innerhalb eines VMMs ist es des Weiteren möglich, im Netzwerk eines Rechenzentrums eine virtuelle Umgebung in Form von dynamischen, voneinander gekapselten Virtual Local-Area Network (VLAN)s nach IEEE-Norm 803.1Q³ aufzubauen. Das Verfahren wird als *VLAN-Tagging*⁴ bezeichnet und die einzelnen Ethernet-Pakete werden mit VLAN-IDs versehen. Entsprechende 803.1Q fähige Netzwerkrouter bieten die Option, Pakete mehrerer VLANs über einen einzigen Port weiterzuleiten. Dadurch wird es möglich unterschiedliche virtuelle Netzwerke im VMM mit in das Netzwerk des Rechenzentrums einzugliedern [Pic09, S.404].

³Der Vorgänger des Standards war nicht dynamisch und die physischen Ports eines Switches waren immer genau einem VLAN zugeordnet.

⁴Erweiterung des Ethernet-Paket-Headers um 4 Byte, wovon 12 Bit eine VLAN-ID für das virtuelle Netzwerk darstellen [Pic09, S.404].

A.3 Cloud-Computing

A.3.1 Typische Cloud-Architektur und Komponenten

Eine typische Cloud-Architektur ist ein auf mehrere Rechenknoten verteiltes System mit lokalem Verbindungsnetzwerk (Multi-Tier Architecture). Ein oder mehrere Knoten übernehmen dabei die Kontrollfunktion und stellen die Instanzen der Virtuellen Maschinen auf den verfügbaren Rechenknoten einem Nutzer bereit. Für die größtmögliche Flexibilität ist ein verteiltes Dateisystem oder SAN für die zentrale Verwaltung von Betriebssystem-Images der Nutzer sowie der weiteren relevanten Daten verantwortlich.

Durch die zentrale Anforderung, (virtualisierte) *Ressourcen* auf unterschiedlichen Ebenen als Dienst einer breiten Masse an unterschiedlichsten Nutzern bereitzustellen, ergeben sich eine Reihe von speziellen Cloud-Komponenten, die in der Regel für die drei Kernanwendungen der Bereitstellung von Rechen-, Speicher- und Netzwerkinfrastruktur ausgelegt sind. In den letzten Jahren haben sich in diesem Bereich die Systeme OpenNebula [Ope16a] und OpenStack [Ope16d] etabliert.

OpenStack ist ein Cloud-Betriebssystem, welches ursprünglich von Rackspace sowie der National Aeronautics and Space Administration (NASA) entwickelt und im Jahr 2010 vorgestellt wurde. OpenStack setzt sich aus einer Vielzahl von Komponenten zusammen, welche unter anderem die Virtualisierung der Rechenressourcen (Nova-Compute), sowie die Bereitstellung von Speicher (Swift) und die Verwaltung von Images (Glance) übernehmen [Wen+12].

OpenNebula hingegen entstand bereits im Jahr 2005 als Forschungsprojekt mit einer ersten öffentlichen Bereitstellung im Jahr 2008 [Wen+12]. Der Schwerpunkt von OpenNebula liegt ebenfalls auf der Ebene der Infrastruktur (IaaS) und der Orchestrierung von virtualisierten Rechen-, Speicher- und Netzwerkressourcen, wobei das Monitoring dieser Ressourcen sowie Sicherheitsaspekte einen besonderen Stellenwert einnehmen. Der ursprüngliche Ansatz bestand in der Bereitstellung von Multi-Tier Diensten wie Computerclustern als Virtuelle Maschinen auf verteilten Infrastrukturen [Wen+12]. Es wird des Weiteren Wert auf die Standardisierung, Interoperabilität und Portabilität des Gesamtsystems gelegt, was die Nutzung unterschiedlichster Cloud-APIs und Hypervisoren ermöglicht. Die flexible Architektur erlaubt es außerdem, verschiedene Hard- und Softwarekombinationen in einem Rechenzentrum zu vereinen [Sot+09].

Im Folgenden werden die wichtigsten Komponenten, welche für die Bereitstellung einer Infrastruktur (IaaS) nach [Boh+11, S. 24] notwendig sind, kurz vorgestellt⁵, wobei OpenStack [Ope16c] als Referenz dient.

Verwaltung der Rechenressourcen: Die Ressourcenverwaltung stellt die Hauptkomponente einer Cloud zum Management der verteilten und virtualisierten Rechenressourcen dar. Typischer Bestandteil ist das Management des Hypervisors, welcher Virtuelle Maschinen auf den Rechenknoten bereitstellt. Eine wichtige Aufgabe stellt die Lastverteilung auf den physischen Rechenknoten sowie die Bereitstellung eines Zugangs zu diesen mit Hilfe des Netzwerk-Managers dar.

Lastverteilung der Ressourcen (Load Balancer): Die Lastverteilung ist eine wesentliche Komponente, da sie die Instanzen den eigentlichen physischen Rechenknoten nach unterschiedlichen Kriterien zuordnet. Bei OpenStack wird das Scheduling vom Netzwerk-Manager übernommen, der die Zuordnung der VM-Instanzen zu den physischen Knoten (sowie das Bereitstellen des Zuganges) ermöglicht. Hierbei wird zur Lastverteilung eine API angeboten, um die Integration von eigenen Algorithmen zu vereinfachen.

⁵Derartige Komponenten sind in ähnlicher Form ebenso in Amazons EC2 [Ama17b], sowie in OpenStack [Ope16c] vorhanden.

Netzwerk-Manager (Networking): Diese Komponente ist verantwortlich für die Konfiguration des (virtuellen) Netzwerks, um dem Cloud-Nutzer ein eigenes gekapseltes System mit IP-Adressen für die VMs (statisch oder dynamisch) durch VLANs bereitstellen zu können. Das dynamische Routing zu jeder beteiligten Ressource des Nutzers fällt ebenso in diesen Bereich.

Image-Datenbank, Block- und Objekt-Speicher (Storage): In einer Datenbank werden Images von Virtuellen Maschinen verwaltet. Diese werden als Vorlage genutzt, um Instanzen von virtuellen Maschinen, sowie verschiedene Blockspeicher für virtualisierte Speichermedien zu erzeugen. Eine weitere wichtige Komponente ist der Objekt-Speicher, welcher für die redundante Datenspeicherung verantwortlich ist und als Backend für Image- oder Blockspeicher eingesetzt werden kann.

Nutzerverwaltung (Identity): Die Nutzerverwaltung koordiniert das Authentifizierungs- und Rechte- system zwischen den beteiligten Komponenten sowie die Verwaltung der Nutzer über alle Instanzen, welche systemweit dem Nutzer zugeordnet sind.

Verwaltungsinterface (Dashboard): Die virtuelle Infrastruktur wird durch Administratoren und Nutzer über ein Webinterface verwaltet. Nutzer haben die Möglichkeit, ihre Plattform zu administrieren. Weitere Funktionen sind das Beobachten der Ressourcen sowie die Abrechnung der Nutzung der Cloud-Ressourcen.

Abrechnung (Billing/Accounting): Aufgrund der vielseitigen Möglichkeiten, die Nutzung der unterschiedlichen Dienste zu messen, sind eigenständige Komponenten notwendig, welche sämtliche verfügbaren Daten (Anzahl VM-Instanzen, Speicherbedarf, Rechenleistung, Netzwerkverkehr etc.) zur Nutzung der Ressourcen systemweit sammeln.

Überwachung der Qualität (SLA-Monitor oder Metering): Um eine hohe Qualität der Dienste zu garantieren und umgehend auf Probleme (Latenz, Bandbreiten, Rechenleistung etc.) eingehen zu können sowie den gesamten Systemzustand zu kontrollieren, ist eine Überwachung der Cloud erforderlich.

Aufbauend auf diesem Grundkonzept, welches die wesentliche Infrastruktur enthält (IaaS), kann eine komplexere Architektur realisiert werden, mittels derer die Plattformen (PaaS) für den Nutzer innerhalb dieser die Anwendungen und Dienste (SaaS) bereitgestellt werden.

A.3.2 Arten des Cloud-Betriebs

Des Weiteren können Cloud-Architekturen nach Mell und Grace [MG11] nach Art des Betriebs eingeordnet werden. Hierbei gibt es vier Arten, welche sich anhand Zugang und Verantwortlichkeit unterscheiden. Abbildung A.4 gibt einen Überblick.

Öffentliche Cloud (Public Cloud): Die Bereitstellung der Cloud-Dienste erfolgt über das Internet für eine Vielzahl von unterschiedlichen Kunden und somit für eine breite Öffentlichkeit. Beispiele sind die Clouds großer Unternehmen wie Amazon, Apple, Microsoft und Google. Der Nutzer der Cloud ist nicht der Eigentümer des physischen Systems und hat keine Kontrolle über die Speicherung der eigenen Daten oder eine Möglichkeit der Mitbestimmung bei Prozessabläufen. Das Modell bietet als Vorteil die größtmögliche Flexibilität und das Entfallen der Notwendigkeit einer eigenen Infrastruktur beim Kunden. Durch eine hohe Anzahl von Nutzern und die Ausnutzung der Elastizität werden die Betriebskosten für alle Beteiligten gesenkt, wobei die Kontrolle über die eigenen Daten und deren Sicherheit jedoch verloren gehen kann [Brä+12].

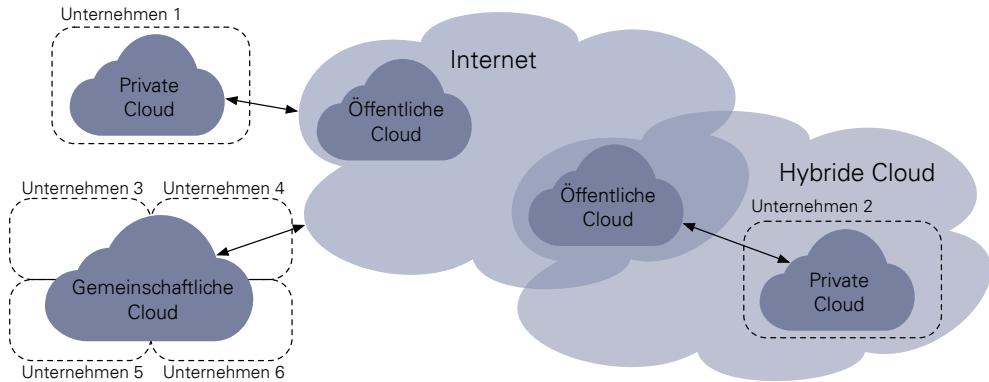


Abbildung A.4: Arten des Cloud-Betriebs. Die verantwortlichen Unternehmen für spezielle Cloud-Arten sind entsprechend gekennzeichnet. Für den Bereich des *Internets* liegt die Verantwortung außerhalb des Unternehmens, beziehungsweise Nutzers der Cloud. Nach [HV10, S. 17].

Private Cloud (Private Cloud): Im Gegensatz zur öffentlichen Cloud ist der Zugriff auf die private Cloud auf einen engeren Kreis von Nutzern, zum Beispiel die Mitarbeiter eines Unternehmens oder dessen Kunden und Geschäftspartnern, beschränkt. Die Organisation und Verwaltung der Cloud, sowie deren Standort fällt in den Verantwortungsbereich des Unternehmens, welches damit die vollständige Kontrolle über die Dienste und die Daten hat [Brä+12]. Die Sicherheit der Daten steigt im Vergleich zur öffentlichen Cloud, allerdings steigen auch die Betriebskosten der Bereitstellung der Dienste, und die Auslastung der Ressourcen und die Flexibilität verringert sich in der Regel deutlich. Für sicherheitskritische private Clouds in fremden Rechenzentren ist eine räumliche Trennung der Daten und der volle Zugriff zur Konfiguration auf die physischen Systeme erforderlich.

Gemeinschaftliche Cloud (Community Cloud): Die Cloud-Infrastruktur wird von einer Gemeinschaft (Unternehmen, Universitäten, Organisation oder Interessengruppe) betrieben und auch nur den Angehörigen dieser Gemeinschaft zur Verfügung gestellt. Die Anforderung an die Cloud in Bezug auf Sicherheit sind innerhalb der Gemeinschaft dieselbe [Brä+12]. Bereitgestellt und verwaltet wird die Cloud direkt von einem der teilnehmenden Partner. Die Dienste und Daten können typischerweise von den Mitgliedern kontrolliert werden.

Hybride Cloud (Hybrid Cloud): Hierbei handelt es sich um eine Mischform von öffentlicher und privater Cloud, welche durch eine Kooperation verschiedener Cloud-Anbieter oder durch *Cloud Bursting*⁶ entstehen kann. Die Kommunikation zwischen den unterschiedlichen Clouds erfolgt in diesem Fall durch eine spezielle Vermittlungsschicht, mit standardisierten Schnittstellen und Protokollen, welche sowohl Daten- als auch Anwendungsportabilität ermöglichen. Eine hybride Cloud ermöglicht eine hohe Flexibilität, führt allerdings zu Einschränkungen in der Sicherheit der Daten [Brä+12].

Das physische Rechenzentrum kann in jedem Fall von einem externen Dienstleister betrieben werden. Der Zugang zu den Ressourcen, die Möglichkeiten der Konfiguration, das Angebot an konkreten Diensten sowie die Anforderungen an die Sicherheit bestimmen letztendlich das konkrete Modell.

⁶Auslagern von Anwendungen aus einer Privaten in eine Öffentliche Cloud, sobald die benötigten Ressourcen die physisch vorhandenen übersteigen. Cloud Bursting ist notwendig, um innerhalb von Privaten Clouds eine größtmögliche Elastizität zu ermöglichen, wobei bei diesem Vorgehen Probleme hinsichtlich der Sicherheitsanforderungen entstehen können.

A.3.3 Weitere Beteiligte und Rollen

Aufbauend auf den klassischen Rollen haben sich noch weitere zusätzliche beziehungsweise optionale Rollen in den vergangenen Jahren herausgebildet und das ursprüngliche Modell erweitert:

Cloud-Ressourcenadministrator (Cloud Resource Administrator): Die Rolle bezeichnet den Administrator eines Dienstes. Der Nutzer kann in speziellen Fällen wie PaaS auch selbst für die Konfiguration der Ressourcen verantwortlich sein und ist in dem Fall der *Cloud-Ressourcenadministrator* [EPM13, S. 54].

Cloud-Prüfer (Cloud Auditor): Um ein hohes Maß an Sicherheit, den Schutz der Privatsphäre oder eine Leistungsmessung der Cloud eines Anbieters zu ermöglichen, werden externe *Cloud-Prüfer* eingesetzt [Boh+11, S. 8]. Cloud-Anbieter müssen hierzu allerdings dem Prüfer die Möglichkeit bieten, auf sicherheitsrelevante Komponenten zuzugreifen [Kun11].

Cloud-Vermittler (Cloud Broker): Durch die wachsende Komplexität von Cloud-Diensten werden externe *Vermittler* eingesetzt, um zwischen Diensten unterschiedlicher Cloud-Anbieter zu übermitteln, diese zusammen zu führen oder zu arbitrieren [Boh+11, S. 8].

Cloud-Carrier: Der Anbieter, welcher die Verbindung zwischen dem Anbieter der Cloud und dem Nutzer herstellt, wird als *Cloud-Carrier* bezeichnet. In der Regel handelt es sich dabei um ein Telekommunikationsunternehmen [Boh+11, S. 8]. Um die vereinbarten Datenraten und Latenzen garantieren zu können, welche für den Cloud-Dienst erforderlich sind, müssen in vielen Fällen Anbieter der Netzwerkinfrastruktur in das Gesamtkonzept einbezogen werden.

A.3.4 Typische Cloud-Anwendungen

Typische Cloud-Anwendungen wurden von Ferdman et al. in [Fer+12] umfassend untersucht und im *CloudSuite* Benchmark zusammengefasst. Der Fokus liegt dabei auf rechenintensiven Anwendungen, da diese ideal für eine Auslagerung auf spezielle Hardwarebeschleuniger geeignet sind, beziehungsweise auf Anwendungen, welche Daten in irgendeiner Form streamen und damit zusätzlich eine geringe Latenz erfordern. Aktuelle Anwendungen können dabei nach der auf dieser Grundlage aufbauenden Arbeit von Kachris et al. von 2016 [KS16] in zwei Gruppen unterteilt werden, welche sich in Komplexität der Verarbeitung und entsprechend der benötigten Zeit für die Berechnungen unterscheiden:

Batch-Verarbeitung (offline): Verarbeitung großer Datenmengen; dabei erfolgen typischerweise mehrere komplexe und zeitintensive Rechenschritte simultan. Die Anforderung an die Verarbeitung besteht daher primär in einem hohen Datendurchsatz und nicht in einer schnellen Antwortzeit und einer direkten Nutzerinteraktion (offline). Die primär genutzte Ressource ist der Prozessor, und nur in geringem Maße das Netzwerk. Die entscheidende Metrik beziehungsweise das Optimierungsziel ist bei diesen Anwendungen der Durchsatz. Beispiele für diese Kategorie sind die Analyse von Daten oder komplexe Simulationen. [KS16]

Stream-Verarbeitung (online): Hierbei handelt es sich um Anwendungen, welche keine zeitintensiven Berechnungen erfordern, sondern die Daten im Wesentlichen streamen, ohne sie lange Zeit im Speicher zu behalten und Berechnungen auszuführen. Die entscheidende Metrik ist primär eine geringe Latenz aufgrund einer stärkeren Interaktion mit einem Nutzer (online) und damit nur indirekt der einhergehende Datendurchsatz. Anwendungen in diesem Bereich sind Websuchen, einfache Datenbankanfragen beziehungsweise direktes Streaming von Medien oder einfachen Webservices. [KS16]

Anwendung	Beispiel	Anforderung		Ressourcen		
		Durchsatz	Latenz	Rechenleistung	CPU	Mem
Batch Verarbeitung	Daten Analyse	Mahout [Apa16b]	✓	✓	✓	✓
	In-Memory Analyse	SparkMlib [Apa16d]	✓	✓	✓	✓
	Graph Analyse	GraphX [Apa16c]	✓	✓	✓	✓
	Machine Learning	K-Means [Apa16b]	✓	✓	✓	✓
	Simulationen	Optionspreise [DK09]	✓	✓	✓	✓
Cloud Anwendungen	Daten Caching	Memcached [Mem16]	✓	✓	✓	✓
	Daten Bereitstellung	Cassandra [Apa16a]	✓	✓	✓	✓
	Medien- Streaming	Nginx [Ngi16]	✓	✓	✓	✓
	Datenbanken	Datenbanken [Ora16]	✓	✓	✓	✓
	Websuche	PageRank [Pag16]	✓	✓	✓	✓
Streaming Verarbeitung	Weservice	Nginx [Ngi16]	✓	✓	✓	✓

Abbildung A.5: Einteilung und Charakterisierung von typischen Cloud-Anwendungen mit Beispielen. Für jede Anwendung werden die primäre Anforderung sowie die am stärksten ausgelasteten Ressourcen gezeigt. Erweiterung von [KS16].

Anwendungen, welche auf einer Batch-Verarbeitung basieren, benötigen in den meisten Fällen primär Rechenleistung und den daran geknüpften Speicher. Anwendungen, welche auf einer Stream-Verarbeitung basieren, benötigen hingegen als Ressourcen verstärkt das Netzwerk und sind des Weiteren auf Latenz (und zur besseren Auslastung auf Durchsatz) optimiert, wie Tabelle A.5 für beispielhafte Anwendungen und deren Charakterisierung zeigt. Sicherheit ist dabei nicht als eigene Anwendung aufgenommen, da diese in den meisten Fällen nicht die primäre Anwendung ist, sondern lediglich eine Erweiterung darstellt.

B Zeitliche Einordnung

Um einen besseren Überblick zu relevanten Arbeiten und im Besonderen die zeitliche Einordnung zu ermöglichen, gibt Abbildung B.1 eine Übersicht zu den beiden Themenbereichen (a) FPGAs im Kontext des Cloud-Computings und (b) Virtualisierung rekonfigurierbarer Hardware. Des Weiteren werden die Arbeiten in Abbildung B.2 und Abbildung B.3 entsprechend klassifiziert.

1998	...	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
(a) Rekonfigurierbare Hardware im Cloud Kontext												
												Polgiani [Pog+16]
												Proano [PCC16]
												Kidane [KB16]
												Caulfield [Cau+]
										Weerasinghe [Wee+15]		Kachris [Kac+16b]
										Grigoras [Gri+14]	Fahmy [FVS15]	Asiatici [Asi+16]
										Byma [Bym+14]	Gazzano [Don+15]	Chow [Cho16]
										Chen [Che+14]	IBM [IBM15]	Knodel [KLS16]
										Knodel [Kno14]	Ghasemi [Gha15]	Orellana [OCC16]
						Madhavapeddy [MS11]	Yin [Y+12]	Kochberger [Koc+13]	Putnam [Put+14]	Ovtcharov [Ovt+15]		Kachris [Kac+16a]
						Shan [Sha+10]	Mondol [Mon11]	Eguro [EV12]	Lim [Lim+13]	Xu [XSS14]	Pöppelmann [Pop+15]	Gao [GS16]
(b) Virtualisierung von FPGAs												
										Byma [Bym+14]	Weerasinghe [Wee+15]	Kidane [KB16]
										Chen [Che+14]	Gazzano [Don+15]	Chow [Cho16]
								Zuma [BL12]	Koch [KBL13]	Nguyen [NK14]	Fahmy [FVS15]	Asiatici [Asi+16]
						So [SB08]	Yu [Yu+11]	Gonzalez [Gon+12]	Cheng [Che+13]	Agne [Agn+14]	Knodel [KS15]	Knodel [KLS16]
Fornaciari [FP98a]	Peck [Pec+06]	El-Araby [EG08]				Figuli [Fig+11]	Kirchgessner [Kir+12]	Wang [WBP13]	Eckert [Eck14]	Happe [HTK15]	Knodel [KGS16]	

Sicherheit und Anonymisierung in der Cloud	Hintergrund- beschleunigung	Rekonfigurierbare Hardware in der Cloud	Schwerpunkt der Verwaltung Heterogener Ressourcen
Maximierung der Auslastung und parallele Nutzer	Schnittstellen und Zugriff	Abstraktion von der physischen Architektur	Hardware Betriebssysteme und rekonfigurierbare Prozessoren

Abbildung B.1: Zeitachse mit Einordnung einer Auswahl relevanter Arbeiten zu den beiden Themen (a) FPGAs im Kontext des Cloud-Computings und (b) Virtualisierung rekonfigurierbarer Hardware.

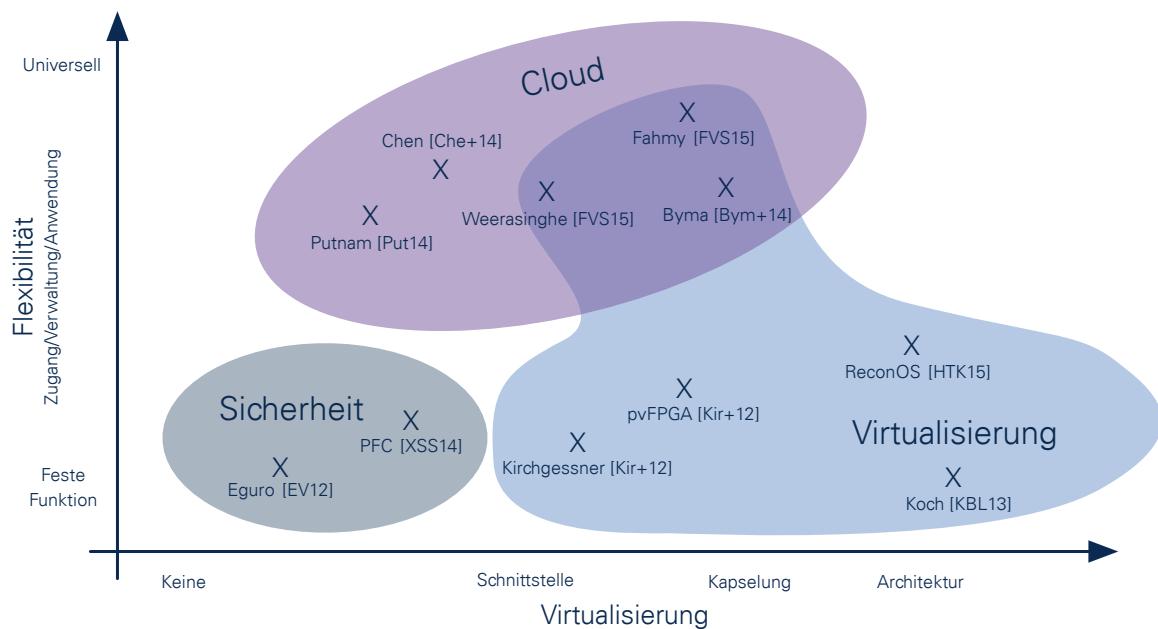


Abbildung B.2: Klassifikation relevanter Arbeiten.

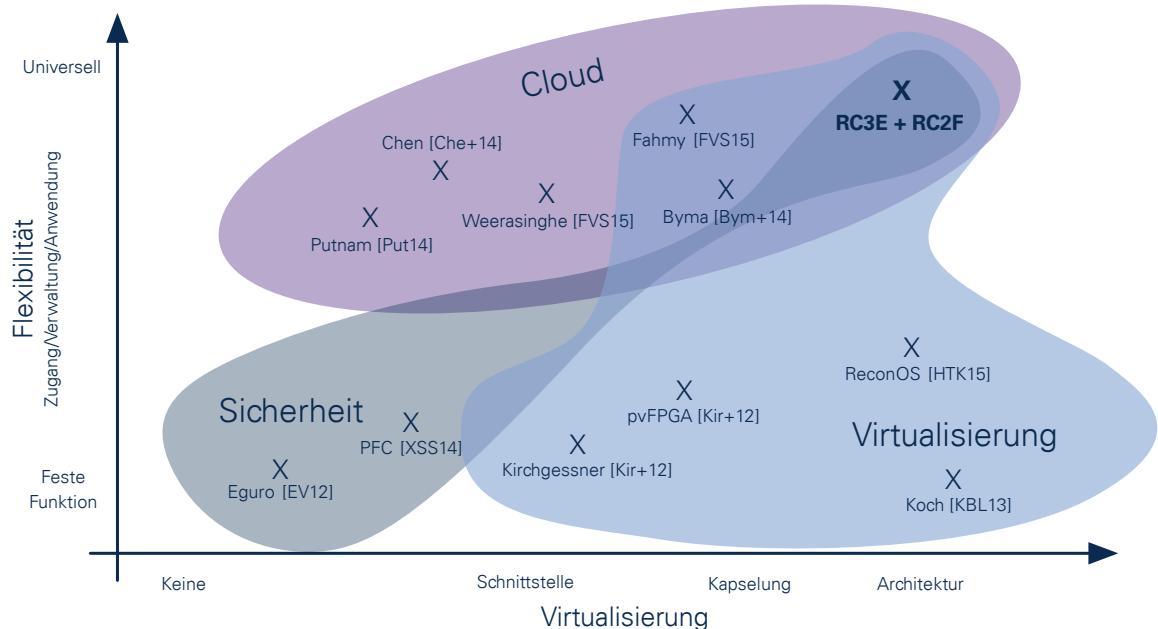


Abbildung B.3: Klassifikation relevanter Arbeiten und Einordnung des RC3E und RC2F.

C Prototypische Implementierung des RC2F

C.1 RC2F-Application Programming Interface

Um die Kommunikation mit den vFPGAs von Seiten des Host-Systems und die Konfiguration über die Konfigurationsspeicher Hypervisor Configuration Space (HCS) und vFPGA Control Unit (vCU) zu vereinfachen, wurde die RC2F-API entwickelt, welche zum Überblick in Tabelle C.1 dargestellt ist. Zusätzlich zeigt Listing C.1 einen Auszug aus der Header-Datei.

Listings C.1: Headerdatei der RC2F-API.

```

/*
 * API Header file
 *
 * Project: FPGA computing framework - rc2f
 * Author: Oliver Knodel (oliver.knodel@tu-dresden.de)
 * Created: 01.02.15
 */

#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define REG_LENGTH 32
#define CONFIG_REG_POS 16

#define CONFIG_LOOP_MASK 0x01
#define SET_LOOP 0x01
#define CLEAR_LOOP 0x00

#define CONFIG_RESET_MASK 0x10
#define CONFIG_CLK_MASK 0x80

#define DEVICE_FILE "/opt/rc2f/software/devices"

#define DEBUG 0

typedef struct
{
    char devID[6];
    int mem;
    int read;
    int write;
} device;

typedef struct str_thdata{
    device *dev;
    uint8_t *buf;
    uint32_t length;
    double runtime;
} thdata;

```

Anhang

```
int rc2f_read_stream(device* dev, uint8_t* buf, uint32_t len);
int rc2f_write_stream(device* dev, uint8_t* buf, uint32_t len);

void rc2f_read_thread(void *ptr);
void rc2f_write_thread(void *ptr);

int rc2f_reg_read(device* dev, int addr, char *buf, int len);
int rc2f_reg_write(device* dev, int addr, char *buf, int len);
int rc2f_reg_writeBit(device* dev, int addr, char data, char mask);

int rc2f_read_global_status(device* dev);
uint16_t rc2f_read_user_status(device* dev);
int rc2f_reset(device* dev);

int rc2f_finalize(device* dev);
device* rc2f_init_physical_device();

double rc2f_stream_performance(device *dev, uint32_t size, FILE* fd_measure);
void rc2f_performance_thread(void *ptr);
```

Tabelle C.1: Auszug aus der RC2F Host-API mit Funktionen welche (a) den Hypervisor-Konfigurationsspeicher (HCS) beeinflussen, (b) den vFPGA-Konfigurationsspeicher (vCS) verändern oder auslesen und (c) die eigentlichen Datentransfers durchführen.

(a) Konfiguration des FPGA Hypervisor	Authentifizierung erfolgt über den Host-Hypervisor
<code>rc2f_alloc(dev)</code> Allocate new/additional device or user region	
<code>dev[] = rc2f_get_dev()</code> Get al list of available/allocated devices	
<code>rc2f_program(dev, bitfile)</code> Reconfigure dev with bitfile	✓
<code>rc2f_reset_device(dev)</code> Reset of a whole FPGA with id dev	✓
<code>rc2f_status(dev)</code> Device status information (global config, user designs etc.)	
<code>rc2f_write_config(dev, adr, data)</code> W/rite data into global config space dev	✓
(b) vFPGA Status und Konfiguration	
<code>rc2f_user_list_cores()</code> Returns all available/allocated user cores	
<code>rc2f_user_init(uid)</code> Initialize specific user design uid	
<code>rc2f_vFPGA_design(uid, bitfile)</code> Partial reconfiguration of user design uid with bitfile	
<code>rc2f_user_cmd(uid, cmd)</code> Send command cmd (start, reset etc.) to user design uid	
<code>rc2f_user_status(uid)</code> Read status information from user design uid	
<code>rc2f_finalize(uid)</code> Close user design uid	
(c) Datentransfers zwischen VM und vFPGA	
<code>rc2f_read_stream(uid, data, length)</code> Send memory block or stream to kernel uid	
<code>rc2f_write_stream(uid, data, length)</code> Read memory block or stream to kernel uid	
<code>rc2f_mem_read(uid, addr, data, lenght)</code> Write at memory location in kernel uid	
<code>rc2f_mem_write(uid, addr, data, lenght)</code> Read from memory location in kernel uid	

C.2 RC2F-Infrastruktur

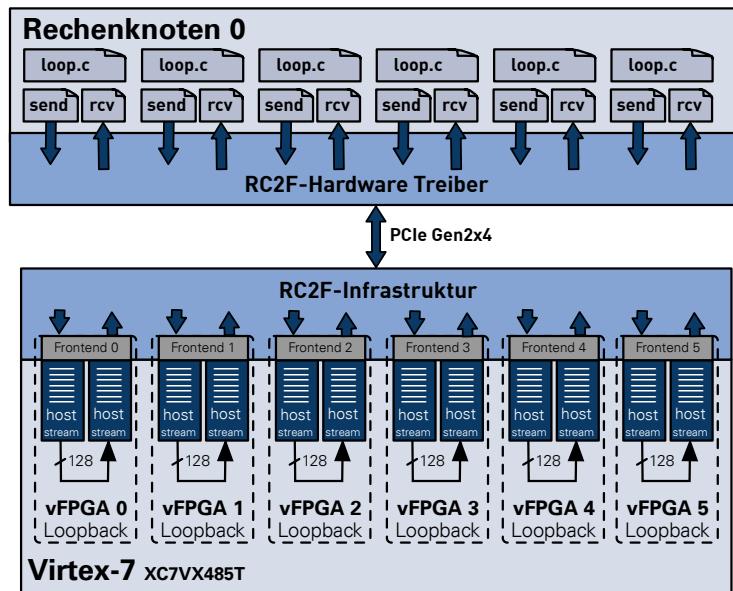


Abbildung C.1: Testanordnung zur Validierung der Kommunikation zwischen Host und vFPGAs.

C.3 Übertragung des RC2F auf einen Xilinx UltraScale+ FPGA

Die Einteilung eines Xilinx UltraScale+ FPGA (XCVU9P-L2FSGD2104E [Xil17d]) in die für die RC2F-Virtualisierung erforderlichen Bereiche ist in Abbildung C.2 dargestellt. Zusätzlich zeigt Tabelle C.2 die FPGA-Ressourcen innerhalb der unterschiedlichen Bereiche, sowie der homogenen und inhomogenen vFPGAs.

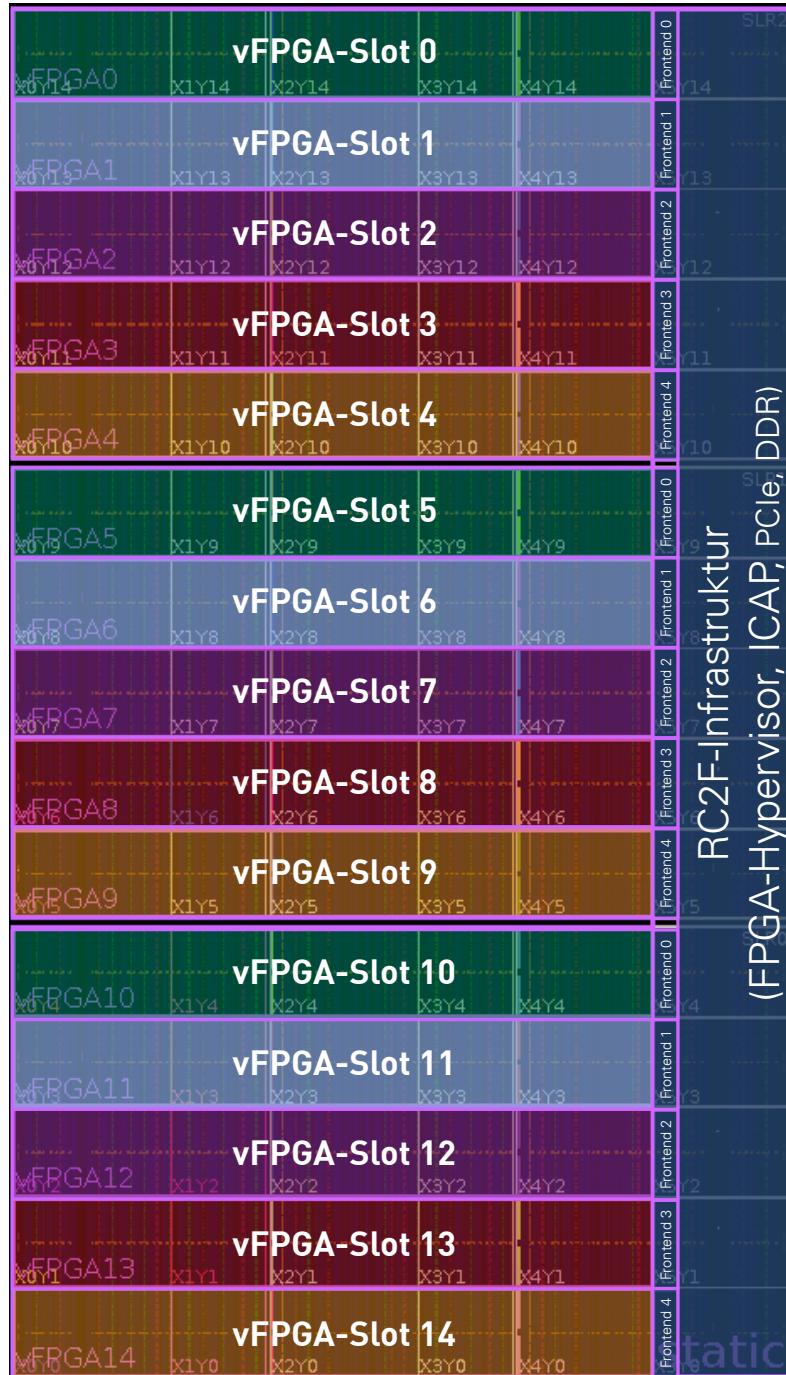
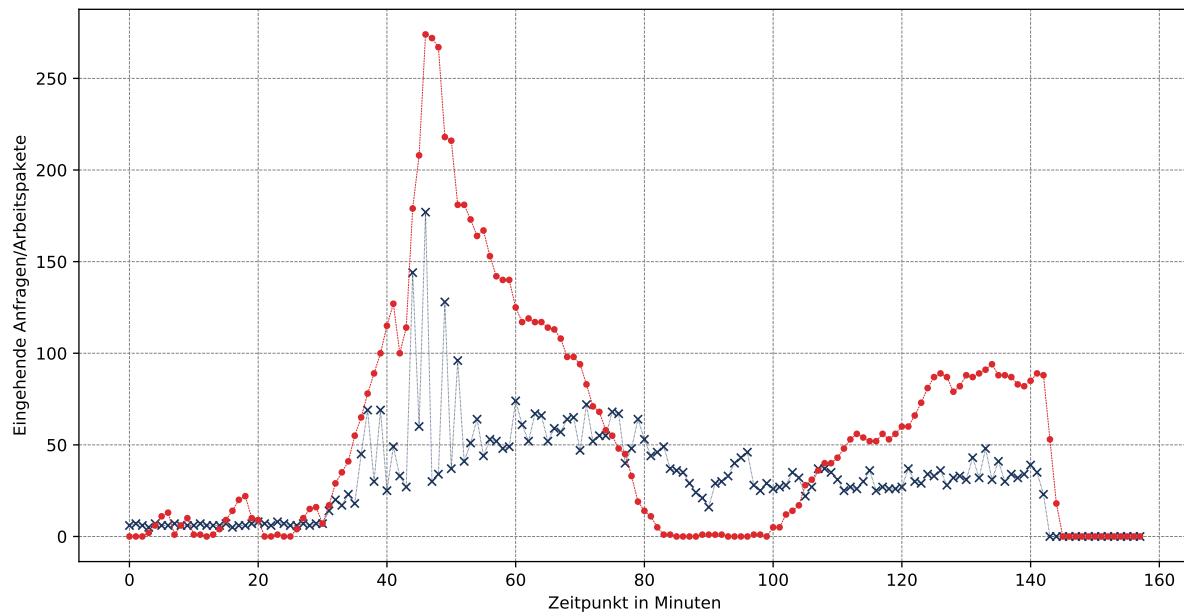


Abbildung C.2: Einteilung der Bereiche innerhalb der RC2F-Virtualisierung auf einem Xilinx UltraScale+ FPGA (XCVU9P). Erzeugt und ausgegeben mit der Vivado Design Suite 2016.4 von Xilinx [Xil16g].

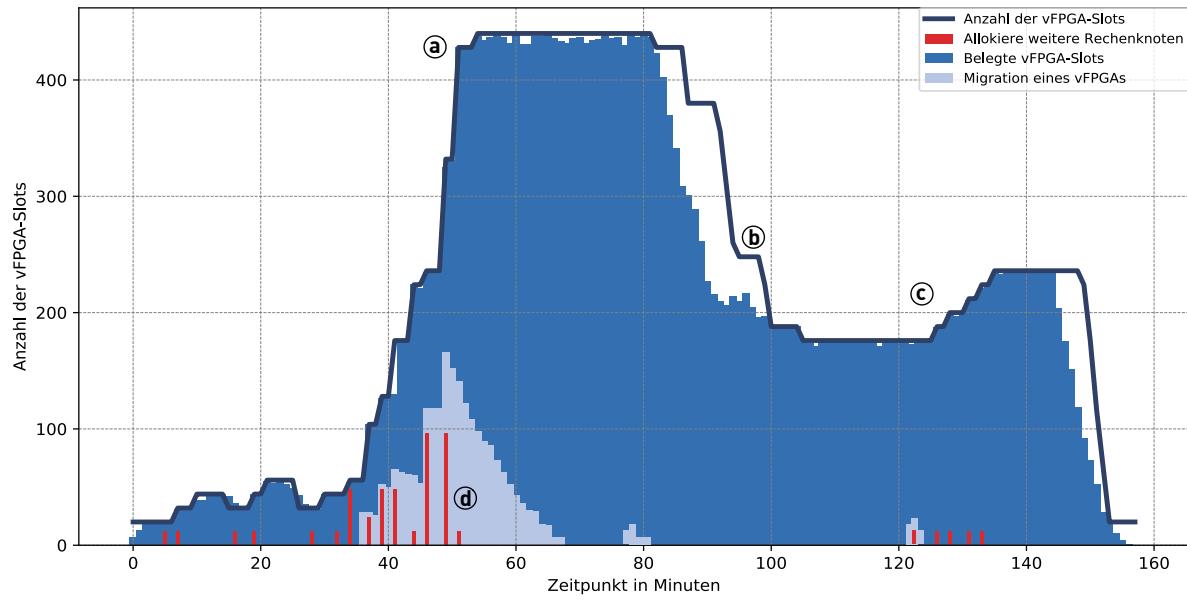
Tabelle C.2: FPGA-Ressourcen in den unterschiedlichen Bereichen für eine RC2F-Virtualisierung eines Xilinx UltraScale+ FPGA (XCVU9P).

	Inhomogene vFPGAs	Homogene vFPGAs	nicht nutzbar	15 x vFPGA	RC2F-Infrastruktur	Gesamte Ressourcen
CLB LUTs	68.160	63.360	4.800	950.400	188.640	1.182.240
LUT as Logic	68.160	63.360	4.800	950.400	188.640	1.182.240
LUT as Memory	35.040	30.240	4.800	453.600	95.040	591.840
CLB Registers	136.320	126.720	9.600	1.900.800	377.280	2.364.480
Register as FlipFlop	136.320	126.720	9.600	1.900.800	377.280	2.364.480
Register as Latch	136.320	126.720	9.600	1.900.800	377.280	2.364.480
CARRY8	8.520	7.920	600	118.800	23.580	147.780
F7 Muxes	34.080	31.680	2.400	475.200	94.320	591.120
F8 Muxes	17.040	15.840	1.200	237.600	47.160	295.560
F9 Muxes	8.520	7.920	600	118.800	23.580	147.780
CLB	8.520	7.920	600	118.800	23.580	147.780
CLBL	4.140	4.140	0	62.100	11.700	73.800
CLBM	4.380	3.780	600	56.700	11.880	73.980
LUT FlipFlop Pairs	68.160	63.360	4.800	950.400	188.640	1.182.240
Block RAM Tile	120	120	0	1.800	360	2.160
RAMB36/FIFO	120	120	0	1.800	360	2.160
RAMB18	240	240	0	3.600	720	4.320
URAM	64	64	0	960	720	960
DSPs	408	408	0	6.120	360	6.840
Bonded IOB	52	52	0	780	360	832
HPIOB_M	24	24	0	360	360	384
HPIOB_S	24	24	0	360	360	384
HPIOB_SNGL	4	4	0	60	6	64
HPIOBDIFFINBUF	48	48	0	720	14	720
HPIOBDIFFOUTBUF	48	48	0	720	14	720
BITSLICE_CONTROL	16	16	0	240	3	240
BITSLICE_RX_TX	104	104	0	1.560	3	1.560
BITSLICE_TX	16	16	0	240	11.520	240
RIU_OR	8	8	0	120	12	120
GLOBAL CLOCK BUFFERS	80	80	0	1.200	3	1.560
BUFGCE	48	48	0	720	3	720
BUFGCE_DIV	8	8	0	120	6	120
BUFG_GT	24	24	0	360	3	360

D RC3E-Simulator



(a) Eingehende Arbeitspakete und offene Anfragen innerhalb des Systems (Warteschlange für eingehende Arbeitspakete).



(b) Auslastung der allokierten Ressourcen.

Abbildung D.1: RC3E-Simulation des Lastszenarios (I) mit FPGA-Virtualisierung und zusätzlicher Migration.