Karlsruher Institut für Technologie

Software Design and Quality

Schnittstellenkonzept für Hardwaresimulationen zur Co-Simulation mit Software

Bachelorarbeit von

Mohammad Nour Dahi

an der Fakultät für Informatik
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

Erstgutachter:	Prof. Dr. Ralf H. Reussner
Zweitgutachter:	Prof. Dr.-Ing. Anne Koziolek
Betreuender Mitarbeiter:	M.Sc. Sebastian Weber
Zweiter betreuender Mitarbeiter:	PD Dr. rer. nat. Robert Heinrich

26. April 2023 – 28. August 2023

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **3**
2.1 Simulation . . . 3
2.2 Emulation . . . 3
2.3 Co-Simulation . . . 4
2.4 Hardwaresimulation . . . 4

3 Hardwaresimulationen **6**

4 Filterung von Hardwaresimulationen **13**
4.1 Kriterien zur Abgrenzung von Hardwaresimulationen . . . 13
4.2 Klassifizierbare Hardwaresimulationen . . . 16

5 Untersuchung von Hardwaresimulationen unter Ein- und Ausgabeparametern **18**
5.1 ESESC . . . 18
5.2 Flexus . . . 19
5.3 gem5 . . . 21
5.4 McSimA+ . . . 22
5.5 Multi2Sim . . . 23
5.6 OVPsim . . . 24
5.7 SESC . . . 26
5.8 Sniper . . . 26
5.9 ZSim++ . . . 28
5.10 Klassifizierung . . . 29

6 Allgemeinen Schnittstelle **32**
6.1 Entwicklung der allgemeinen Schnittstelle . . . 33
6.1.1 Eingabeparameter . . . 33
6.1.2 Hardwaresimulationsergebnisse . . . 45

7 Implementierung **48**
7.1 Eingabeprameter . . . 49
7.2 Hardwaresimulationsergebnisse . . . 51
7.3 Struktur des gesamten Softwareentwurfs für die allgemeine Schnittstelle . . . 55

8 Evaluation **59**
8.1 Klassifizierung . . . 60
8.1.1 Die Anwendbarkeit der Klassifizierung . . . 60

8.1.2 Schnittstellenähnlichkeiten bei Hardwaresimulationen einer Klasse . . 61
8.1.3 Hardwaresimulationen variieren in Zielsetzung 62
8.2 Allgemeine Schnittstelle 63
8.2.1 Abbildung von Schnittstellen evaluieren 63
8.2.2 Austausch-Aufwand durch allgemeine Schnittstelle verringert 64
8.2.3 Performance der allgemeinen Schnittstelle 65
8.2.4 Korrektheit der Hardwaresimulationsergebnisse 66

9 Verwandte Arbeiten **68**
9.1 Klassifizierung von Hardwaresimulation 68
9.1.1 Klassifizierung von Akram und Sawalha 68
9.1.2 Klassifizierung von Graef 70
9.2 Vergleich von Hardwaresimulationen 71
9.2.1 Vergleich von Hardwaresimulationen gemäß Urdén 71
9.2.2 Vergleich von Hardwaresimulationen nach Sanchez und Kozyrakis . . 72
9.2.3 Vergleich von Hardwaresimulationen von Nikolic u. a. 72
9.2.4 Vergleich von Hardwaresimulationen nach Alves u. a. 72
9.2.5 Vergleich von Hardwaresimulationen von Schönwetter 73

10 Zusammenfassung und Ausblick **74**

11 Abkürzungsverzeichnis **75**

Literatur **76**

Abbildungsverzeichnis

6.1 Allgemeine Schnittstelle zur Klasse von Hardwaresimulationen 32

7.1 Klassendiagramm für allgemeine Schnittstelle der Hardwaresimulation 48
7.2 Erstellen von Eingabeparameterdateien 49
7.3 Abbildung von 2 Input.json-Parametern in Sniper/ ZSim-Konfigurationsdateien 50
7.4 Ausgabeparameter 51
7.5 Darstellung von 3 Output.json-Parametern aus den Ergebnissen der ZSim-Hardwaresimulationen 52
7.6 Darstellung von 3 Output.json-Parametern aus den Ergebnissen der gem5-Hardwaresimulationen 54
7.7 Struktur des gesamten Softwareentwurfs für die allgemeine Schnittstelle . . . 55
7.8 Prozessablauf der allgemeinen Schnittstelle mit Aktivitätsdiagramm 57

8.1 Vergleich der Sniper-Hardwaresimulation mit und ohne die allgemeine Schnittstelle 67

Tabellenverzeichnis

3.1 Ein Überblick über die Hardwaresimulationen von Akram und Sawalha [7] mit neuen kursiv markierten Einträgen aus [28] und [71] hervorgehoben 11

4.1 Abgrenzung von Hardwaresimulationen 15
4.2 Übersicht von Hardwaresimulationen nach Abgrenzungen 17

5.1 Parameter nach [9] für die ESESC-Hardwaresimulation 19
5.2 Parameter nach für die Flexus-Hardwaresimulation 20
5.3 Parameter für die gem5-Hardwaresimulation 21
5.4 Parameter [5] für die McSimA+ 22
5.5 Parameter nach [76] für die Multi2Sim-Hardwaresimulation 23
5.6 Beschreibung der Parameter für die OVPsim-Hardwaresimulation 25
5.7 Parameter nach [13] für die SESC-Hardwaresimulation 26
5.8 Parameter [18] für die Sniper-Hardwaresimulation 27
5.9 Parameter [67] für die ZSim++ 29
5.10 Klassen von Hardwaresimulationen 29

6.1 Allgemeine Schnittstelle Parameter für zweite Klasse 34
6.2 Standardwert für die Parameter in ZSim 37
6.3 Standardwert für die Parameter in Sniper 41
6.4 Standardwert für die Parameter in gem5 45
6.5 Allgemeine Schnittstelle Hardwaresimulationsergebnis für zweite Klasse . . . 46

8.1 GQM-Plan 59
8.2 Ungeeignete Eigenschaften und dazugehörige Anzahl der Hardwaresimulationen 60
8.3 Anzahl der Hardwaresimulationen, die auf die Klassifizierung angewendet werden 61
8.4 Anzahl der Eingabeparameter 61
8.5 Anzahl der Hardwaresimulationsergebnisse 62
8.6 Anzahl der Paramter 62
8.7 Mehr mögliche Konfigurationen für einen Parameter, die von der allgemeinen Schnittstelle nicht unterstützt werden 63
8.8 Anzahl der Parameteränderung von gem5 zu ZSim 64
8.9 Gesamtsumme der Änderungen beim Wechsel von gem5 zu ZSim 65
8.10 Anzahl der Parameteränderung von ZSim zu gem5 65
8.11 Gesamtsumme der Änderungen beim Wechsel von ZSim zu gem5 65
8.12 Zeit für Abbildung und Ausführung der allgemeinen Schnittstelle 66

1 Einleitung

Durch Modellierung und Simulation können Systeme analysiert werden, ohne sie physisch zu erzeugen. Dies kann Zeit und Kosten sparen sowie die Möglichkeit bieten, verschiedene Szenarien zu testen und zu optimieren. Die Co-Simulation erlaubt die Kombination mehrerer Simulationen, die jeweils spezifische Domänen des Simulationsprozesses abdecken. Wenn beispielsweise ein autonomes Fahrsystem simuliert wird, können separate Simulationen für Fahrzeugdynamik, Sensorik, Umgebung und Verkehrssteuerung verwendet werden. Durch die Anwendung von Co-Simulation können die Interaktionen zwischen diesen Domänen modelliert werden, da ein einzelner Simulator zu komplex wäre, um alle Aspekte des autonomen Fahrsystems abzudecken. Des Weiteren bedarf es keiner Änderungen an den beteiligten Simulationen, wenn Änderungen in einer bereits existierenden Simulation vorgenommen werden. Stattdessen reicht es aus, eine Änderung an der Co-Simulation vorzunehmen.

Hardwaresimulationen dienen dazu, die Hardware zu simulieren und somit das Verhalten der Software auf der Hardware zu testen. Unabhängig von der exakten Art der Simulation oder Co-Simulation lässt sich allgemein feststellen: Je umfangreicher eine Simulation durchgeführt wird, desto länger dauert üblicherweise der Vorgang. Diese Zeitdauer korreliert direkt mit der Präzision der erzielten Ergebnisse. Beim Testen von Software oder bei einer Co-Simulation, die auf Hardware läuft, entsteht bei jeder Simulation ein Zielkonflikt zwischen Genauigkeit und Geschwindigkeit. Es gibt verschiedene Hardwaresimulationen zur Auswahl, die eine höhere Genauigkeit bieten, aber längere Ausführungszeiten erfordern. Wenn jedoch die Geschwindigkeit der Co-Simulation von größter Bedeutung ist, wählt man eine Simulation, die zwar geringere Genauigkeit bietet, aber schneller ausgeführt werden kann. Je nach Zielsetzung erfordert die Co-Simulation unterschiedliche Hardwaresimulationen. Ein Austausch von Hardwaresimulationen kann jedoch aufwändig sein und Anpassungen an der Co-Simulation zu erfordern, was zu einer Verschlechterung von Stabilität und Wartbarkeit der Co-Simulation führt.

Daher ist das Ziel dieser Arbeit die Entwicklung einer allgemeinen Schnittstelle für Hardwaresimulationen, um den Austausch von Hardwaresimulationen zu erleichtern. Diese allgemeine Schnittstelle erfordert keine Anpassungen an der Co-Simulation, was wiederum die Stabilität und Wartbarkeit der Co-Simulation verbessern kann.

Im Verlauf der Recherche zu dieser Arbeit konnte keine bestehende allgemeine Schnittstelle für Hardwaresimulationen oder eine Gruppe davon gefunden werden. Es ist nicht möglich, eine allgemeine Schnittstelle für die gesamte Bandbreite aller Hardwaresimulationen bereitzustellen. Daher wird sich im Rahmen dieser Arbeit auf einzelne Klassen beschränkt. Aus diesem Grund wird eine Klassifizierung der Hardwaresimulationen vorgenommen, um ähnliche Hardwaresimulationen zusammenzufassen und für eine Klasse eine allgemeine Schnittstelle zu entwickeln. Durch die Entwicklung einer Klassifizierung als allgemeines Verfahren wird die Herstellung von allgemeinen Schnittstellen vereinfacht. Es wird untersucht, ob diese allgemeine Schnittstelle auf verschiedene Klassen von Hardwaresimulationen anwendbar gemacht werden kann und ob sie für alle Hardwaresimulationen geeignet ist. Außerdem ermöglicht die Klassifizierung die

Identifizierung gemeinsamer Funktionen und Eigenschaften, um eine allgemeine Schnittstelle für alle Hardwaresimulationen in dieser Klasse zu definieren.

Im Rahmen dieser Bachelorarbeit soll der Fokus darauf gelegt werden, eine allgemeine Schnittstelle zu definieren, die es ermöglicht, dass Hardwaresimulationen in derselben Klasse einfach ausgetauscht werden können. Konkret wird in der Arbeit Folgendes behandelt:

- Vorstellung verschiedener Hardwaresimulationen anhand dieser Bachelorarbeit.
- Auflistung der Eigenschaften von Hardwaresimulationen sowie sorgfältige Berücksichtigung derjenigen Eigenschaften, die Einfluss auf die mögliche Schnittstelle haben.
- Durchführung einer Klassifizierung basierend auf den Eigenschaften von Hardwaresimulationen.
- Eine allgemeine Schnittstelle für eine Klasse soll definiert werden.
- Implementierung der Abbildung der allgemeinen Schnittstelle auf die individuellen Schnittstellen der Hardwaresimulationen.
- Entwicklung eines generellen Vorgehens zur Ableitung einer Schnittstelle für eine Klasse von Hardwaresimulationen.

In Kapitel 2 werden die grundlegenden Konzepte für die Bachelorarbeit erläutert, während Kapitel 3 verschiedene Hardwaresimulationen präsentiert. In Kapitel 4 wird ein Schema zur Kategorisierung von Hardwaresimulationen entwickelt, und in Kapitel 5 werden die Hardwaresimulationen unter verschiedenen Ein- und Ausgabeparametern untersucht. Kapitel 6 widmet sich der Definition der allgemeinen Schnittstelle für die Hardwaresimulation, während Kapitel 7 die Implementierung der allgemeinen Schnittstelle und die Anbindung der Hardwaresimulation daran beschreibt. Die Korrektheit und Performance der Schnittstellenabbildung wird in Kapitel 8 evaluiert, während Kapitel 9 verwandte Arbeiten behandelt. Schließlich fasst Kapitel 10 die Ergebnisse und Erkenntnisse der Arbeit zusammen.

2 Grundlagen

Dieses Kapitel beschreibt die Grundlagen, die für das Verständnis der nachfolgenden Kapitel erforderlich sind. Es erläutert die Begriffe Simulation, Emulation, Co-Simulation und Hardwaresimulation. Schließlich werden zwei Beispiele von Hardwaresimulationen vorgestellt.

2.1 Simulation

Ein **Modell** ist nach Ludewig in [48] eine vereinfachte und abstrakte Darstellung eines Teils der Realität, die verwendet wird, um bestimmte Aspekte des Urbilds zu untersuchen. Es dient als Werkzeug zur Analyse, Prognose und Entscheidungsfindung. Ein Modell besitzt nach Stachowiak in [73] zwangsläufig drei Eigenschaften, ohne die es kein Modell wäre.
Das **Abbildungsmerkmal** besagt, dass es ein Original geben muss, zu dem das Modell eine Abbildung darstellt. Ohne ein Original ist das Modell bedeutungslos.
Das **Verkürzungsmerkmal** besagt, dass das Modell nur eine Teilmenge der Attribute des Originals wiedergibt und nicht alle.
Das **Pragmatische Merkmal** besagt, dass das Modell in bestimmten Situationen und in Bezug auf spezifische Fragestellungen das Original ersetzen kann.

Simulation ist nach Mattern [50] das Nachbilden von Prozessen oder Systemen in einem experimentierfähigen Modell, um deren Verhalten und Eigenschaften zu untersuchen und Erkenntnisse zu gewinnen, die auf die Wirklichkeit übertragbar sind. Außerdem gibt es zwei Klassen von Modellen, die sich unterscheiden.
Analytische Modelle beschreiben das Verhalten von dynamischen Systemen mittels einer Menge gekoppelter Gleichungen. Diese Gleichungen basieren auf fundierten wissenschaftlichen Theorien und ermöglichen es, das Verhalten des realen Systems zu erklären und zu berechnen. Im Gegensatz dazu, verwenden **deskriptive Modelle** eine Vielzahl von zustandsabhängigen Simulationskomponenten, die miteinander interagieren, um das reale System zu beschreiben. Diese Art von Modellen ermöglicht es, Erkenntnisse durch Experimente an einer künstlichen Welt im Computer zu gewinnen. Dazu wird der aktuelle Zustand des Systems benötigt, der eine Reihe von Variablen enthält, die das System zu einem bestimmten Zeitpunkt beschreiben.

Durch die Simulation kann man Vorhersagen über das Verhalten des Systems treffen und verschiedene Szenarien testen, ohne das eigentliche System zu beeinflussen. Die Simulation ist in vielen Bereichen ein wichtiges Werkzeug, so auch bei Strömungs- oder Materialsimulationen.

2.2 Emulation

Emulation wird nach McGregor [51] verwendet, um Systeme oder Umgebungen nachzubilden, um bestimmte Funktionen auszuführen oder das Verhalten des Originals zu analysieren.

In diesem Zusammenhang kann ein funktionaler Aspekt des Modells durch einen Teil des realen Systems ersetzt werden, um Modell und Realität in Übereinstimmung zu bringen. Es gibt jedoch auch eine umgekehrte Definition, bei der ein Modell verwendet wird, um einen Teil des realen Systems zu ersetzen. Simulation und Emulation unterscheiden sich in Zielen, Ausführungsgeschwindigkeit, Echtzeitfähigkeit und Berücksichtigung von Reaktionszeiten. Ein Emulationsmodell bildet das System detaillierter ab und überprüft die Leistung oder Reaktionsfähigkeit des Steuerungssystems in begrenzter Anzahl. Es sollte in Echtzeit ausgeführt werden und die Realität widerspiegeln. Simulationsmodellierung ist schnell, während Emulation in Echtzeit erfolgen sollte und Reaktionszeiten berücksichtigt werden sollten.

2.3 Co-Simulation

Co-Simulation bezieht sich laut Gomes u. a., [27], auf die Verfahren und Techniken, die es ermöglichen, das Verhalten eines gekoppelten Systems durch die Zusammenführung von einzelnen Simulationen zu simulieren. Jede Simulation wird hierbei als eine Art „Black Box“ betrachtet, die Eingaben verarbeitet und daraus Ausgaben erzeugt. Co-Simulation ist eine geeignete Methode für die Simulation komplexer Systeme, da sie hohe Flexibilität aufweist und die Möglichkeit bietet, mehrere Simulationen zusammenzuführen, um das Verhalten des Gesamtsystems zu analysieren und zu verstehen, wie sich die verschiedenen Komponenten des Systems gegenseitig beeinflussen.
Ein Beispiel für die Anwendung von Co-Simulation ist die Energiesystemsimulation, bei der verschiedene Simulationen wie die Stromnetzsimulation, Simulation erneuerbarer Energieerzeugung, die Wetterdatensimulation und die Verbrauchssimulation zusammengeführt werden, um das Verhalten des Energiesystems zu untersuchen. Darüber hinaus kann Co-Simulation verwendet werden, um die Leistung eines Systems zu testen und zu optimieren.

2.4 Hardwaresimulation

Hardwaresimulation ist eine Simulation von Hardware, die für die Simulation der Ausführung von Software verantwortlich ist. Die Hauptaufgabe einer Hardwaresimulation nach Frank [26] ist die Abbildung des CPU-Verhaltens. Darüber hinaus kann die Hardwaresimulation das Verhalten der anderen Teile der Hardware repräsentieren.
Aufgrund der Fortschritte in der CPU- und Hardwarearchitektur sind verschiedene Hardwarearchitekturdesigns entstanden. Diese Eigenschaften weisen einen breiten Wertebereich auf, und einzelne Hardwaresimulationen erfassen lediglich einen Teil davon. Bei einem Austausch der Hardwarearchitektur kann es erforderlich sein, die entsprechenden Hardwaresimulationen anzupassen. Daher ist eine allgemeine Schnittstelle erforderlich, um einen einfachen und schnellen Austausch zu ermöglichen.
Außerdem bieten Hardwaresimulationen unterschiedliche Zielsetzungen, deswegen hängt die Wahl der Hardwaresimulation in einer Co-Simulation von den Anforderungen der Simulation ab. Entscheidend sind dabei die Prioritäten in Bezug auf Energieverbrauch, Genauigkeit und Geschwindigkeit. Je nachdem, welche Anforderungen im Vordergrund stehen, wird die

passende Hardwaresimulation ausgewählt. Da die Simulation des Energieverbrauchs bei Hardwaresimulationen noch nicht weit genug verbreitet ist, wird dies daher nicht weiter behandelt.

Jede Hardwaresimulation hat eine Schnittstelle, die mit einer Simulation verbunden werden kann, um den Informationsaustausch zwischen ihnen zu ermöglichen. Nach Rodchenko u. a. [64] enthält die Schnittstelle einer Hardwaresimulation alle notwendigen Informationen, um sowohl die Co-Simulationen als auch die Simulationen zu synchronisieren und zu verbinden. Zu diesen Informationen gehören Eingabeparameter und Rückgabewerte, die zur genauen Steuerung der Interaktion zwischen der Co-Simulation und der Hardwaresimulation verwendet werden, um Ergebnisse zu erzielen. Das heißt, sie ist das Schlüsselkonzept, um die Kommunikation zwischen Hardwaresimulationen und Co-Simulationen zu ermöglichen.

Die Hardwaresimulationen unterscheiden sich in Bezug auf die CPU-Architektur, da jede Architektur ihre eigenen Anforderungen und Einschränkungen hat. Zusätzlich können Unterschiede in Bezug auf den Speicher, den Cache und die Anzahl der Kerne die Leistung und die Genauigkeit der Hardwaresimulationen beeinflussen. Nach Kogge [42] gibt es verschiedene Multi-Core-Speicher-Designs für Hardwarearchitekturen, wie z.B. das hierarchische und das Array-Design. Jedes dieser Designs erfordert eine spezifische Hardwaresimulation, da sie sich in ihrer Funktionsweise unterscheiden. Im hierarchischen Design werden Kerne und Caches in einer baumartigen Konfiguration verwendet, während im Array-Design der In-Chip-Speicher in separate Banks aufgeteilt wird. Es ist wichtig zu beachten, dass es eine Bandbreite an möglichen Hardwarearchitekturen gibt, die nicht durch einzelne Hardwaresimulationen abgedeckt werden.

Es werden zwei einfache beispielhafte Hardwaresimulationen, nämlich gem5 und Multi2Sim, vorgestellt.

Die gem5-Hardwaresimulation wird von Binkert u. a. [16] vorgestellt. Sie bietet eine anpassbare Simulationsumgebung, verschiedene Instruction Set Architecture (ISA)- und CPU-Modelle und ein detailliertes und flexibles Speichersystem. Sie wird von vielen akademischen und industriellen Einrichtungen unterstützt und unterstützt die meisten gängigen ISAs, einschließlich ARM, ALPHA, MIPS, Power, SPARC und x86. Sie ist ein wertvolles Werkzeug für die Full-system Simulation, da sie die Simulation eines Systems einschließlich eines Betriebssystems ermöglicht.
Multi2Sim, basierend auf der Arbeit von Ubal u. a. [77], ist ein Simulationsframework, das die zentralen Komponenten von Mikroprozessoren wie Prozessorkerne, Speicherhierarchie und Verbindungsnetzwerke modelliert. Es wurde entwickelt, um wichtige Merkmale beliebter Simulator-Programme zu integrieren, wie separate funktionale und zeitliche Simulation, Unterstützung von Simultaneous Multithreading (SMT) und Multiprozessoren sowie Cache-Kohärenz. Aufgrund dieser Eigenschaften ist Multi2Sim optimal geeignet, um modernste Prozessoren zu bewerten und deckt aktuelle Themen im Bereich der Computerarchitektur ab. Somit bietet es die Möglichkeit, neue Designs zu evaluieren.

3 Hardwaresimulationen

In diesem Kapitel werden unterschiedliche Hardwaresimulationen vorgestellt. Die Tabelle 3.1 enthält diejenigen Hardwaresimulationen, die in [7] beschrieben wurden. Ergänzend dazu werden weitere Hardwaresimulationen, die im Rahmen der Recherche gefunden wurden und kursiv hervorgehoben sind, vorgestellt. Diese Hardwaresimulationen können zur Simulation von Mikroprozessoren, Cache-Speicher, Dynamic Random Access Memory (DRAM) und Graphics Processing Units (GPUs) verwendet werden.

Die Hardwaresimulationen sind in Tabelle 3.1 nach Namen, unterstützter ISA/Operating System (OS), Pipeline-Modellen, Multi-Core-Fähigkeit und anderen Eigenschaften aufgelistet. Unter den präsentierten Hardwaresimulationen gibt es solche, die entweder Out-Of-Order (OOO) oder In-Order (IO) Pipelines verwenden. Unterschiedliche Hardwaresimulationen können verschiedene Eigenschaften aufweisen, wie z.B. integrierte Timing- und Funktionsmodelle oder nur Timing (TIM)- oder Functional (FUNC)-modelle. Eine Unterscheidungsmöglichkeit bei Timing-Hardwaresimulationen besteht darin, ob sie auf Zyklus-Ebene oder ereignisgesteuert Event-Driven (EvDr) arbeiten.

Darüber hinaus kann eine Hardwaresimulation entweder im Full System (FSys) oder im User Mode (UM) konzipiert werden. Eine Full System Hardwaresimulation ist in der Lage, jedes unterstützte Binärformat zu simulieren und somit das Betriebssystem sowie Anwendungsbenchmarks so auszuführen, als ob sie auf einem echten Computer laufen würden. User-Modus Hardwaresimulationen hingegen konzentrieren sich darauf, spezifische Zielanwendungen zu simulieren, indem sie lediglich den Mikroprozessor emulieren und die Systemaufrufe vom Host-Betriebssystem ausführen lassen.

Einige der vorgestellten Hardwaresimulationen unterstützen auch Multicore-Modelle, während andere nur für Single-Core-Modelle geeignet sind. Dennoch unterscheiden sich die Hardwaresimulationen hinsichtlich der Eingabe bei der Kommunikation mit der Co-Simulation, ob sie Trace-Dateien (Trace-Driven (TD)) oder ausführbare Binärdateien (Execution-Driven (EDr)) verwenden. Einige der genannten Hardwaresimulationen sind kommerzielle Produkte, während andere als Open-Source-Tools zur Verfügung stehen. Obwohl die meisten dieser Hardwaresimulationen für x86-Architekturen optimiert sind, gibt es auch einige, die für andere Architekturen wie Acorn RISC Machines (ARM), Microprocessor without Interlocked Pipeline Stages (MIPS), Alpha und ähnliche ausgelegt sind.

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
1	ASim [24]	x86	Alpha, x86	OOO	ja	UM, MOD, TIM (Entkoppelte Timing- und Funktionsmodelle)	*EDr*
2	Augmint [56]	x86/Unix, x86/Windows NT	x86	-	ja	EDr	EDr, *TD*
3	CMP$im [37]	x86	x86	-	ja	Paralleler UM-Cache (Entkoppelt)	*EDr*
4	COTSon [10]	x86	x86	-	ja	FSys, FUNC	*EDr*
5	Dinero IV [23]	x86/Linux, Alpha/Linux, x86/Solaris, Alpha/OSF, SPARC/Solaris	*mehrstufige Caches*	-	nein	TD-Cache	*TD*
6	DRAMSim [84]	x86/Linux	*SDRAM, DDR, DDR2, DRDRAM und FB-DIMM*	-	nein	TD Dynamic Random Access Memory (DRAM) auf Zyklusebene	*TD*
7	ESESC [9]	x86-64/Linux und ARMv7	ARMv7	OOO, IO	ja	TIM, UM (Zyklusebene)	*EDr*
8	*FireSim [40]*	*FPGA*	*FPGA*	*FSys*	*ja*	*TIM (Zyklusebene)*	*EDr*
9	Flexus [25]	x86/Linux	SPARC, x86	OOO, IO	ja	FSys, TIM, EDr (Zyklusebene)	*EDr*
10	gem5 [16]	x86, ARM, SPARC, Alpha, PPC, Linux, MacOSx, Solaris, OpenBSD	x86, ARM, MIPS, Alpha, PPC, SPARC	OOO, IO	ja	FSys, MOD, TIM (Zyklusebene)	*C++, (Java)*
11	GEMS [49]	x86/Linux, AMD64-linux, SPARCV9, (Solaris 8)	SPARC, x86	OOO	ja	FSys, TIM (Entkoppelte Timing- und Funktionsmodelle)	*EDr*

Fortsetzung auf der nächsten Seite

Tabelle 3.1 – Fortsetzung von der vorherigen Seite

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
12	GPGPU-Sim [1]	Linux	PTX und SASS, PTX-Plus	OOO, IO	ja	UM Zyklusebene (Entkoppelte Timing- und Funktionsmodelle)	*EDr*
13	Graphite [52]	x86/Linux	x86	IO mit IO oder OOO memory completion	ja	Paralleler UM, TIM (Entkoppelt)	*EDr*
14	HASE [33]	x86/Linux, MAC, Windows	MIPS	OOO	nein	MOD, FSys	*EDr*
15	HAsim [59]	FPGA	MIPS	OOO	ja	FPGA-basiertes TIM	*EDr*
16	*HORNET [61]*	*MIPS*	*MIPS, TD*	*NoC*	*ja*	*TIM (Zyklusebene)*	*TD*
17	LSE [82]	x86/Unix	PowerPC, SPARC, IA64, DLX	OOO	ja	MOD	*EDr*
18	LiveSim [30]	x86	MIPS64	OOO	nein	TIM, UM (Zyklusebene)	*EDr*
19	*MaxSim [64]*	*x86-64*	*x86-64*	*OOO, IO*	*ja*	*TIM*	*Java*
20	MARSS-x86 [58]	x86-64/Linux	x86-64	OOO, IO	ja	FSys, TIM (Entkoppelte Timing- und Funktionsmodelle)	*C++*
21	McPAT [44]	x86/Linux	Alpha, ARM, x86, SPARC	-	ja	Power, area und TIM	*EDr*
22	McSimA+ [5]	x86	x86	OOO, IO	ja HMP	UM, TIM (Entkoppelte Timing- und Funktionsmodelle)	*EDr*

Fortsetzung auf der nächsten Seite

Tabelle 3.1 – Fortsetzung von der vorherigen Seite

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
23	MicroLib [60]	x86	Alpha, PowerPC, SHARC	OOO	no	MOD, FSys	*EDr*
24	Mint [83]	SGI, SPARC und DEC stations	MIPS	-	ja	UM, EDr	*EDr*
25	MLRSim [70]	x86/Linux, SGI, IRIX, SPARC/-Solaris	SPARC v8	OOO	nein	FSys, TIM (EvDr, MOD)	*EDr*
26	Multi2Sim [77]	x86/Linux	MIPS32, x86, ARM, Evergreen, NVIDIA Fermi	OOO	ja HMP	UM, MOD TIM	*EDr*, *C++*
27	MSim [72]	Linux, x86, Win2000, SPARC/Solaris	Alpha	OOO, IO	ja	UM, TIM (Zyklusebene)	*EDr*
28	OVPsim *[65]*	x86/Windows, x86/Linux	ARM, MIPS, x86	-	ja HMP	FSys, FUNC	*EDr*
29	PTLsim [86]	x86/Linux	x86	OOO	ja	FSys, TIM (Zyklusebene)	*EDr*
30	RSim [32]	SUN-Rechner mit Solaris 2.5, SGI Power Challenge running IRIX 6.2	SPARC v8	OOO	ja	UM, EDr, EvDr TIM	*EDr*
31	SESC [12]	Unix-basierte Systeme (z. B. Linux und Darwin/MacOSx)	MIPS	OOO	ja	UM, TIM, EvDr	*EDr*
32	Shade [20]	SPARC	SPARC (v8 und v9)	-	nein	Profiler	*EDr*
33	SIMCA [31]	SPARC/Solaris	Alpha, x86	OOO	ja	UM, EDr, TIM	*EDr*

Fortsetzung auf der nächsten Seite

Tabelle 3.1 – Fortsetzung von der vorherigen Seite

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
34	SimCore [41]	x86/Linux, Alpha/Linux, UltraSPARC/-Solaris, MIPS IRIX	Alpha	-	ja	UM FUNC	*EDr*
35	SIMFLEX [29]	x86/Linux	x86, SPARC	OOO	ja	FSys, MOD, TIM (Entkoppelte Timing- und Funktionsmodelle)	*EDr*
36	SIMICS [2] *[3]*	Alpha, PPC, UltraSPARC, x86/Linux, Windows	Alpha, ARM, MIPS, PPC, SPARC, x86, Linux, Solaris, Windows	-	ja	FSys, FUNC	*EDr*
37	SimOS *[66]*	x86/Linux, MIPS IRIX	SGI, IRIX MIPS	OOO	ja	FSys, TIM	*EDr*
38	SimpleScalar [11]	Linux/x86, Win2000/x86, SPARC/Solaris	Alpha, Pisa, ARM, x86	OOO	nein	UM, EDr, TIM	*EDr*
39	SiNUCA [8]	x86-64/Linux	x86-64	OOO	ja	TD UM, TIM	*TD*
40	Sniper [18]	x86/Linux	x86, RISC-V	OOO, IO	ja HMP	Paralleler UM, TIM	*EDr*, *TD*, *C++*
41	SMTSIM [75]	Alpha/Unix, x86/Linux	Alpha	OOO	ja	TIM	*EDr*
42	SPim [43]	Windows, MacOSx, Linux	MIPS32	-	nein	FUNC	*EDr*
43	*Tejas [69]*	*x86*	*x86*	*OOO, IO*	*ja*	*UM, TIM (Zyklusebene)*	*TD (Java Trace) oder C++*
44	TEM2P2EST [22]	x86/Linux	Alpha, Pisa, ARM, x86	OOO	nein	Power, TIM (Zyklusebene)	*EDr*
45	Turandot [53]	AIX, Linux	PowerPC	OOO, IO	ja	UM, TIM	*EDr*

Fortsetzung auf der nächsten Seite

Tabelle 3.1 – Fortsetzung von der vorherigen Seite

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
46	Wisconsin Wind Tunnel II [55]	SPARC	SPARC	IO	ja	Paralleler diskreter EvDr, EDr	*EDr*
47	Zesto [47]	x86	x86	OOO	ja	UM, TIM (Zyklusebene)	*EDr*
48	ZSim++ [67]	x86/Linux	x86-64	OOO, IO	ja	Paralleler TIM, UM Simulator	*(Java)*, *C++*, *Python*, *EDr*

Tabelle 3.1: Ein Überblick über die Hardwaresimulationen von Akram und Sawalha [7] mit neuen kursiv markierten Einträgen aus [28] und [71] hervorgehoben

Im Folgenden werden einige Begriffe erläutert, die in Tabelle 3.1 vorkommen und sich auf die Instruction Set Architecture sowie auf weitere Abkürzungen im Bereich der Betriebssysteme beziehen.

Instruction Set Architecture : In Tabelle 3.1 sind verschiedene Arten von Architekturen aufgelistet, die in Computern und elektronischen Geräten verwendet werden. Die spezifischen Eigenschaften und Vorteile jeder Architektur nach [21] variieren je nach Anwendung. Nachfolgend werden die Abkürzungen und ihre entsprechenden Architekturen kurz erläutert.

- x86: Eine Prozessorarchitektur von Intel, die weit verbreitet in PCs und Laptops ist. Sie ist auch als ISA-64 oder x86-64 bekannt.
- ARM: Eine Prozessorarchitektur, die von ARM Holdings entwickelt wurde und weit verbreitet in mobilen Geräten wie Smartphones und Tablets ist. Sie ist auch als ARMv7 oder ARMv8 bekannt.
- PowerPC: Eine Prozessorarchitektur, die von IBM, Apple und Motorola entwickelt wurde und in einigen älteren Mac-Computern und Spielkonsolen wie der Xbox 360 eingesetzt wurde.
- SPARC: Eine von Sun Microsystems entwickelte Prozessorarchitektur, die in einigen Servern und Workstations eingesetzt wurde. Sie ist auch als SPARC V8 oder SPARC V9 bekannt.
- MIPS: Eine Prozessorarchitektur, die in einigen eingebetteten Systemen wie Routern und Fernsehgeräten eingesetzt wird. Sie ist auch als MIPS32 oder MIPS64 bekannt.
- Alpha: Eine von Digital Equipment Corporation (DEC) entwickelte Prozessorarchitektur, die in einigen Workstations und Servern eingesetzt wurde.

- RISC: Steht für „Reduced Instruction Set Computer“ und bezeichnet eine Prozessorarchitektur, die sich auf Effizienz, Geschwindigkeit und Skalierbarkeit konzentriert. RISC-Designs finden Anwendung in Supercomputern, eingebetteten Systemen und mobilen Geräten.

Weitere Abkürzungen gemäß [46] im Bereich der Betriebssysteme sind:

- Linux: Ein kostenloses und quelloffenes Betriebssystem, das auf verschiedenen Architekturen wie x86, Alpha, MIPS und PPC läuft.
- MacOSx: Das Betriebssystem von Apple für ihre Computer, das auf der x86 und der PowerPC Architektur läuft.
- Solaris: Ein Betriebssystem von Oracle für ihre Server und Workstations, das auf der SPARC und x86 Architektur läuft.
- OpenBSD: Ein kostenloses und sicheres Betriebssystem, das auf verschiedenen Architekturen wie x86 läuft.
- NTx86: Eine Abkürzung für das Betriebssystem Windows von Microsoft, das auf der x86 Architektur läuft.
- IRIX: Ein Betriebssystem von Silicon Graphics für ihre Workstations und Server, das auf der MIPS Architektur läuft.
- Darwin: Das Basisbetriebssystem von MacOSx, das auf der x86 und der PowerPC Architektur läuft.

4 Filterung von Hardwaresimulationen

Aufgrund der großen Vielfalt an Hardwaresimulationen, die in unterschiedlichen Bereichen eingesetzt werden können, stellt ihre Klassifizierung eine Herausforderung dar. Es ist daher schwierig, einheitliche Kriterien für die Klassifizierung aller Hardwaresimulationen in Tabelle 3.1 festzulegen. Um dieser Herausforderung zu begegnen, werden im nächsten Schritt Kriterien zur Abgrenzung von Hardwaresimulationen definiert. Dabei werden diejenigen Hardwaresimulationen ausgeschlossen, die diese Kriterien erfüllen, um sie bei der Klassifizierung nicht weiter zu berücksichtigen.

4.1 Kriterien zur Abgrenzung von Hardwaresimulationen

Es gibt verschiedene Arten von Hardwaresimulationen, wie KI-basierte Hardwaresimulationen, grafische Hardwaresimulationen und Standard-Hardwaresimulationen. Die KI-basierte Hardwaresimulation verwendet Künstliche Intelligenz, um bestimmte Vorhersagen der Hardwaresimulation zu verbessern.

Ein Beispiel dafür ist das in [38] vorgestellte Modell, das künstliche neuronale Netze (ANN) einsetzt, um das Verhalten des privaten Least Recently Used (LRU)-Caches auf Out-Of-Order-Prozessoren um das 2,5- bis 3-fache schneller als die gem5-Hardwaresimulation vorherzusagen. Dabei konzentriert sich diese Art der Simulation auf bestimmte Komponenten der Hardwaresimulation, während andere Aspekte dabei nicht berücksichtigt werden.

Grafische Simulationen hingegen enthalten spezielle GUI-Komponenten und zielen darauf ab, bestimmte Aspekte der Hardware zu simulieren, insbesondere im Zusammenhang mit Grafikverarbeitung und Hardware-Beschleunigern. Diese Art der Simulation legt den Fokus auf Hardwarekomponenten und -verhalten, die für grafische Anwendungen und spezielle Berechnungen relevant sind.

Standard-Hardwaresimulationen sind umfassende Simulationen der Standard-Hardware, bei denen eine detaillierte Nachbildung der CPU-Simulation erfolgt. Hier werden keine spezifischen Techniken wie KI oder Grafiksimulation verwendet. Stattdessen liegt der Fokus auf der Simulation der Gesamtfunktionalität und des Verhaltens der Standard-Hardware.

Deshalb werden ausschließlich Hardwaresimulationen berücksichtigt, die eine Nachbildung der CPU-Simulation durchführen. Die Fokussierung auf die vollständige Simulation der Standard-Hardware ermöglicht eine umfassende Analyse und Bewertung der Hardwarearchitektur. Es gibt zwei verschiedene Abgrenzungen. Erstens werden bestimmte Arten von Hardwaresimulationen von vornherein ausgeschlossen, wie solche, die auf Künstlicher Intelligenz basieren, und sind daher auch nicht Teil der vorgestellten List 4.1. Zweitens gibt es eine Liste von Kriterien für Hardwaresimulationen, die im Rahmen der Recherche gefunden wurden, aber nicht zur Klassifizierung geeignet sind. Eine detaillierte Liste dieser nicht klassifizierungsfähigen Hardwaresimulationen findet sich in Tabelle 4.1.

Ein erstes Kriterium für den Ausschluss von Hardwaresimulationen betrifft **grafische Simulationen**, die spezielle GUI-Komponenten enthalten. Beispiele solcher grafischen Hardwaresimulationen sind GPGPUsim und HAsim.

Des Weiteren werden keine Hardwaresimulationen einbezogen, die lediglich einen Teil der Hardware simulieren, wie beispielsweise **Simulationen des Caches oder des Speichers**. Obwohl solche Hardwaresimulationen in der Lage sind, bestimmte Befehle auszuführen, sind sie hier nutzlos, da sie nur einen begrenzten Teil der Hardware simulieren. Da eine vollständige Simulation der gesamten Hardware angestrebt wird, ist eine Hardwaresimulation für die gesamte Simulation erforderlich. Ein Beispiel für eine Speicher-Hardwaresimulation ist DRAMSim, welche Befehlszugriffe auf den Speicher simuliert. Für Cache-Hardwaresimulationen gibt es beispielsweise Dinero IV- und CMP$im-Hardwaresimulation.

In der vorliegenden Untersuchung werden ausschließlich Hardwaresimulationen betrachtet, die auf einem spezifischen **Host-System** ausgeführt werden können. Da für diese Arbeit nur eine x86-64-Architektur zur Verfügung steht, wurde diese als Host-System ausgewählt, um sicherzustellen, dass die Hardwaresimulationen erfolgreich ausgeführt werden können. Andere Hardwaresimulationen, die auf Architekturen wie PowerPC, SPARC, PTX oder SASS basieren, werden nicht in die Untersuchung einbezogen. Beispiele solcher Simulationen sind FireSim, Mint, RSim und Wisconsin Wind Tunnel II.

Eine Möglichkeit, die Konvertierung von Schnittstellen der Hardwaresimulation in allgemeine Schnittstellen zu erleichtern, besteht darin, anstelle von **Trace-Dateien** eine ausführbare Datei als Eingabe für die Hardwaresimulation zu verwenden. Trace-Dateien repräsentieren eine Abfolge von Anweisungen, die von einem Benchmark mit festen Eingaben ausgeführt werden. Diese Art der Eingabe berücksichtigt jedoch nicht die Laufzeitänderungen, die bei der Verwendung von Multithread-Anwendungen auftreten können. Darüber hinaus ist es erforderlich, einen auf die Hardware abgestimmten Tracer zu generieren, bevor eine Simulation durchgeführt werden kann, was für die Exploration verschiedener Architekturen ungünstig ist. Beispiele für Hardwaresimulationsprogramme, die Trace-Dateien als Eingabe verwenden, sind Shade, DRAMSim, Dinero IV, HORNET, SiNUCA und Tejas.

Des Weiteren sollten nur diejenigen Hardwaresimulationen berücksichtigt werden, die lauffähig sind und von der Gemeinschaft unterstützt werden. Dies bedeutet, dass nur Hardwaresimulationen in Betracht gezogen werden sollten, bei denen der **letzte Commit höchstens 5 Jahre alt** ist. Folgende Hardwaresimulationen haben seit mehr als 5 Jahren keinen Commit mehr erhalten: Augmint, ASim, COTSon, SimCore, SIMFLEX, GEMS, Graphite, LSE, MaxSim, MARSSx86, McPAT, MSim, PTLsim, SIMCA, SimOS, SMTSIM, Turandot und Zesto.

In Anbetracht der aktuellen Entwicklungen sollten heutzutage nur Hardwaresimulationen in Betracht gezogen werden, die die Unterstützung von **Multicore-Prozessoren** bieten. Beispiele für Hardwaresimulationen, die nur die Unterstützung von Single-Core-Prozessoren bieten, sind HASE, LiveSim, MicroLib, MLRSim, SimpleScalar, SPim und TEM2P2EST. Für Hardwaresimulationen ist es außerdem wichtig, dass die benötigten **Informationen öffentlich verfügbar** sind, um sie klassifizieren zu können. Aus diesem Grund werden Hardwaresimulationen, bei denen die notwendigen Informationen nicht öffentlich verfügbar sind, wie bei SIMICS, ausgeschlossen.

#	**Kriterien**	**Hardwaresimulationen**
1	Grafische Hardwaresimulationen	GPGPUSim [1] und HAsim [59]
2	Speicher-Hardwaresimulation	DRAMSim [84]
3	Cache-Hardwaresimulationen	Dinero IV [23] und CMP$im [37]
4	Trace-Driven	Shade [20], DRAMSim [84], Dinero IV [23], HORNET [61], SiNUCA [8] und Tejas [69]
5	Keine Commits seit über 5 Jahren	Augmint [56], ASim [24], COTSon [10], SimCore [41], SIMFLEX [29], GEMS [49], Graphite [52], LSE [82], MaxSim [64], MARSSx86 [58], McPAT [44], MSim [72], PTLsim [86], SIMCA [31], SimOS [66], SMTSIM [75], Turandot [53] und Zesto [47]
6	Single-Core-Prozessoren	HASE [33], LiveSim [30], MicroLib [60], MLRSim [70], SimpleScalar [11], SPim [43] und TEM2P2EST [22]
7	Notwendige Informationen nicht öffentlich verfügbar	SIMICS [2]
8	Keine Unterstützung für x86-64-Architektur als Host	FireSim [40], Mint [83], RSim [32] und Wisconsin Wind Tunnel II [55]

Tabelle 4.1: Abgrenzung von Hardwaresimulationen

4.2 Klassifizierbare Hardwaresimulationen

Nach Anwendung der Abgrenzungskriterien auf die Hardwaresimulationen in Tabelle 3.1 bleiben diejenigen übrig, die für mögliche Klassifizierungen verwendet werden können. Diese ausgewählten Hardwaresimulationen in Tabelle 4.2 werden dann genauer untersucht, um potenzielle Eigenschaften zu identifizieren. Eine allgemeine Schnittstelle wird für eine bestimmte Klasse von Hardwaresimulationen entwickelt. Bei der Ausführung dieser Hardwaresimulationen ist es von entscheidender Bedeutung, Faktoren wie die erforderlichen Ausführungstools, die Eingabe- und Ausgabeformate bei der Klassifizierung zu beachten. Dennoch bietet die nachfolgende Tabelle einen Überblick über die Host- und Zielhardware sowie deren Eigenschaften, die auch bei der Klassifizierung berücksichtigt werden sollen.

#	**Name**	**Hosts (ISA/OS)**	**Ziele (ISA)**	**Pipeline-Modelle**	**Multi-Core**	**Eigenschaften**	***Eingabetyp***
1	ESESC [9] *[39]*	x86-64/Linux und ARMv7	ARMv7	OOO, IO	ja	TIM, UM (Zyklusebene)	*EDr*
2	Flexus [25] *[74]*	x86/Linux	SPARC, x86	OOO, IO	ja	FSys, TIM, EDr (Zyklusebene)	*EDr*
3	gem5 [16] *[15]*	x86, ARM, SPARC, Alpha, PPC, Linux, MacOSx, Solaris, OpenBSD	x86, ARM, MIPS, Alpha, PPC, SPARC	OOO, IO	ja	FSys, MOD, TIM (Zyklusebene)	*C++, EDr, (Java)*
4	McSimA+ [5] *[6]*	x86	x86	OOO, IO	ja HMP	UM, TIM (Entkoppelte Timing- und Funktionsmodelle)	*EDr*
5	Multi2Sim [77] *[76] [78]*	x86/Linux	MIPS32, x86, ARM, Evergreen, NVIDIA Fermi	OOO	ja HMP	UM, MOD TIM	*C++, EDr*
6	OVPsim *[65] [45]*	x86/Windows, x86/Linux	ARM, MIPS, x86	-	ja HMP	FSys, FUNC	*EDr*

Fortsetzung auf der nächsten Seite

Tabelle 4.2 – Fortsetzung von der vorherigen Seite

#	Name	Hosts (ISA/OS)	Ziele (ISA)	Pipeline-Modelle	Multi-Core	Eigenschaften	*Eingabetyp*
7	SESC [12] *[13]*	Unix-basierte Systeme (z. B. Linux und Darwin/MacOSx)	MIPS	OOO	ja	UM, TIM, EvDr	*EDr*
8	Sniper [18] *[19]*	x86/Linux	x86, RISC-V	OOO, IO	ja HMP	Paralleler UM, TIM	*EDr, TD, C++*
9	ZSim++ [67] *[68]*	x86/Linux	x86-64	OOO, IO	ja	Paralleler TIM, UM Simulator	*(Java), C++, Python, EDr*

Tabelle 4.2: Übersicht von Hardwaresimulationen nach Abgrenzungen

5 Untersuchung von Hardwaresimulationen unter Ein- und Ausgabeparametern

Im Abschnitt 4.2 werden die Hardwaresimulationen in Tabelle 4.2 dargestellt, um eine Klassifizierung durchzuführen. Aufgrund der Tatsache, dass alle Hardwaresimulationen in Tabelle 4.2 Execution-Driven (EDr) sind, ist für jede von ihnen eine binäre Datei erforderlich. Dieser Eingabeparameter ist bei allen gleich und wird daher nicht jedes Mal aufgeführt. Zu diesem Zweck werden die Hardwaresimulationen zunächst anhand ihrer Eingabe und Ausgabe verglichen, um ähnliche Parameter sowohl bei der Eingabe als auch bei der Ausgabe zu identifizieren. Teilweise sind diese Eingabe- und Ausgabeparameter in Hardwaresimulations-Papers, auf GitHub von Hardwaresimulationen oder in Tutorials zur Hardwaresimulation zu finden, wobei stets die Quellenreferenz angegeben wird. Dieser Vergleich kann bei der Entscheidungsfindung während der Klassifizierung hilfreich sein.

Es können unterschiedliche Parameter genutzt werden, um die dargestellte Hardwaresimulation anzupassen. Es erfordert, diese Parameter zu finden, was das Durchsuchen einer großen Menge an Informationen bedeutet. Gelegentlich sind gewisse Parameter in Repositorien oder wissenschaftlichen Arbeiten recht spezifisch beschrieben. Dies schließt aber nicht automatisch aus, dass auch andere Parameter unterstützt werden könnten. Die Wahrscheinlichkeit, eine funktionierende Konfiguration und eine einsatzfähige Eingabe zu haben, die auf dem basiert, was bereits spezifiziert wurde, ist hoch. Daher wurde keine Identifikation von Parametern anhand weiterer Ressourcen, wie zum Beispiel Quellcode, durchgeführt.

Das übergeordnete Ziel besteht zudem darin, eine allgemeine Schnittstelle zu konzipieren. Angesichts der Menge an in Betracht zu ziehenden Hardwaresimulationen muss eine Art und Weise gewählt werden, die vergleichsweise schnell zu guten Ergebnissen führt. Eine Möglichkeit hierbei ist die Nutzung von Beispielen, die von den Entwicklern der Hardwaresimulation bereitgestellt wurden. Allerdings decken diese Beispiele nicht notwendigerweise sämtliche möglichen Parameter ab.

5.1 ESESC

Gemäß [62] handelt es sich bei ESESC um eine leistungsfähige Multiprozessor-Hardwaresimulation, die detaillierte Modelle für Leistung, Wärme und Performance von modernen Out-of-Order-Multicores bietet. ESESC ist außerdem eine Weiterentwicklung der SESC-Hardwaresimulation und bringt viele neue Funktionen mit sich. Im Unterschied zu SESC ermöglicht ESESC die Ausführung unveränderter Linux-Binärdateien für RISCV und MIPS ohne die Notwendigkeit einer speziellen Toolchain. Für die Emulation benötigt ESESC die Verwendung von Quick Emulator (QEMU).

Eingabe: Um die Simulation durchzuführen, werden eine Konfigurationsdatei und die entsprechenden Binärdateien als Eingabe benötigt. Die möglichen Parameter für die Hardwaresimulation sind nach [9] in der Tabelle 5.1 dargestellt. Dabei beziehen sich die Werte für „schnell“ und „langsam“ auf unterschiedliche Typen von Prozessorkernen in den CMP-Systemen.

Parameter	**Wert für schnell**	**Wert für langsam**
Freq	3.0 GHz	
I$	32KB 2w (2c hit) privat	
D$	32KB 8w (3c hit) privat	
L2	256KB 16w (12c hit) privat	
L3	4MB 16w (12c hit) shared	
Coherence	MESI	
Memory-latency	180 Zyklen	
BPred.	10 tab. ogehl 76Kb	Hybrid 38Kb
Issue	4	2
ROB	256	56
IWin.	32	16
Load/StoreQ	48/32	16/8
Reg(I/F)	128/128	80/6

Tabelle 5.1: Parameter nach [9] für die ESESC-Hardwaresimulation

Ausgabe: Das ausführbare Skript „report.pl“ dient zur Anzeige von Statistiken aus einem ESESC-Lauf. Hierbei wird ein sogenannter „Dump“ verwendet. Die Datei „report.pl“ enthält Informationen zu Statistiken über Memory-Lese-/Schreibvorgänge, Caches, IPC (Instructions per Cycle), Instruktionszählungen und Zyklen

5.2 Flexus

Zur Ausführung nutzt Flexus nach [74] die QFlex-Komponente, die auf QEMU basiert.

Eingabe: Flexus verwendet nach [25] in Dokument „Flexus - Ein Leitfaden für den Einstieg“ verschiedene Verzeichnisse und Dateien zur Festlegung von Konfigurationsparametern.
Hier sind einige wichtige Konfigurationsparameter für Flexus:
Globale Konfigurationsparameter: Diese Parameter werden in der Datei
`SCRIPT_ROOT/global.run_job.rc.tcl` definiert.
Dort finden Sie relevante Parameter wie:

- `basedir`: Der Pfad, in dem die Ergebnisse der Simulation gespeichert werden sollen.
- `specdir`: Ist der Pfad, in dem die globalen Konfigurationsparameter für Flexus-Simulationen gespeichert werden. Diese Parameter umfassen wichtige Einstellungen wie das Speicherlimit, die Konsolenausgabe, Ausgabedateien und das Watchdog-Timeout für Flexus.

- `userspecdir`: Der Pfad, in dem wichtige Konfigurationsparameter wie Cache-Größe, Assoziativität usw. gespeichert werden, enthält die wesentlichen Konfigurationsparameter für die Hardwaresimulation, wie in Tabelle 5.2 nach [74] dargestellt.
- `ckptdir`: Der Pfad, in dem die generierten Flex-Points (Checkpoint-Dateien) abgelegt werden.

Parameter	**Wert**	**Beschreibung**
bpwarm:cores	1	Anzahl der Kerne
feeder:stick	0,0	CPU System-Taktfrequenz. 0,0 belässt die Frequenz unverändert
feeder:housekeeping_period	1000	Simics-Zyklen zwischen den Housekeeping-Ereignissen
feeder:ifetch	1	Verfolge und protokolliere Instruction Fetches
feeder:CMPwidth	1	Anzahl der Kerne pro CMP-Chip (0 = Systembreite)
feeder:send_non_allocating-_stores	0	Sende NonAllocatingStores ein/aus
L1d:mt_width	1	Anzahl der Threads, die diesen Cache gemeinsam nutzen
L1d:size	32768	Cache-Größe in Bytes
L1d:assoc	2	Assoziativität einstellen
L1d:bsize	64	Blockgröße
L1i:mt_width	1	Anzahl der Threads, die diesen Cache gemeinsam nutzen
L1i:size	49152	Cache-Größe in Bytes
L1i:assoc	3	Assoziativität einstellen
L1i:bsize	64	Blockgröße
L2:CMPWidth	1	Anzahl der Kerne pro CMP-Chip (0 = Systembreite)
L2:size	2097152	Cache-Größe in Bytes
L2:assoc	16	Assoziativität einstellen
L2:bsize	64	Blockgröße
memory:time	90	Zugriffszeit
memory:max_requests	64	Maximale Anzahl an Anfragen in der Loopback-Warteschlange
memory:UseFetchReply	1	Sende FetchReply als Antwort auf FetchReq (statt MissReply)

Tabelle 5.2: Parameter nach für die Flexus-Hardwaresimulation

Ausgabe: Flexus stellt nach [25] in Dokument „Flexus - Ein Leitfaden für den Einstieg“ eine umfangreiche Auswahl an Ergebnissen und Statistiken bereit. Bei der Durchführung der Timing-Simulation in Flexus werden separate Datenbanken für jeden Flex-Punkt erstellt. Diese

Datenbanken enthalten eine Vielzahl von Statistiken, einschließlich Informationen über die Anzahl der ausgeführten Befehle (Instruction Counts). Durch die Verwendung von Flexus ist es möglich, die Ausführung der Befehle zu überwachen und aufzuzeichnen.

Zusätzlich dazu bietet Flexus Cache-Ereigniszähler (Cache Event Counts), welche Informationen über Cache-Miss-Raten, Cache-Treffer und andere Ereignisse im Cache-System liefern. Diese Statistiken werden über einen oder mehrere Messbereiche erfasst.

5.3 gem5

Eingabe: Die Konfigurationsskripte für gem5 sind nach [15] Python-Skripte, die als Parameter von der gem5-Binärdatei akzeptiert werden. Mit Hilfe dieser Skripte wird die Hardware-Simulation konfiguriert und gestartet. Das Python-Skript enthält die in Tabelle 5.3 aufgeführten Parameter, die verwendet werden, um eine Binärdatei mit der entsprechenden Konfiguration für die Hardwaresimulation gemäß der Beschreibung in [17] auszuführen.

Komponente	**Parameter**	**Wert**
ISA	isa_required	X86
Kohärenzprotokoll	coherence_protocol_required	MESI_TWO_LEVEL
Cache-Hierarchie	l1d_size	32KiB
	l1d_assoc	8
	l1i_size	32KiB
	l1i_assoc	8
	l2_size	256KiB
	l2_assoc	16
	num_l2_banks	1
Speicher	size	3GiB
Prozessor	starting_core_type	CPUTypes.TIMING
	switch_core_type	CPUTypes.O3
	num_cores	2
	isa	ISA.X86
Board	clk_freq	3GHz
	processor	SimpleSwitchableProcessor
	memory	SingleChannelDDR3_1600
	cache_hierarchy	MESITwoLevelCacheHierarchy

Tabelle 5.3: Parameter für die gem5-Hardwaresimulation

Ausgabe: Nach der Ausführung von gem5 im Verzeichnis „m5out“ werden drei Dateien generiert.

- Die Datei „stats.txt“ enthält umfangreiche statistische Informationen über die Simulation, darunter die Gesamtzahl der ausgeführten Befehle, der Cache-Treffergrad, die Speicherzugriffe sowie spezifische Statistiken für jedes SimObject wie gelesene Bytes und die durchschnittliche Bandbreite des Speichercontrollers. Die Datei enthält auch CPU-Statistiken zu Systemaufrufen, Verzweigungen und der Gesamtanzahl der ausgeführten Instruktionen.

- Die Datei „config.ini“ bietet eine detaillierte Übersicht über die Konfiguration der Simulationsobjekte. Hier werden alle Parameter jedes SimObjects aufgeführt, einschließlich derjenigen, die im Konfigurationsskript festgelegt wurden oder Standardwerte verwenden.
- Zusätzlich gibt es die Datei „config.json“, die eine alternative Konfigurationsdarstellung im JSON-Format liefert. Sie enthält ebenfalls Informationen zur Konfiguration der Simulationsobjekte, ist jedoch in einem anderen Dateiformat strukturiert.

In der Arbeit wird nur die Datei `stats.txt` betrachtet, da sie die Simulationsergebnisse enthält, die von größerer Bedeutung sind und umfangreichere Informationen liefern als die Konfigurationsübersicht. Die Konfigurationsdateien `config.ini` und `config.json` zeigen zwar die vom Simulationssystem festgelegte Konfiguration, sind jedoch als Ausgabe nur begrenzt relevant.

5.4 McSimA+

Für die Hardwaresimulation wird das Pin-Tool verwendet, das als dynamische Bibliothek im Pthread-Verzeichnis kompiliert wird. In [4] wird der McSimA+ Hardwaresimulation modifiziert, um moderne C++17 Features zu nutzen.

Eingabe: In Tabelle 5.4 sind gemäß [5] die potenziellen Parameter für die Hardwaresimulation aufgeführt.

Parameter	**Wert**
Freq (GHz)	2.53
RS entry	36
(IF/CM/IS) width	4/4/6
L2$ per core	256KB, 8-way, inclusive
Cores/chip	4
L1 I-TLB entry	128
L1 I$	32KB, 4-way
L3$ (shared)	8MB, 16-way, inclusive
ROB entry	128
L1 D-TLB entry	64
L1 D$	32KB, 8-way
Main memory	3 channels, DDR3-1333

Tabelle 5.4: Parameter [5] für die McSimA+

Ausgabe: McSimA+ zeichnet die Anzahl der ausgeführten Instruktionen, die Ergebnisse des IPC (Instructions Per Cycle), die Gesamtzeit für Speicher-Schreibvorgänge und -Lesevorgänge, sowie die Anzahl der Treffer und Fehlschläge im Level-1-Cache und Level-2-Cache sowie den Durchschnitt der Datenabhängigkeiten und Zyklen auf.

5.5 Multi2Sim

Das „INI“-Dateiformat wird sowohl für die Eingabe der Konfigurationsdatei als auch für die Ausgabe von Konfigurationsberichten verwendet.

Eingabe: Gemäß [76] und [79] stellt die Konfiguration für die Hardwaresimulation in Tabelle 5.5 dar.

Parameter	**Wert**
Anzahl der Kerne	2
Frequenz	1 GHz
Anzahl der Banken	32
L1-Cache Assoziativität	8 Wege
L1-Cache Zugriff durch	16 Latenz (Zyklen)
L1-Cache Blockgröße	64B
L1-Cache Anzahl der Lese-/Schreibports	16R/16W
L1-Cache Gesamtgröße	8 KB
Anzahl der L2-Caches	32
L2-Cache Assoziativität	8-Wege-Set
L2-Cache Latenz (Zyklen)	10
L2-Cache Blockgröße	256B
L2-Cache Anzahl der Lese-/Schreibports	2R/2W
L2-Cache Gesamtgröße	4x128 KB = 512 KB
Anzahl der Banken im L2-Cache	128
Busbreite	256B/Zyklus
Speicher Latenz (Zyklen)	100
Globale Speichergröße	1 GB

Tabelle 5.5: Parameter nach [76] für die Multi2Sim-Hardwaresimulation

Ausgabe: Der Statistikbericht enthält nach [54] Informationen über die Leistung des Systems und spezifische Metriken für den x86-Prozessor. Hier sind die einzelnen Variablen und ihre Erklärungen:

Allgemein

- *RealTime*: Die tatsächlich benötigte Zeit für die Simulation in Sekunden.
- *SimEnd*: Die Anzahl der abgeschlossenen Kontexte (ContextsFinished) zum Zeitpunkt des Simulationsendes.
- *SimTime*: Die Simulationszeit in Nanosekunden.
- *Frequency*: Die Frequenz des Systems in Megahertz (MHz).
- *Cycles*: Die Anzahl der durchgeführten Zyklen während der Simulation.

x86

- *RealTime*: Die tatsächlich benötigte Zeit für die x86-Simulation in Sekunden.

- *Instructions*: Die Gesamtzahl der ausgeführten Instruktionen während der x86-Simulation.
- *InstructionsPerSecond*: Die Anzahl der Instruktionen pro Sekunde.
- *SimTime*: Die Simulationszeit für x86 in Nanosekunden.
- *Frequency*: Die Frequenz des x86-Prozessors in Megahertz (MHz).
- *Cycles*: Die Anzahl der durchgeführten Zyklen während der x86-Simulation.
- *CyclesPerSecond*: Die Anzahl der Zyklen pro Sekunde.
- *FastForwardInstructions*: Die Anzahl der Instruktionen, die im Schnellvorlauf übersprungen wurden.
- *CommittedInstructions*: Die Anzahl der ausgeführten (committed) Instruktionen.
- *CommittedInstructionsPerCycle*: Die Anzahl der ausgeführten Instruktionen pro Zyklus.
- *CommittedMicroInstructions*: Die Anzahl der ausgeführten (committed) Mikroinstruktionen.
- *CommittedMicroInstructionsPerCycle*: Die Anzahl der ausgeführten Mikroinstruktionen pro Zyklus.
- *BranchPredictionAccuracy*: Die Genauigkeit der Verzweigungsvorhersage (Branch Prediction Accuracy).

5.6 OVPsim

Die OVPsim-Hardwaresimulation wird von Imperas gemäß [45] entwickelt. Ohne dass eine Kompilierung, aufwendige Anpassungen oder externe Abhängigkeiten erforderlich sind, funktioniert die Hardwaresimulation mit dem Standardwert. Außerdem kann der Standardwert überschrieben werden.

Eingabe: Diese Konfigurationsparameter nach [45] dienen dazu, das Verhalten des RISC-V-Prozessormodells in der riscvOVPsimPlus-Simulation anzupassen. In der Tabelle 5.6 werden einige dieser Parameter gefunden.

Parameter	**Wert**	**Beschreibung**
riscvOVPsim/cpu/variant	RV32I	wählt die Variante des Prozessormodells aus.
riscvOVPsim/cpu/user_version	20190305	gibt die erforderliche User-Architekturversion an.
riscvOVPsim/cpu/priv_version	20190405	gibt die erforderliche Privileged-Architekturversion an.
Fortsetzung auf der nächsten Seite		

Tabelle 5.6 – Fortsetzung von der vorherigen Seite

Parameter	Wert	Beschreibung
riscvOVPsim/cpu/mstatus_fs_mode	write_1	legt fest, wann mstatus.FS als dirty markiert wird.
riscvOVPsim/cpu/debug_mode	none	definiert die Implementierung des Debug-Modus.
riscvOVPsim/cpu/verbose	F	gibt an, ob detaillierte Ausgabemeldungen angezeigt werden sollen.
riscvOVPsim/cpu/numHarts	0	legt die Anzahl der Hart-Kontexte in einem Multiprozessor fest.
riscvOVPsim/cpu/updatePTEA	F	gibt an, ob die Hardware-Aktualisierung des PTE-A-Bits unterstützt wird.
riscvOVPsim/cpu/updatePTED	F	gibt an, ob die Hardware-Aktualisierung des PTE-D-Bits unterstützt wird.
riscvOVPsim/cpu/unaligned	F	gibt an, ob nicht ausgerichtete Speicherzugriffe unterstützt werden.
riscvOVPsim/cpu/unalignedAMO	F	gibt an, ob nicht ausgerichtete Speicherzugriffe für AMO-Anweisungen unterstützt werden.
riscvOVPsim/cpu/wfi_is_nop	F	legt fest, ob WFI als NOP behandelt werden soll.
riscvOVPsim/cpu/mtvec_is_ro	F	gibt an, ob mtvec-CSR schreibgeschützt ist.
riscvOVPsim/cpu/tvec_align	0	legt die hardwaregesteuerte Ausrichtung von mtvec/stvec/utvec im Vectored Interrupt-Modus fest.

Tabelle 5.6: Beschreibung der Parameter für die OVPsim-Hardwaresimulation

Ausgabe: Nach Beendigung einer Simulation werden nach [45] die folgenden Ergebnisse und Statistiken angezeigt:

CPU-STATISTIKEN für „riscvOVPsim/cpu“:

- Typ
- Nominale MIPS
- Endgültiger Program Counter
- Simulierte Anweisungen
- Simulierte MIPS

STATISTIKEN zur SIMULATIONSZEIT:

- Simulierte Zeit
- Benutzerzeit
- Systemzeit
- Verstrichene Zeit
- Echtzeit-Verhältnis

5.7 SESC

Eingabe: Es wird empfohlen, das Konfigurationsskript aus dem Build-Verzeichnis heraus auszuführen, da in diesem Fall standardmäßige Konfigurationseinstellungen verwendet werden, sofern keine weiteren Optionen angegeben werden.
Die Konfigurationsdatei für die SESC-Hardwaresimulation enthält gemäß [13] die folgenden Parameter, die in Tabelle 5.7 dargestellt sind.

Parameter	**Wert**	**Beschreibung**
Freq	5e9 Hz	Taktfrequenz des Prozessors
DataL1Size	8 * 1024	Größe des L1-Daten-Caches
DataL1Assoc	4	Assoziativität des L1-Daten-Caches
InstL1Size	16 * 1024	Größe des L1-Instruktions-Caches
InstL1Assoc	4	Assoziativität des L1-Instruktions-Caches
L2Size	32 * 1024	Größe des L2-Caches
L2Assoc	8	Assoziativität des L2-Caches
RASsize	32	realistische Größe des Registrierungsfensters
issue	3	Anzahl der gleichzeitig auszuführenden Instruktionen
nCPUs	1	Anzahl der verwendeten CPUs
AdvMemMap	M3TMemMap	fortgeschrittene Speicherabbildung
BlkSize	32	Blockgröße
LDUnits	0	Anzahl der Last-/Speichereinheiten
UseTLS	0	Verwendung von Thread-Local Storage (TLS)

Tabelle 5.7: Parameter nach [13] für die SESC-Hardwaresimulation

Ausgabe: Der Ausgabebericht enthält nach [14] Informationen zu verschiedenen Aspekten wie Simulationsgeschwindigkeit, Ausführungszeit, Simulationszeit, Return Address Stack, Branch Predictor, Branch Target Buffer, Anweisungen pro Zyklus, Zyklen, Auslastung, Kontrollfenster, Struktur und Speicher.

5.8 Sniper

Sniper nach [85] besteht aus drei unterschiedlichen Phasen, nämlich „Ausführung“, „Konfiguration“ wie in Tabelle 5.8 und „Simulationsergebnisse“.

Eingabe: In Tabelle 5.8 sind gemäß [18] die Parameter für die Hardwaresimulation aufgeführt.

Parameter	**Wert**
Frequenz	1 GHz
Sockets per system	4
Cores per socket	6
Dispatch width	4 Mikrooperationen
Reorder buffer	96 Einträge
Branch predictor	Pentium M [28]
Cache line size	64 B
L1-I Cachegröße	32 KB
L1-I Assoziativität	8-fach set-assoziativ
L1-I Latenz	3 Zyklen Datenzugriff, 1 Zyklus Tag-Zugriff
L1-D Cachegröße	32 KB
L1-D Assoziativität	8-fach set-assoziativ
L1-D Latenz	3 Zyklen Datenzugriff, 1 Zyklus Tag-Zugriff
L2 Cachegröße	3 MB pro 2 Kerne
L2 Assoziativität	12-fach set-assoziativ
L2 Latenz	14 Zyklen Datenzugriff, 3 Zyklen Tag-Zugriff
L3 Cachegröße	16 MB pro 6 Kerne
L3 Assoziativität	16-fach set-assoziativ
L3 Latenz	96 Zyklen Datenzugriff, 10 Zyklen Tag-Zugriff
Coherence Protocol	MSI
Main Memory	200 ns Zugriffszeit
Memory Bandwidth	4 GB/s

Tabelle 5.8: Parameter [18] für die Sniper-Hardwaresimulation

Ausgabe: Bei den Simulationsergebnissen gibt es nach [85] zum Beispiel verschiedene Dateien, die generiert werden:

- `sim.cfg`: Diese Datei enthält alle Konfigurationsoptionen, die für die aktuelle Ausführung verwendet wurden. Dies umfasst Standardwerte sowie alle `-c-` und `-g-`Optionen.
- `sim.out`: In dieser Datei werden grundlegende Statistiken über die Simulation bereitgestellt. Dazu gehören Informationen wie die Anzahl der durchlaufenden Zyklen, die Anzahl der ausgeführten Anweisungen pro Kern sowie Zugriffs- und Fehlerraten des Caches.
- `sim.stats[.sqlite3]`: Diese Datei enthält einen umfassenden Satz von aufgezeichneten Statistiken zu wichtigen Punkten in der Simulation. Dazu gehören Startzeit, Beginn und Ende des Bereichs von Interesse (ROI) sowie Stopzeit.

5.9 ZSim++

Pin ist erforderlich, um die dynamische Binärübersetzung DBT für OOO-Kerne durchzuführen.
Eingabe: Gemäß [67] stellt die Konfiguration für die Hardwaresimulation in Tabelle 5.9 dar.

Komponente	**Parameter**	**Wert**
HW	Prozessor	Xeon L5640 (6-Core Westmere)
	Speicher	24 GB DDR3-1333
	Hyperthreading	Deaktiviert
	Turbo/DVFS	Deaktiviert
SW	Betriebssystem	Linux 3.5 x86-64
	Compiler	gcc 4.6.2
	Pin-Version	2.12
Bound-weave	Intervalle	1000 Zyklen
	Weave-Threads	6
Cores	Anzahl der Kerne	6 x86-64 OOO-Kerne
	Taktfrequenz	2,27 GHz
L1I-Caches	Größe	32 KB
	Assoziativität	4-fach
	Ersatzstrategie	LRU (Least Recently Used)
	Latenz	3 Zyklen
L1D-Caches	Größe	32 KB
	Assoziativität	8-fach
	Ersatzstrategie	LRU (Least Recently Used)
	Latenz	4 Zyklen
L2-Caches	Größe	256 KB
	Assoziativität	8-fach
	Ersatzstrategie	LRU (Least Recently Used)
	Latenz	7 Zyklen
L3-Cache	Größe	12 MB
	Assoziativität	16-fach
	Ersatzstrategie	Gehasht
	Anzahl der 2 MB Bänke	6
	Banklatenz	14 Zyklen
	MSHRs (Memory Request Handlers)	16
Network	Typ	Ring
	Zyklus pro Hop	1
	Einspritzlatenz	5 Zyklen
Mem Ctrl	Anzahl der Controller	1
	Anzahl der DDR3-Kanäle	3
	Scheduling	FCFS
	Powerdown mit Schwellwert-Timer	15 Mem-Zyklen
DRAM	Größe	24 GB
	Typ	DDR3-1333

Fortsetzung auf der nächsten Seite

Tabelle 5.9 – Fortsetzung von der vorherigen Seite

Komponente	Parameter	Wert
	RDIMMs pro Kanal	2x 4GB

Tabelle 5.9: Parameter [67] für die ZSim++

Ausgabe: Die Statistikausgabe beinhaltet die Anzahl der Hits in einer bestimmten L2-Cache-Bank. Es wird auch die Gesamtanzahl der Hits in allen L2-Caches berechnet und ausgegeben. Des Weiteren wird die Gesamtzahl der ausgeführten Anweisungen berechnet, indem die Anzahl der Anweisungen pro Kern summiert wird.

5.10 Klassifizierung

#	Klasse	Eingabeparameter	Ausgabeparameter	Eigenschaften	Ziele (ISA)
1	ESESC, Flexus, McSimA+ und SESC	#Cores, Frequenz, L1D, L1I und L2 Caches	IPC, Caches, Zyklen, #Instruktion	TIM, UM, FSys	-
2	gem5, Sniper und ZSim	#Cores, Frequenz, L1D, L1I und L2 Caches	IPC, Caches, Zyklen, #Instruktion, Host-Zeit, Sim-Zeit,	TIM, UM, FSys, MOD	x86
3	Multi2Sim und OVPsim	#Cores, Frequenz, L1D, L1I und L2 Caches	Sim-Zeit, #Instruktion, Host-Zeit	TIM, UM, FSys, MOD, FUNC	ARM, x86, MIPS

Tabelle 5.10: Klassen von Hardwaresimulationen

Nachdem die Hardwaresimulationen anhand der Eingabe- und Ausgabeparameter vorgestellt wurden, können sie gemäß Tabelle 5.10 in drei Klassen eingeteilt werden.
Die Klassifizierung erfolgte in mehreren Schritten, beginnend mit der Betrachtung der Eingabe- und Ausgabeparameter, dann unter Berücksichtigung der Zielsetzung, um sicherzustellen, dass eine Klasse alle möglichen Kombinationen von Eigenschaften abdeckt. Schließlich erfolgte die Klassifizierung bezüglich des Zielsystems, da diese Entscheidung auf der Notwendigkeit basiert, die Hardwaresimulationsergebnisse angemessen bewerten zu können.

Die erste Klasse zeichnet sich durch eine hohe Anzahl von Eingabeparametern aus, während die zweite Klasse sowohl bei den Eingabe- als auch bei den Ausgabeparametern eine ähnliche Anzahl von Parametern aufweist. Zum Beispiel haben Flexus und gem5 beide dieselbe Zielsetzung, nämlich FSys und TIM (Zyklusebene), und sie zielen auch beide auf das X86-Zielsystem ab. Die Unterschiede zwischen den beiden liegen jedoch in der Anzahl der Parameter pro Komponente. Bei Cache L1i hat Flexus beispielsweise 23 Parameter, während gem5 nur 2 Parameter hat. Da gem5 eine angemessene Anzahl von Parametern pro Komponente aufweist, ist es besser geeignet, sich mit anderen Hardwaresimulationen in der zweiten Klasse zu klassifizieren. Die letzte Klasse umfasst Hardwaresimulationen mit einer begrenzten Anzahl an Ausgabeparametern.

In der **ersten Klasse** sind vier Hardwaresimulationen vertreten: ESESC, Flexus, McSimA+ und SESC. Diese Klasse legt den Fokus sowohl auf die Eingabeparameter als auch auf die Hardwaresimulationsergebnisse. Bei den Eingabeparametern erfordern diese Hardwaresimulationen eine Vielzahl von Parametern, von denen viele für verschiedene Hardwaresimulationen gemeinsam verwendet werden können, wie zum Beispiel Anzahl der Kerne, die Frequenz und die L1- und L2-Caches. Es gibt jedoch auch spezifische Eingabeparameter, die nur für bestimmte Hardwaresimulationen relevant sind. Ähnlich verhält es sich mit den Ausgabeparametern bzw. Hardwaresimulationsergebnissen, bei denen es ebenfalls viele gemeinsame Parameter gibt, wie zum Beispiel IPC, Zyklen und Instruktionen. ESESC unterstützt die Timing-Funktion auf Zyklen-Ebene und den User Mode, was eine detaillierte Analyse und eine gute Geschwindigkeit ermöglicht. Flexus hingegen unterstützt den Full System und die Timing-Funktion. McSimA+ bietet die Timing-Funktion auf Zyklen-Ebene und den User Mode, was eine hohe Genauigkeit und eine ausreichende Geschwindigkeit gewährleistet. SESC bietet den UM und die Timing-Funktion auf Zyklen-Ebene, was eine hohe Genauigkeit und eine schnellere Ausführung ermöglicht. In dieser Klasse werden sowohl die Genauigkeit als auch die Geschwindigkeit berücksichtigt, wobei der Full System zur Verfügung steht.
In dieser Klasse gibt es kein gemeinsames Zielsystem, da jede Hardwaresimulation ein unterschiedliches Zielsystem bietet.

Die **zweite Klasse** umfasst verschiedene Hardwaresimulationen: gem5, Sniper und ZSim. In dieser Klasse liegt der Fokus sowohl auf den Eingabeparametern als auch auf den Hardwaresimulationsergebnissen. Denn auch in diesem Fall verwenden alle Hardwaresimulationen eine ähnliche Anzahl von Eingabeparametern, von denen viele gemeinsam betrachtet werden können. Es gibt jedoch auch spezifische Parameter, die nur in bestimmten Simulationen vorhanden sind. Bei den Ausgabeparametern lassen sich viele gemeinsame Hardwaresimulationsergebnisse ableiten. gem5 unterstützt den Full System und die Timing-Funktion auf Zyklen-Ebene, was eine detaillierte Analyse ermöglicht. Sniper ermöglicht parallelen User Mode und die Timing-Funktion, wodurch eine Balance zwischen Genauigkeit und Geschwindigkeit erreicht wird. ZSim++ ermöglicht die parallele Timing-Funktion und den User Mode, was eine ausgewogene Balance zwischen Genauigkeit und Geschwindigkeit bietet. In dieser Klasse werden die Anforderungen an Genauigkeit und Geschwindigkeit je nach Anwendungsszenario abgewogen, wobei der Full System angeboten wird.
In dieser Klasse besteht ein gemeinsames Zielsystem, da jede Hardwaresimulation dasselbe Zielsystem anbietet, nämlich x86. Diese Kategorie kann dazu verwendet werden, ähnliche Zielsysteme mit unterschiedlichen Zielsetzungen abzubilden.

In der **dritten Klasse** sind die Hardwaresimulationen Multi2Sim und OVPsim vertreten. In dieser Klasse liegt der Fokus auf den Hardwaresimulationsergebnissen, da die Hardwaresimulationen nur eine begrenzte Anzahl von Hardwaresimulationsergebnissen liefern. Daher lassen sich auch nur begrenzt gemeinsame Parameter ableiten. In Bezug auf die Eingabeparameter haben alle Hardwaresimulationen in dieser Klasse eine ähnliche Anzahl von Parametern, von denen viele gemeinsam verwendet werden können. Es gibt jedoch auch spezifische Parameter, die nur für bestimmte Simulationen relevant sind. Multi2Sim bietet den User Mode und die Timing-Funktion, was sowohl Genauigkeit als auch Geschwindigkeit garantiert. OVPsim konzentriert sich auf die Functional-Simulation und den Full System. In dieser Klasse werden sowohl die Genauigkeit als auch die Geschwindigkeit basierend auf den spezifischen Anforderungen berücksichtigt, wobei der Full System zur Verfügung steht.

In dieser Kategorie sind mehrere gemeinsame Zielsysteme vertreten, da jede Hardwaresimulation dieselben Zielsysteme anbietet, nämlich ARM, x86 und MIPS. Diese Klasse kann dazu verwendet werden, unterschiedliche Zielsysteme mit vielfältigen Zielsetzungen abzubilden.

6 Allgemeinen Schnittstelle

Nachdem Hardwaresimulationen in Klassen wie in Tabelle 5.10 aus Abschnitt 5.10 eingeteilt wurden, kann eine allgemeine Schnittstelle für alle Hardwaresimulationen einer Klasse entwickelt werden, um den Austausch von Hardwaresimulationen zu vereinfachen und die Wartbarkeit zu verbessern. Um eine allgemeine Schnittstelle für eine Klasse zu implementieren, fiel die Auswahl auf die zweite Klasse. Diese liegt daran, dass alle Hardwaresimulationen in dieser Klasse den X86-Architektur-Host unterstützen und auch die Möglichkeit bieten, das x86-Zielsystem zu emulieren. Diese Entscheidung basiert auf der Notwendigkeit, die Durchführbarkeit der Hardwaresimulation zu gewährleisten und die Ergebnisse der Simulation angemessen bewerten zu können. Darüber hinaus enthält die zweite Klasse die gem5-Hardwaresimulation, die kontinuierlich weiterentwickelt wird und verschiedene Aspekte wie Leistung und Geschwindigkeit abdeckt. Das Ziel besteht darin, eine allgemeine Schnittstelle zu entwerfen, die alle Schnittstellen der Hardwaresimulationen in dieser Klasse abbildet.

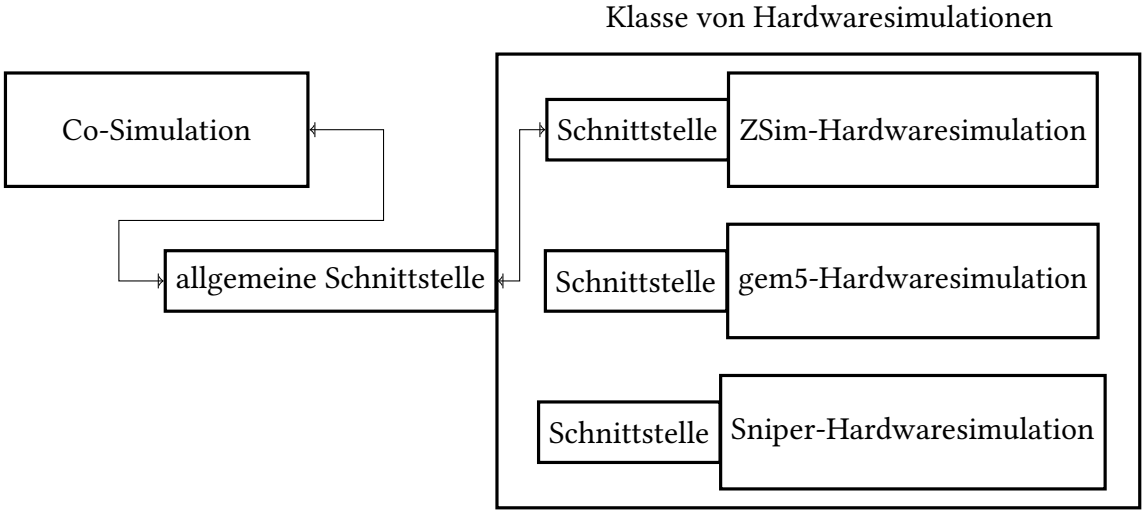

Abbildung 6.1: Allgemeine Schnittstelle zur Klasse von Hardwaresimulationen

Die Abbildung 6.1 präsentiert eine übersichtliche Darstellung einer allgemeinen Schnittstelle, die speziell für eine Klasse von Hardwaresimulationen entwickelt wurde. Im Zentrum der Abbildung befindet sich die allgemeine Schnittstelle, die als Hauptkomponente dient. Jede dieser Hardwaresimulationen – gem5, Sniper und ZSim – besitzt ihre eigene Schnittstelle, die mit der allgemeinen Schnittstelle verbunden ist. Die Co-Simulation nutzt die allgemeine Schnittstelle, um eine Hardwaresimulation anzubinden, die einfach austauschbar sein soll. Das Hauptziel dieser Entwicklung besteht darin, eine allgemeine Schnittstelle zu entwerfen, die den Zugriff auf alle Schnittstellen der Hardwaresimulationen ermöglicht. Dabei werden sowohl die

Eingabe- als auch das Hardwaresimulationsergebnis berücksichtigt. Die allgemeine Schnittstelle dient als ein einheitlicher Zugangspunkt, der es ermöglicht, auf die Funktionen und Eigenschaften zuzugreifen, die alle Hardwaresimulationen innerhalb einer Klasse bereitstellen können. Dadurch wird die Effizienz beim Zugriff auf die verschiedenen Hardwaresimulationen erhöht, indem eine einzige Schnittstelle verwendet wird, anstatt spezifische Schnittstellen für jede Hardwaresimulation zu verwenden. Dies führt zu einer erhöhten Flexibilität und Wiederverwendbarkeit, da die Implementierung und Interaktion zwischen der Co-Simulation und den Hardwaresimulationen vereinheitlicht wird. Dadurch ist es möglich, ohne Änderungen an der Co-Simulation einfach zu einer anderen Hardwaresimulation in dieser Klasse zu wechseln.

6.1 Entwicklung der allgemeinen Schnittstelle

Es ist wichtig, dass die allgemeine Schnittstelle klar definiert ist und dass sie auf relevante Eigenschaften der Hardwaresimulationen in der Klasse ausgerichtet ist, um den Austausch von Hardwaresimulationen zu ermöglichen.

6.1.1 Eingabeparameter

Um eine allgemeine Schnittstelle für Hardwaresimulationen der zweiten Klasse zu entwickeln, sind in Tabelle 6.1 die erforderlichen Parameter aufgeführt. Da verschiedene Hardwaresimulationen unterschiedliche Eingabeparameter haben, kann es schwierig sein, eine allgemeine Schnittstelle zu entwickeln, die für alle Hardwaresimulationen in einer Klasse geeignet ist. Daher wird eine allgemeine Schnittstelle entwickelt, die die Schnittmenge aller möglichen Eingabeparameter der Hardwaresimulationen abdeckt und spezifische Parameter für jede einzelne Hardwaresimulation enthält, die in den anderen Hardwaresimulationen nicht vorhanden sind. Darüber hinaus wird die Schnittstelle so konzipiert, dass die spezifischen Parameter einer bestimmten Hardwaresimulation eindeutig zugeordnet werden können. Das Hauptziel besteht darin, sämtliche Parameter in dieser allgemeinen Schnittstelle zu erfassen, um eine einheitliche Ausführung der drei Hardwaresimulationen über eine allgemeine Schnittstelle zu ermöglichen. Zusätzlich soll bei einer Änderung an einem Parameter deutlich erkennbar sein, ob es sich um einen allgemeinen Parameter handelt oder ob er spezifisch einer bestimmten Hardwaresimulation zugeordnet ist.

Component	**Parameter**	**Wert**
Common Parameters	hardwaresimulation	gem5
	programPath	D:/fibonacci.c
	binaryPath	D:/fibonacci
	statsOutputPath	D:/outputStats
	cache_hierarchy.l1d_size	32KiB
	cache_hierarchy.l1d_assoc	8
	cache_hierarchy.l1i_size	32KiB
	cache_hierarchy.l1i_assoc	8
	cache_hierarchy.l2_size	256KiB
Fortsetzung auf der nächsten Seite		

Tabelle 6.1 – Fortsetzung von der vorherigen Seite

Component	Parameter	Wert
	cache_hierarchy.l2_assoc	16
	board.frequency	3GHz
zsim	lineSize	64
	caches.l1d.latency	4
	caches.l1i.latency	3
	caches.l2.mshrs	10
	caches.l2.latency	4
gem5	memory.size	2GiB
sniper	perf_model/core.logical_cpus	1
	perf_model/itlb.size	48
	perf_model/itlb.associativity	48
	perf_model/dtlb.size	48
	perf_model/dtlb.associativity	48
	perf_model/stlb.size	128
	perf_model/stlb.associativity	4
	perf_model/dram_directory.total_entries	1048576
	perf_model/dram_directory.associativity	16

Tabelle 6.1: Allgemeine Schnittstelle Parameter für zweite Klasse

Die Tabelle 6.1 zeigt eine Liste von Parametern, die für verschiedene Hardwaresimulationen wie gem5, ZSim und Sniper relevant sind. Diese Parameter steuern unterschiedliche Aspekte der Hardwaresimulationsumgebung und haben jeweils eine spezifische Bedeutung. Hier ist eine Erklärung für jeden dieser Parameter:

Common Parameters (Gemeinsame Parameter): umfasst eine Kategorie von Parametern, die in der allgemeinen Schnittstelle definiert sind und von verschiedenen Hardwaresimulationen innerhalb derselben Klasse gemeinsam genutzt werden. Diese Parameter dienen der allgemeinen Konfiguration und haben für alle Simulationen innerhalb dieser Klasse dieselbe Bedeutung und Funktion. Dazu gehören Informationen über den Pfad zur C-Datei, den Pfad zur ausführbaren Datei, den Pfad zur Ausgabe der Statistikdaten und die Größe und Assoziativität der Caches in der Hierarchie.

- *hardwaresimulation*: Hier wird der Name der verwendeten Hardwaresimulation angegeben, in diesem Fall gem5.
- *programPath*: Der Pfad zur C-Datei „fibonacci.c“, die als Eingabe für die Hardwaresimulation dient. Diese Datei enthält den Quellcode eines Programms, das in der Simulation ausgeführt wird und die ersten 30 Fibonacci-Zahlen berechnet.
- *binaryPath*: Der Pfad zur ausführbaren Datei „fibonacci“, die während der Simulation verwendet wird. Diese Datei wird aus dem Quellcode der „fibonacci.c“-Datei kompiliert und enthält den maschinenlesbaren Code, der von der Hardwaresimulation ausgeführt wird.

- *statsOutputPath*: Dieser Parameter enthält den Pfad zum Speichern der Simulationsergebnisse der Hardwaresimulation. Die Simulationsergebnisse enthalten wichtige Informationen, wie in Unterabschnitt 6.1.2 erläutert, über die Leistung und das Verhalten der Hardwaresimulation während der Ausführung der ausführbaren Datei („binary datei“). Die gesammelten Simulationsergebnisse werden in der Datei mit dem Namen „generatestatsOutput.json“ im Verzeichnis „outputStats“ gespeichert, und der angegebene Pfad zeigt den Ort an, an dem diese Datei abgelegt wird.
- *cache_hierarchy.l1d_size*: Die Größe des Level-1 Daten-Caches wird hier angegeben, in diesem Fall 32 Kilobyte (KiB).
- *cache_hierarchy.l1d_assoc*: Die Assoziativität des Level-1 Daten-Caches wird hier angegeben, in diesem Fall 8.
- *cache_hierarchy.l1i_size*: Die Größe des Level-1 Instruktions-Caches wird hier angegeben, in diesem Fall 32 KiB.
- *cache_hierarchy.l1i_assoc*: Die Assoziativität des Level-1 Instruktions-Caches wird hier angegeben, in diesem Fall 8.
- *cache_hierarchy.l2_size*: Die Größe des Level-2 Caches wird hier angegeben, in diesem Fall 256 Kilobyte (KiB).
- *cache_hierarchy.l2_assoc*: Die Assoziativität des Level-2 Caches wird hier angegeben, in diesem Fall 16.
- *board.frequency*: Die Frequenz der CPU, die in diesem Fall auf 3 GHz eingestellt ist. Dies bestimmt die Geschwindigkeit, mit der der Prozessor arbeitet.

zsim:

- *lineSize*: Die Größe einer Zeile (Line) in den Caches. Dies bestimmt, wie viele Bytes in einer Cache-Zeile gespeichert werden können und beeinflusst die Cache-Effizienz, in diesem Fall 64 Byte.
- *caches.l1d.latency*: Die Latenz ist die Zeitspanne, die benötigt wird, um auf Daten aus dem Cache zuzugreifen. In diesem Fall wird die Latenz des Level-1 Daten-Caches angegeben und beträgt 4 Takte.
- *caches.l1i.latency*: Die Latenz des Level-1 Instruktions-Caches wird hier angegeben, in diesem Fall 3 Takte.
- *caches.l2.mshrs*: Die Anzahl der maximalen MSHRs (Memory Request Handlers) im Level-2 Cache wird hier angegeben, in diesem Fall 10.
- *caches.l2.latency*: Die Latenz des Level-2 Caches wird hier angegeben, in diesem Fall 4 Takte.

gem5:

- *memory.size*: Die Größe des Hauptspeichers wird hier angegeben, in diesem Fall 2 Gibibyte (GiB).

sniper:

- *perf_model/core.logical_cpus*: Die Anzahl der logischen CPUs (Threads) im Kern wird hier angegeben, in diesem Fall 1.
- *perf_model/itlb.size*: Die Größe des Instruktionsübersetzungspuffers (Instruction Translation Lookaside Buffer - ITLB) wird hier angegeben, in diesem Fall 48 Einträge.
- *perf_model/itlb.associativity*: Die Assoziativität des ITLB wird hier angegeben, in diesem Fall 48.
- *perf_model/dtlb.size*: Die Größe des Datenübersetzungspuffers (Data Translation Lookaside Buffer - DTLB) wird hier angegeben, in diesem Fall 48 Einträge.
- *perf_model/dtlb.associativity*: Die Assoziativität des DTLB wird hier angegeben, in diesem Fall 48.
- *perf_model/stlb.size*: Die Größe des Seitentranslationspuffers (STLB) wird hier angegeben, in diesem Fall 128 Einträge.
- *perf_model/stlb.associativity*: Die Assoziativität des STLB wird hier angegeben, in diesem Fall 4.
- *perf_model/dram_directory.total_entries*: Die Gesamtanzahl der Einträge im DRAM- Verzeichnis wird hier angegeben, in diesem Fall 1.048.576 Einträge.
- *perf_model/dram_directory.associativity*: Die Assoziativität des DRAM-Verzeichnisses wird hier angegeben, in diesem Fall 16.

Darüber hinaus verfügt jede Hardwaresimulation über spezifische Standardwerte, die in anderen Hardwaresimulationen möglicherweise nicht auftreten. Die Variation dieser Standardwerte erfordert ein tiefes Verständnis der jeweiligen Hardwaresimulation, was jedoch in dieser Arbeit nicht weiter untersucht wurde. Darüber hinaus war es aufgrund von Zeitbeschränkungen nicht möglich, die detaillierte Variation dieser Standardwerte zu berücksichtigen. Der Fokus lag hauptsächlich auf den allgemeinen Parametern, die in verschiedenen Hardwaresimulationen vorhanden sind. Infolgedessen werden diese spezifischen Standardwerte für jede Hardwaresimulation vorgestellt.

Zu beachten ist, dass ZSim über einen Cache L3 als Standardwerte verfügt, während gem5 standardmäßig keinen L3-Cache besitzt. Jedoch kann ein L3-Cache in gem5 hinzugefügt werden. Im Fall von Sniper gibt es ebenfalls einen L3-Cache, jedoch funktionierte die Konfiguration dieses L3-Caches nicht wie erwartet und war somit nicht funktionsfähig. Deshalb wurde die Verwendung des L3-Caches ausgeschlossen. In einigen Fällen hängt die Auswahl der Parameter auch von der Technik der jeweiligen Hardwaresimulation ab. Ein Beispiel hierfür ist Sniper, bei dem nur zwei bestimmte Konfigurationsdateien ordnungsgemäß funktionierten, und diese beiden Konfigurationen beinhalteten keinen L3-Cache.

Komponente	Parameter	Wert
simulation	domains	1
	phaseLength	1000
	statsPhaseInterval	10000
system	caches.l1d.array.type	SetAssoc
	caches.l1d.caches	1
	caches.l1d.latency	4
	caches.l1i.array.type	SetAssoc
	caches.l1i.caches	1
	caches.l1i.latency	3
	caches.l2.array.type	SetAssoc
	caches.l2.type	Timing
	caches.l2.mshrs	10
	caches.l2.caches	1
	caches.l2.latency	4
	caches.l2.children	l1i|l1d
	caches.l3.array.hash	H3
	caches.l3.array.type	SetAssoc
	caches.l3.ways	16
	caches.l3.type	Timing
	caches.l3.mshrs	16
	caches.l3.banks	32
	caches.l3.caches	1
	caches.l3.latency	27
	caches.l3.children	l2
	caches.l3.size	67108864
	cores.skylake.cores	1
	cores.skylake.dcache	l1d
	cores.skylake.icache	l1i
	cores.skylake.type	OOO
	networkType	mesh
	networkFile	network_32.mesh
memory	addrMapping	rank:col:bank
	tech	DDR4-2400-CL17
	type	DDR
	mem.splitAddrs	false
	mem.closedPage	true
	mem.controllerLatency	40
	mem.controllers	6

Tabelle 6.2: Standardwert für die Parameter in ZSim

Die Tabelle 6.2 zeigt die Standardwerte für verschiedene Parameter in der ZSim-Hardwaresimulation. Hier ist eine Erklärung für jeden dieser Parameter und ihre entsprechenden Standardwerte:

simulation:

- *domains*: Gibt die Anzahl der Simulationsdomänen an. In diesem Fall wird nur eine Simulationsdomäne verwendet.
- *phaseLength*: Definiert die Länge einer Phase in der Simulation. Eine Phase ist ein Zeitabschnitt, in dem bestimmte Simulationseinstellungen unverändert bleiben. Hier beträgt die Länge einer Phase 1000 Simulationsschritte.
- *statsPhaseInterval*: Gibt an, wie oft Statistiken in der Simulation gesammelt werden. In diesem Fall werden alle 10.000 Simulationsschritte Statistiken erfasst.

system:

- *caches.l1d.array.type*: Legt den Typ des Level-1 Daten-Cache-Arrays fest. Hier wird eine „Set Associative“ (SA) Struktur verwendet.
- *caches.l1d.caches*: Gibt die Anzahl der Level-1 Daten-Caches an. In diesem Fall wird ein einzelner Level-1 Daten-Cache verwendet.
- *caches.l1d.latency*: Definiert die Latenzzeit des Level-1 Daten-Caches. Hier beträgt die Latenzzeit 4 Einheiten.
- *caches.l1i.array.type*: Legt den Typ des Level-1 Instruktions-Cache-Arrays fest. Hier wird ebenfalls eine „Set Associative“ (SA) Struktur verwendet.
- *caches.l1i.caches*: Gibt die Anzahl der Level-1 Instruktions-Caches an. In diesem Fall wird ein einzelner Level-1 Instruktions-Cache verwendet.
- *caches.l1i.latency*: Definiert die Latenzzeit des Level-1 Instruktions-Caches. Hier beträgt die Latenzzeit 3 Einheiten.
- *caches.l2.array.type*: Legt den Typ des Level-2 Cache-Arrays fest. Hier wird ebenfalls eine „Set Associative“ (SA) Struktur verwendet.
- *caches.l2.type*: Legt den Typ des Level-2 Caches fest. Hier wird der Cache-Typ „Timing“ verwendet.
- *caches.l2.mshrs*: Gibt die Anzahl der „Miss Status Handling Registers“ (MSHRs) im Level-2 Cache an. Hier beträgt die Anzahl 10.
- *caches.l2.caches*: Gibt die Anzahl der Level-2 Caches an. In diesem Fall wird ein einzelner Level-2 Cache verwendet.
- *caches.l2.latency*: Definiert die Latenzzeit des Level-2 Caches. Hier beträgt die Latenzzeit 4 Einheiten.
- *caches.l2.children*: Gibt die Kinder-Caches des Level-2 Caches an. In diesem Fall sind es die Level-1 Instruktions-Caches und Level-1 Daten-Caches.

- *caches.l3.array.hash*: Legt den Hash-Typ für das Level-3 Cache-Array fest. Hier wird „H3“ verwendet.
- *caches.l3.array.type*: Legt den Typ des Level-3 Cache-Arrays fest. Hier wird eine „Set Associative“ (SA) Struktur verwendet.
- *caches.l3.ways*: Gibt die Anzahl der Wege (ways) im Level-3 Cache an. Hier beträgt die Anzahl 16.
- *caches.l3.type*: Legt den Typ des Level-3 Caches fest. Hier wird der Cache-Typ „Timing“ verwendet.
- *caches.l3.mshrs*: Gibt die Anzahl der „Miss Status Handling Registers“ (MSHRs) im Level-3 Cache an. Hier beträgt die Anzahl 16.
- *caches.l3.banks*: Gibt die Anzahl der Banken im Level-3 Cache an. Hier beträgt die Anzahl 32.
- *caches.l3.caches*: Gibt die Anzahl der Level-3 Caches an. In diesem Fall wird ein einzelner Level-3 Cache verwendet.
- *caches.l3.latency*: Definiert die Latenzzeit des Level-3 Caches. Hier beträgt die Latenzzeit 27 Einheiten.
- *caches.l3.children*: Gibt die Kinder-Caches des Level-3 Caches an. In diesem Fall ist es der Level-2 Cache.
- *caches.l3.size*: Gibt die Größe des Level-3 Caches an. Hier beträgt die Größe 67108864 Bytes.
- *cores.skylake.cores*: Gibt die Anzahl der Skylake-Kerne an. Hier wird ein einzelner Kern verwendet. *Skylake* ist nach [35] der Codename für eine Intel-Prozessormikroarchitektur ab 2015, die in Computern und Laptops genutzt wurde. Sie bietet Leistungsverbesserungen und unterstützt moderne Technologien wie DDR4-Speicher und Thunderbolt 3-Schnittstellen.
- *cores.skylake.dcache*: Gibt den Level-1 Daten-Cache (D-Cache) für den Skylake-Kern an. Hier ist es der Level-1 Daten-Cache.
- *cores.skylake.icache*: Gibt den Level-1 Instruktions-Cache (I-Cache) für den Skylake-Kern an. Hier ist es der Level-1 Instruktions-Cache.
- *cores.skylake.type*: Legt den Typ des Skylake-Kerns fest. Hier wird der Kern-Typ OOO verwendet.
- *networkType*: Legt den Typ des Netzwerks fest. Hier wird ein „Mesh“ verwendet. Ein Mesh ist ein Netzwerktyp, bei dem alle Knoten direkt miteinander verbunden sind, ohne eine zentrale Steuereinheit.
- *networkFile*: Gibt die Netzwerkdatei an. Hier wird die Datei „network_32.mesh“ verwendet.

memory:

- *addrMapping*: Legt die Adresszuordnung für den Speicher fest. Hier wird die Zuordnung „rank:col:bank“ verwendet.
- *tech*: Legt die Speichertechnologie fest. Hier wird „DDR4-2400-CL17“ verwendet. Es ist die vierte Generation der DDR-Speichertechnologie mit einer Taktfrequenz von 2400 MHz und einer CAS-Latenz von 17 Takten
- *type*: Legt den Speichertyp fest. Hier wird „DDR“ verwendet.
- *mem.splitAddrs*: Gibt an, ob die Adressen im Speicher getrennt sind. Hier ist es „false“.
- *mem.closedPage*: Gibt an, ob die Seiten im Speicher geschlossen sind. Hier ist es „true“.
- *mem.controllerLatency*: Definiert die Latenzzeit des Speichercontrollers. Hier beträgt die Latenzzeit 40 Einheiten.
- *mem.controllers*: Gibt die Anzahl der Speichercontroller an. Hier werden 6 Speichercontroller verwendet.

Komponente	**Parameter**	**Wert**
general	enable_icache_modeling	true
perf_model/core	core_model	nehalem
	type	interval
perf_model/core/interval_timer	dispatch_width	2
	window_size	32
	num_outstanding_loadstores	10
perf_model/sync	reschedule_cost	1000
caching_protocol	type	parametric _dram_directory_msi
perf_model/branch_predictor	type	pentium_m
	mispredict_penalty	8
perf_model/tlb	penalty	30
perf_model/cache	levels	2
perf_model/l1_icache	perfect	false
	address_hash	mask
	replacement_policy	lru
	data_access_time	4
	tags_access_time	1
	perf_model_type	parallel
	writethrough	0
	shared_cores	1

Fortsetzung auf der nächsten Seite

Tabelle 6.3 – Fortsetzung von der vorherigen Seite

Komponente	Parameter	Wert
perf_model/l1_dcache	perfect	false
	address_hash	mask
	replacement_policy	lru
	data_access_time	4
	tags_access_time	1
	perf_model_type	parallel
	writethrough	0
	shared_cores	1
perf_model/l2_cache	perfect	false
	address_hash	mask
	replacement_policy	lru
	data_access_time	12
	tags_access_time	3
	writeback_time	50
	perf_model_type	parallel
	writethrough	0
	shared_cores	2
	dvfs_domain	global
	prefetcher	none
perf_model/dram_directory	directory_type	full_map
perf_model/dram	num_controllers	-1
	controllers_interleaving	4
	latency	45
	per_controller_bandwidth	12.8
	chips_per_dimm	8
	dimms_per_controller	4
network	memory_model_1	bus
	memory_model_2	bus
network/bus	bandwidth	128
	ignore_local_traffic	false
clock_skew_minimization	scheme	barrier
clock_skew_minimization/ barrier	quantum	100
dvfs	transition_latency	2000
dvfs/simple	cores_per_socket	1
power	vdd	1.0
	technology_node	22

Tabelle 6.3: Standardwert für die Parameter in Sniper

Die Tabelle 6.3 zeigt die Standardwerte für verschiedene Parameter in der Sniper-Hardwaresimulation. Hier ist eine Erklärung für jeden dieser Parameter und ihre entsprechenden Standardwerte:

- *enable_icache_modeling*: Gibt an, ob das Modellieren des Instruktionscaches aktiviert ist (true) oder nicht (false).

perf_model/core:

- *core_model*: Definiert das Kernmodell für die Leistungsbewertung. Hier wird das „nehalem“ Kernmodell verwendet. „Nehalem“ ist nach [34] eine Mikroarchitektur von Intel-Prozessoren, die im Jahr 2008 eingeführt wurde. Diese Mikroarchitektur zeichnete sich durch verbesserte Leistung pro Taktzyklus, die Unterstützung der Hyper-Threading-Technologie und einen integrierten Speichercontroller aus. Diese Eigenschaften trugen zu einer erhöhten Rechenleistung und Effizienz der Prozessoren bei.

- *type*: Gibt den Typ des Parameters an, in diesem Fall „interval“. Die Interval-Simulation ist nach [36] ein innovativer Simulationsansatz für Mehrkern- und Multiprozessorsysteme, der eine höhere Abstraktionsebene als herkömmliche zyklengenaue Simulationen verwendet. Dabei wird ein mechanistisches analytisches Modell genutzt, um die Kernleistung abstrakt zu erfassen, indem Fehlereignisse in Intervalle aufgeteilt werden. Dies ermöglicht eine effizientere Modellierung der Leistungsverflechtung zwischen Threads auf Mehrkernprozessoren. Der Simulation nutzt Fenster von Anweisungen, um Fehlereignisse zu identifizieren und die zeitliche Abfolge für jeden Kern abzuleiten, was zu einer präziseren und rechenintensiven Simulation führt.

perf_model/core/interval_timer: Definiert den Intervall-Timer für das Kernmodell.

- *dispatch_width*: Gibt die Breite des Ausgabe-Dispatchers an und bestimmt, wie viele Instruktionen gleichzeitig aus dem Front-End in das Backend verschickt werden können (hier 2).

- *window_size*: Gibt die Größe des Ausführungsfensters an, das die Anzahl der Instruktionen begrenzt, die sich gleichzeitig im Backend befinden können (hier 32).

- *num_outstanding_loadstores*: Gibt die Anzahl der ausstehenden Lade-/ Speicheranforderungen an, die das System verarbeiten kann, bevor die Leistung beeinträchtigt wird (hier 10).

- *perf_model/sync reschedule_cost*: Definiert die Kosten für die Umplanung von Befehlen aufgrund von Synchronisationsoperationen (hier 1000).

- *caching_protocol type*: Gibt den Typ des Caching-Protokolls an, hier „parametric_dram _directory_msi“.

perf_model/branch_predictor:

- *type*: Definiert den Typ des Zweigvorhersagemodells, hier „pentium_m“.

- *mispredict_penalty*: Gibt die Strafzeit für einen Fehlervorhersage an (hier 8).

- *perf_model/tlb penalty*: Definiert die Strafzeit für den Zugriff auf die Translation Lookaside Buffer (hier 30).

- *perf_model/cache levels*: Gibt die Anzahl der Cache-Ebenen an, hier 2.

perf_model/l1_icache:

- *perfect*: Definiert das Modell für den Befehlscache der ersten Ebene (L1 ICache), hier false.
- *address_hash mask*: Definiert die Adresshash-Maske für den Cache.
- *replacement_policy*: Gibt die Austauschrichtlinie für den Cache an, hier LRU.
- *data_access_time*: Gibt die Zugriffszeit auf Cache-Daten in Zyklen an, hier 4.
- *tags_access_time*: Gibt die Zugriffszeit auf Cache-Tags in Zyklen an, hier 1.
- *perf_model_type*: Definiert den Leistungsmodelltyp, hier „parallel“.
- *writethrough*: Gibt an, ob der Schreibvorgang direkt in den Speicher erfolgt (hier 0, was wahrscheinlich Schreibrückhaltung bedeutet).
- *shared_cores*: Gibt die Anzahl der gemeinsam genutzten Kerne an, hier 1.

perf_model/l1_dcache:

- *perfect*: Definiert das Modell für den Datencache der ersten Ebene (L1 DCache), hier false.
- *address_hash mask*: Definiert die Adresshash-Maske für den Cache.
- *data_access_time*: Gibt die Zugriffszeit auf Cache-Daten in Zyklen an, hier 4.
- *tags_access_time*: Gibt die Zugriffszeit auf Cache-Tags in Zyklen an, hier 1.
- *writethrough*: Gibt an, ob der Schreibvorgang direkt in den Speicher erfolgt (hier 0, was wahrscheinlich Schreibrückhaltung bedeutet).
- *shared_cores*: Gibt die Anzahl der gemeinsam genutzten Kerne an, hier 1.

perf_model/l2_cache:

- *perfect*: Definiert das Modell für den Cache der zweiten Ebene (L2 Cache), hier false.
- *address_hash mask*: Definiert die Adresshash-Maske für den Cache.
- *data_access_time*: Gibt die Zugriffszeit auf Cache-Daten in Zyklen an, hier 12.
- *tags_access_time*: Gibt die Zugriffszeit auf Cache-Tags in Zyklen an, hier 3.
- *writeback_time*: Gibt die Schreib-Rückholzeit in Zyklen an, hier 50.
- *writethrough*: Gibt an, ob der Schreibvorgang direkt in den Speicher erfolgt (hier 0, was Schreibrückhaltung bedeutet).
- *shared_cores*: Gibt die Anzahl der gemeinsam genutzten Kerne an, hier 2.

- *dvfs_domain*: Gibt das Domain-Modell für Dynamic Voltage and Frequency Scaling (DVFS) an, hier „global“.
- *prefetcher*: Gibt den Typ des Prefetchers an, hier „none“ (kein Prefetcher aktiviert).
- *perf_model/dram_directory directory_type*: Definiert den Typ des DRAM-Verzeichnisses, hier „full_map“.

perf_model/dram:

- *num_controllers*: Gibt die Anzahl der DRAM-Controller an, hier -1.
- *controllers_interleaving*: Gibt an, wie viele DRAM-Controller miteinander verflochten sind, um den Speicherzugriff zu optimieren (hier 4).
- *latency*: Gibt die Speicherlatenz in Zyklen an, hier 45.
- *per_controller_bandwidth*: Gibt die Bandbreite pro DRAM-Controller in GB/s an, hier 12.8.
- *chips_per_dimm*: Gibt die Anzahl der Chips pro DIMM-Modul an, hier 8.
- *dimms_per_controller*: Gibt die Anzahl der DIMM-Module pro DRAM-Controller an, hier 4.

network:

- *memory_model_1*: Gibt den Typ des Speichermodells für das Netzwerk an, hier „bus“.
- *memory_model_2*: Gibt den Typ des Speichermodells für den Speicher 2 an, hier „bus“.

network/bus:

- *bandwidth*: Definiert die Bandbreite des Speicherbusses in GB/s, hier 128.
- *ignore_local_traffic*: Gibt an, ob der lokale Datenverkehr ignoriert werden soll (false bedeutet, dass er nicht ignoriert wird).
- *clock_skew_minimization scheme*: Definiert das Schema zur Minimierung der Taktverzerrung, hier „barrier“.
- *clock_skew_minimization/quantum*: Gibt die Zeitquanten für das Barriereschema zur Taktverzerrung an, hier 100.
- *dvfs transition_latency*: Gibt die Latenz für den DVFS-Übergang in Zyklen an, hier 2000.
- *dvfs/simple cores_per_socket*: Gibt die Anzahl der Kerne pro CPU-Sockel an, hier 1.

power:

- *vdd*: Gibt die Versorgungsspannung in Volt an, hier 1.0.
- *technology_node*: Gibt den Technologieknoten für die Herstellung des Prozessors an, hier 22 nm.

Komponente	Parameter	Wert
cache_hierarchy	MESITwoLevelCacheHierarchy.num _l2_banks	1
processor	SimpleSwitchableProcessor.num_cores	2
	starting_core_type	TIMING
	switch_core_type isa	O3 X86

Tabelle 6.4: Standardwert für die Parameter in gem5

Die Tabelle 6.4 zeigt die Standardwerte für verschiedene Parameter in der gem5-Hardwaresimulation. Hier ist eine Erklärung für jeden dieser Parameter und ihre entsprechenden Standardwerte:

cache_hierarchy: Definiert die Cache-Hierarchie des Systems.

- *MESITwoLevelCacheHierarchy.num_l2_banks*: Das System nutzt eine Cache-Hierarchie mit zwei Ebenen, die dem MESI-Protokoll folgt. Dieser Parameter gibt die Anzahl der Banken (Segmente) im Level-2-Cache an, wobei der Standardwert 1 beträgt.

processor: Enthält Informationen über den Prozessor im System.

- *SimpleSwitchableProcessor.num_cores*: Hier wird die Anzahl der Prozessorkerne im System festgelegt. Der Standardwert ist 2, was bedeutet, dass das System zwei Prozessorkerne hat.
- *starting_core_type*: Definiert den Typ des Startkerns im System. Im Standardwert wird der Startkern mit dem Timing-Modell TIM gestartet.
- *switch_core_type*: Hier wird der Typ des Kerns festgelegt, der verwendet wird, nachdem der Startkern beendet ist oder ausgetauscht wird. Im Standardwert wird der O3-Kern (OOO) verwendet.
- *isa*: Gibt die Befehlssatzarchitektur ISA des Systems an. Der Standardwert ist X86, was darauf hinweist, dass das System die x86-Befehlssatzarchitektur verwendet.

6.1.2 Hardwaresimulationsergebnisse

Es ist von äußerster Wichtigkeit, eine einheitliche Benennung und Einheit der Hardwaresimulationsergebnisse in allen Hardwaresimulationen dieser Kategorie sicherzustellen. Bestimmte Hardwaresimulationen wie zsim und gem5 liefern eine umfangreiche und detaillierte Menge an Hardwaresimulationsergebnissen, die bei der Betrachtung mit Sniper-Hardwaresimulationsergebnissen zusammengefasst werden können. Daher erfolgt eine sorgfältige Auswahl, bei der nur die Hardwaresimulationsergebnis berücksichtigt werden, die in allen Hardwaresimulationen vorhanden sind. Dabei wird die Schnittmenge dieser Parameter berücksichtigt und gegebenenfalls werden Parameter hinzugefügt, die aus anderen Parametern berechnet oder abgeleitet werden können. Dieser Vorgehensweise liegt das Ziel zugrunde, eine klare und vergleichbare Analyse über verschiedene Hardwaresimulationen hinweg zu ermöglichen. Durch

die einheitliche Betrachtung der wichtigsten Hardwaresimulationsergebnisse wird eine effiziente Interpretation der Hardwaresimulationsergebnisse ermöglicht, ohne von einer übermäßigen Detailfülle überfordert zu werden. In der Tabelle 6.5 werden Hardwaresimulationsergebnis präsentiert.

Parameter	**Wert**
Cycles	945300
HostNanoseconds	1416438958
IPC	0.45
Instructions	421185
Time (ns)	315100
Cache Summary.Cache L1D.miss rate	2.83%
Cache Summary.Cache L1D.mpki	10.62
Cache Summary.Cache L1D.num cache accesses	158295
Cache Summary.Cache L1D.num cache misses	4474
Cache Summary.Cache L1I.miss rate	3.79%
Cache Summary.Cache L1I.mpki	4.52
Cache Summary.Cache L1I.num cache accesses	50205
Cache Summary.Cache L1I.num cache misses	1903
Cache Summary.Cache L2.miss rate	79.23%
Cache Summary.Cache L2.mpki	12.17
Cache Summary.Cache L2.num cache accesses	6470
Cache Summary.Cache L2.num cache misses	5126

Tabelle 6.5: Allgemeine Schnittstelle Hardwaresimulationsergebnis für zweite Klasse

- *Cycles*: Dieser Parameter gibt die Anzahl der simulierten Zyklen an und beträgt 945300.
- *HostNanoseconds*: Hier wird die simulierte Zeit in Nanosekunden für das Host-System angegeben und beträgt 1416438958 ns (Nanosekunden).
- *IPC*: Dieser Parameter steht für Ïnstructions Per Cycle"(Anweisungen pro Zyklus) und beträgt 0.45. Es gibt an, wie viele Anweisungen durchschnittlich pro Zyklus ausgeführt werden.
- *Instructions*: Hier wird die Gesamtzahl der ausgeführten Anweisungen angegeben, die 421185 beträgt.
- *Time (ns)*: Dieser Parameter gibt die Gesamtzeit in Nanosekunden an, die für die Simulation verwendet wurde, und beträgt 315100 ns (Nanosekunden).

Cache Summary: enthält Informationen über die Cache-Leistung in einem System. Sie umfasst die Miss-Raten, die Anzahl der Cache-Zugriffe und die Anzahl der Cache-Miss-Zugriffe für die Level-1-Daten-Cache (L1D), den Level-1-Instruktions-Cache (L1I) und den Level-2-Cache (L2). Die angegebenen Miss-Raten zeigen den Prozentsatz der fehlgeschlagenen Cache-Zugriffe im Vergleich zur Gesamtzahl der Zugriffe auf den entsprechenden Cache. Die Werte geben wichtige

Einblicke in die Effizienz der Cache-Hierarchie und ihre Auswirkungen auf die Leistung des Systems.

- *Cache L1D.miss rate*: Dieser Parameter gibt die Fehlerrate (Miss-Rate) des Level-1-Daten-Caches an, die bei 2.83% liegt. Dies bedeutet, dass etwa 2.83% der Zugriffe auf den Level-1-Daten-Cache fehlschlagen und die benötigten Daten nicht im Cache gefunden werden.
- *Cache L1D.mpki*: Dieser Parameter steht für "Misses Per Kilo Instructions"(Fehlzugriffe pro Tausend Anweisungen) und beträgt 10.62. Es gibt an, wie viele Cache-Miss-Zugriffe pro tausend ausgeführten Anweisungen im Level-1-Daten-Cache auftreten.
- *Cache L1D.num cache accesses*: Hier wird die Gesamtzahl der Cache-Zugriffe im Level-1-Daten-Cache angegeben, die 158295 beträgt.
- *Cache L1D.num cache misses*: Dieser Parameter gibt die Gesamtzahl der Cache-Miss-Zugriffe im Level-1-Daten-Cache an, die 4474 beträgt.
- *Cache L1I.miss rate*: Dieser Parameter gibt die Fehlerrate (Miss-Rate) des Level-1-Instruktions-Caches an, die bei 3.79% liegt. Dies bedeutet, dass etwa 3.79% der Zugriffe auf den Level-1-Instruktions-Cache fehlschlagen und die benötigten Instruktionen nicht im Cache gefunden werden.
- *Cache L1I.mpki*: Dieser Parameter steht für "Misses Per Kilo Instructions"(Fehlzugriffe pro Tausend Anweisungen) und beträgt 4.52. Es gibt an, wie viele Cache-Miss-Zugriffe pro tausend ausgeführten Anweisungen im Level-1-Instruktions-Cache auftreten.
- *Cache L1I.num cache accesses*: Hier wird die Gesamtzahl der Cache-Zugriffe im Level-1-Instruktions-Cache angegeben, die 50205 beträgt.
- *Cache L1I.num cache misses*: Dieser Parameter gibt die Gesamtzahl der Cache-Miss-Zugriffe im Level-1-Instruktions-Cache an, die 1903 beträgt.
- *Cache L2.miss rate*: Dieser Parameter gibt die Fehlerrate (Miss-Rate) des Level-2-Caches an, die bei 79.23% liegt. Dies bedeutet, dass etwa 79.23% der Zugriffe auf den Level-2-Cache fehlschlagen und die benötigten Daten nicht im Cache gefunden werden.
- *Cache L2.mpki*: Dieser Parameter steht für "Misses Per Kilo Instructions"(Fehlzugriffe pro Tausend Anweisungen) und beträgt 12.17. Es gibt an, wie viele Cache-Miss-Zugriffe pro tausend ausgeführten Anweisungen im Level-2-Cache auftreten.
- *Cache L2.num cache accesses*: Hier wird die Gesamtzahl der Cache-Zugriffe im Level-2-Cache angegeben, die 6470 beträgt.
- *Cache L2.num cache misses*: Dieser Parameter gibt die Gesamtzahl der Cache-Miss-Zugriffe im Level-2-Cache an, die 5126 beträgt.

7 Implementierung

Nachdem im Kapitel 6 die allgemeinen Eingabeparameter festgelegt wurden und die Ergebnisse der Hardwaresimulation definiert sind, richtet sich der Fokus nun auf die Implementierung der allgemeinen Schnittstelle. Dabei ist es von entscheidender Bedeutung, sicherzustellen, dass diese Implementierung erweiterbar und leicht wartbar ist. Zunächst wird ein übersichtliches Klassendiagramm für die gesamte Schnittstelle präsentiert. Anschließend werden separate Klassendiagramme für die Erstellung der Eingabeparameterdateien und für die Hardwaresimulationsergebnisse vorgestellt. Abschließend erfolgt eine Vorstellung der Gesamtstruktur des Softwareentwurfs für die allgemeine Schnittstelle.

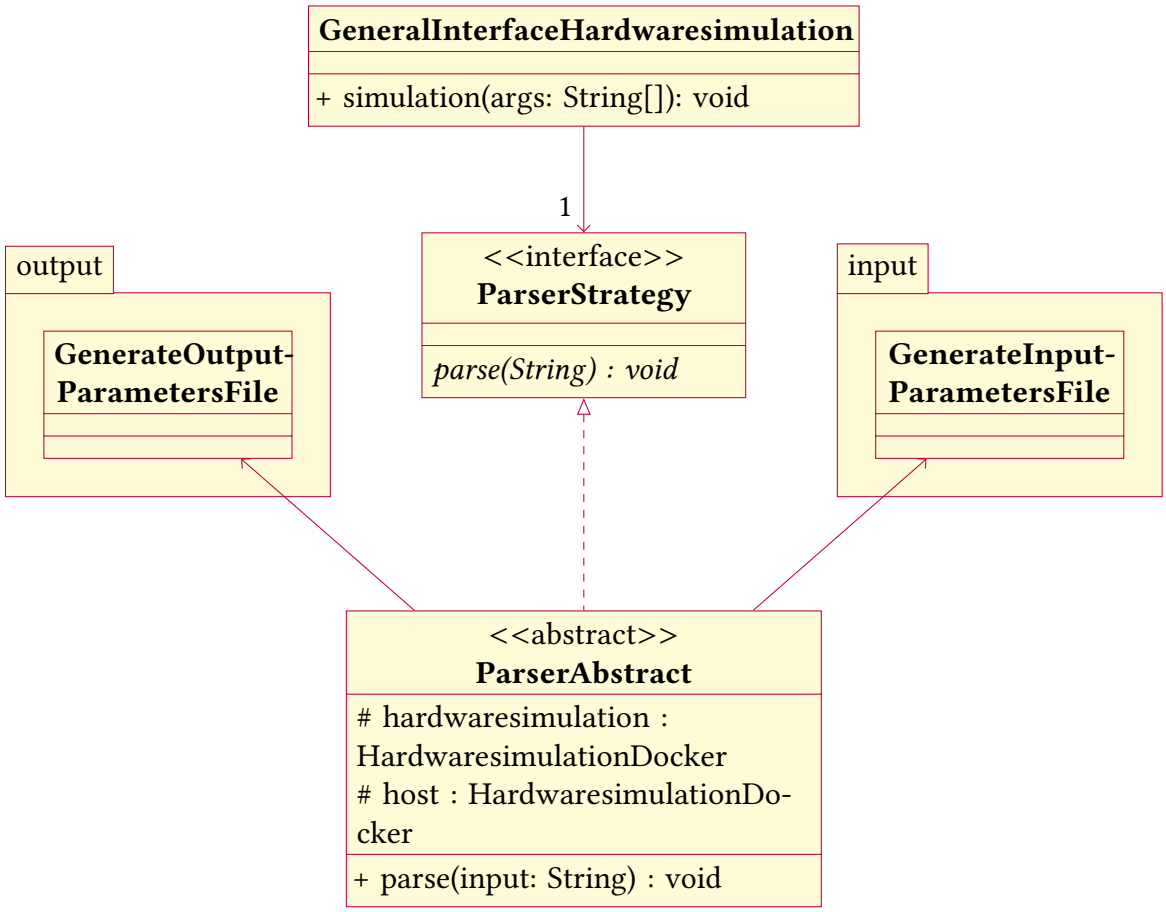

Abbildung 7.1: Klassendiagramm für allgemeine Schnittstelle der Hardwaresimulation

In Abbildung 7.1 wird ein umfassender Überblick über die Softwarearchitektur der allgemeinen Schnittstelle für die Hardwaresimulation vermittelt. Die Klasse `GeneralInterfaceHardwaresimulation` steht dabei im Mittelpunkt und fungiert als Simulationsschnitt-

stelle, die die Eingabedatei entgegennimmt und die Ausgabedatei erzeugt. Neben der `GeneralInterfaceHardwaresimulation` wird auch die Schnittstelle `Parser-Strategy` gezeigt, welche von der abstrakten Klasse `ParserAbstract` implementiert wird. Letztere verfügt über private Attribute wie `hardwaresimulation` und `host`, die als Instanzen der Klasse `HardwaresimulationDocker` fungieren.
Die Klassen `GenerateInputParametersFile` und `GenerateOutputParametersFile` gehören zu den Paketen `input` und `output` und beschreiben den Vorgang des Generierens der Eingabe- und Ausgabeparameterdateien.

7.1 Eingabeprameter

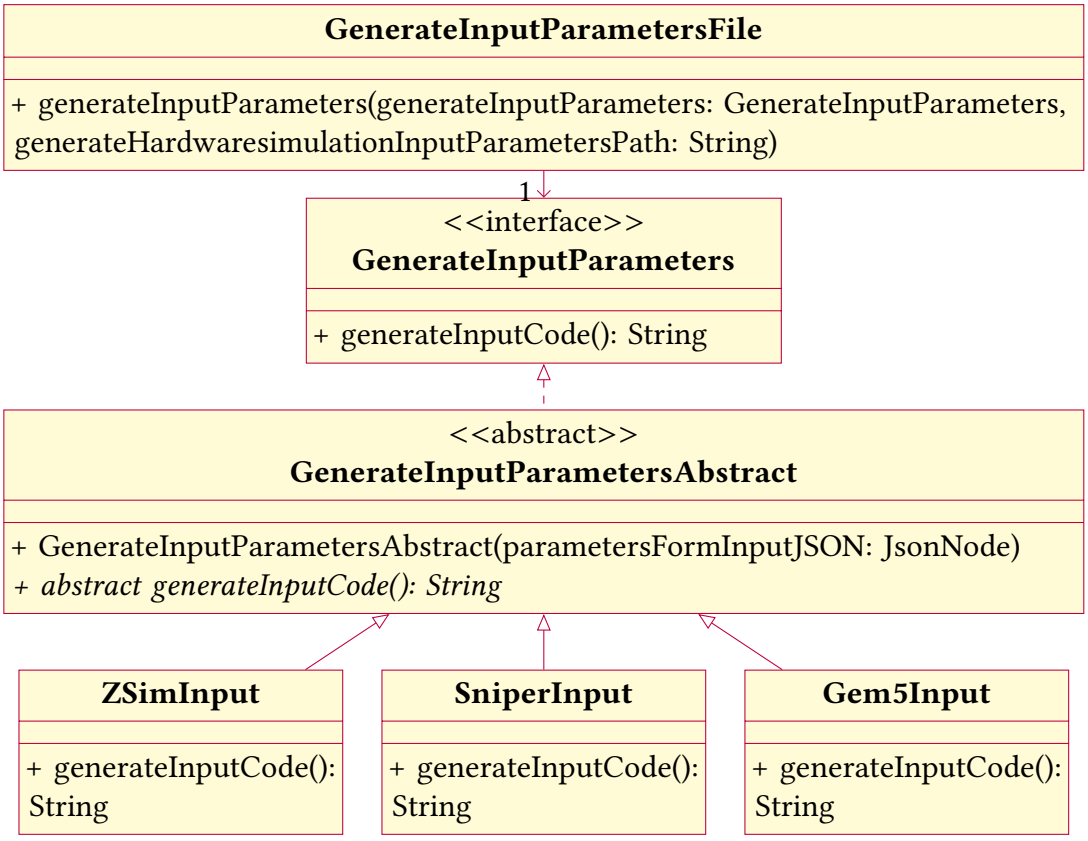

Abbildung 7.2: Erstellen von Eingabeparameterdateien

Das Klassendiagramm in Abbildung 7.2 veranschaulicht den Prozess der Erstellung von Eingabeparameterdateien für verschiedene Hardwaresimulationen. Die Klasse `GenerateInputParametersFile` übernimmt die Verantwortung für die Erzeugung der Eingabedatei. Sie greift auf die Methode `generateInputCode` der Schnittstelle `GenerateInputParameters` zu, Um spezifische Eingabeparameter für die entsprechende Hardwaresimulation aus einer JSON-Datei zu generieren, werden die Parameter gemäß Unterabschnitt 6.1.1 der Eingabe-JSON-Datei verwendet. Die Klasse `GenerateInputParametersAbstract` implementiert die Schnittstelle `GenerateInputParameters` und erweitert sie. Sie enthält die Methode `generateInputCode`, die

den spezifischen Eingabecode für die Hardwaresimulation generiert. Die Klassen `Gem5Input`, `SniperInput` und `ZSimInput` erben von `GenerateInputParametersAbstract` und implementieren die Methode `generateInputCode` gemäß den Anforderungen ihrer jeweiligen Hardwaresimulation, wie in Abbildung 7.3 dargestellt.

Listing 7.1: Input.json

```
"commonParameters": {
    "cache_hierarchy":{
        "l1d_size": "32KiB"
                    }
    "board":{
        "frequency": "3GHz"
                    }
                       }
```

Listing 7.2: SniperCfg

```
[perf_model/l1_dcache]
cache_size = 32

[perf_model/core]
frequency = 3
```

Listing 7.3: ZSimCfg

```
sys = {
    caches = {
        l1d = { size = 32768;
                }
            }
    frequency = 3000;
        }
```

Abbildung 7.3: Abbildung von 2 Input.json-Parametern in Sniper/ ZSim-Konfigurationsdateien

In der Abbildung 7.3 wird gezeigt, wie zwei Parameter aus der input.json-Datei in die Konfigurationsdateien für die Hardwaresimulationen Sniper und ZSim umgewandelt werden. Listing 7.1 stellt die allgemeine Schnittstelle für die Eingabeparameter dar, die in den Konfigurationsdateien für Sniper und ZSim abgebildet wird. Dabei werden nicht nur die Namen und Positionen der Parameter angepasst, sondern auch die Einheiten.
Der **l1d_size**-Parameter wird in der input.json-Datei 7.1 unter commonParameters mit dem Wert 32 KiB angegeben. In der Sniper-Konfigurationsdatei 7.2 wird der Parameter in cache_size umbenannt, und die Einheit KiB (Kilobyte) wird entfernt. Die Größe wird als numerischer Wert 32 in die Sniper-Konfigurationsdatei übernommen, da Sniper eine numerische Eingabe für die Cache-Größe ohne Einheit erwartet. In der ZSim-Konfigurationsdatei 7.3 wird der Parameter cache_hierarchy.l1d_size in Bytes umgerechnet und als size im Abschnitt 7.3 sys/caches/l1d mit dem Wert 32768 angegeben. Dies entspricht der Umrechnung von 32 KiB in Bytes ($32 \times 1024 = 32768$ Bytes), da ZSim die Cache-Größe in Byte erwartet. In der inputjson-Datei 7.1 ist der Parameter **boardfrequency** mit dem Wert 3GHz unter commonParameters angegeben. In der Sniper-Konfigurationsdatei 7.2 wird die Einheit GHz entfernt. Die Frequenz 3 wird direkt als numerischer Wert für die frequency in die Sniper-Konfigurationsdatei übernommen, da Sniper eine numerische Eingabe für die Frequenz ohne Einheit erwartet. In der ZSim-Konfigurationsdatei 7.3 wird der Parameter board.frequency in Megahertz umgerechnet und als frequency im Abschnitt sys mit dem Wert 3000 angegeben. Dies entspricht der Umrechnung von 3GHz in Megahertz ($3 \times 10^3 = 3000$ Megahertz), da ZSim die Frequenz in MHz erwartet.

7.2 Hardwaresimulationsergebnisse

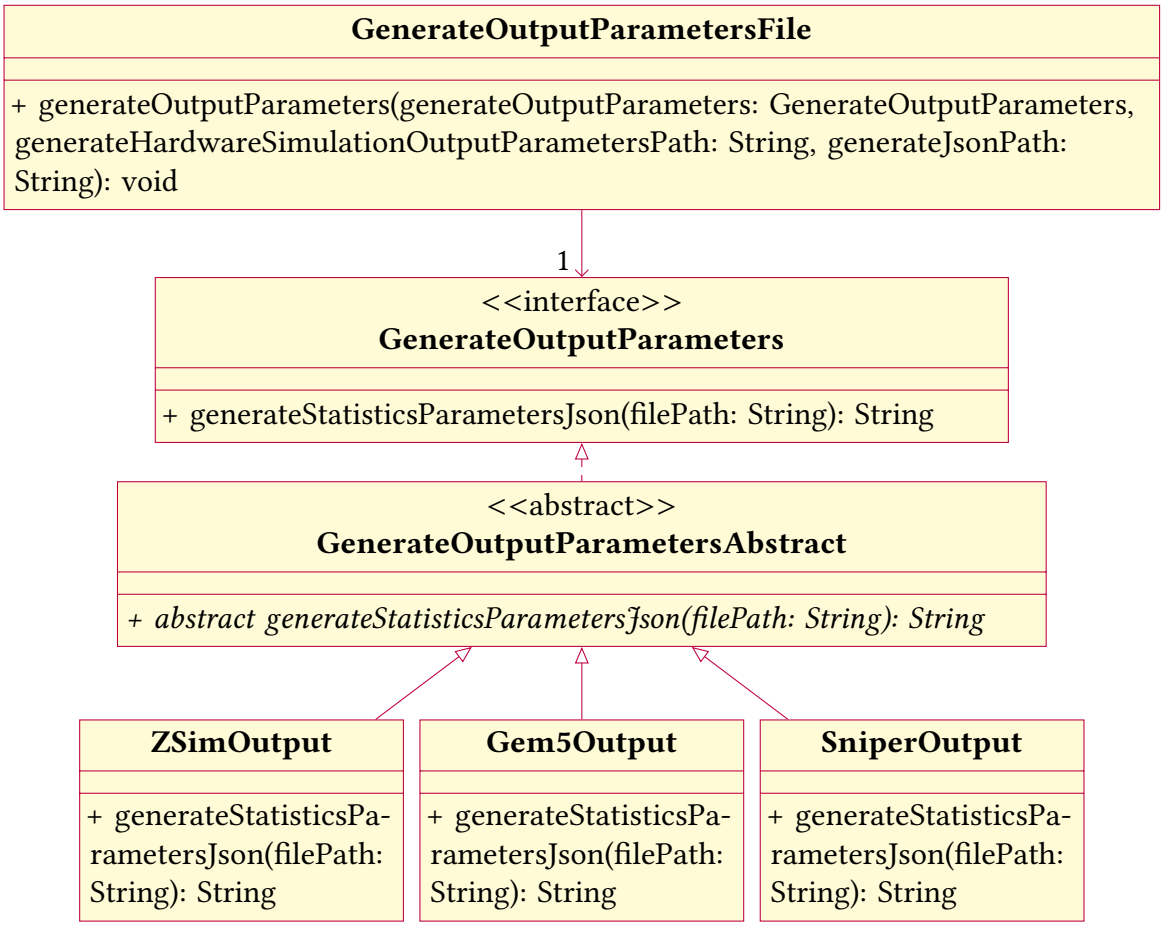

Abbildung 7.4: Ausgabeparameter

Das Klassendiagramm in Abbildung 7.4 veranschaulicht den Prozess des Erstellens von Hardwaresimulationsergebnisdateien für verschiedene Hardwaresimulationen. Die Schnittstelle `GenerateOutputParameters` definiert die Methode `generateStatisticsParametersJson`, um Hardwaresimulationsergebnisse im JSON-Format zu generieren. Die Klasse `GenerateOutputParametersFile` implementiert diese Schnittstelle und erzeugt Hardwaresimulationsergebnissedateien für verschiedene Hardwaresimulationen. Dabei werden spezifische Klassen wie `Gem5Output`, `ZSimOutput` und `SniperOutput` verwendet, um die Parameterwerte zu generieren. Jede dieser spezifischen Klassen implementiert die Methode `generateStatisticsParametersJson` und enthält ihre eigenen Parameterdaten. Durch die Verwendung der Schnittstelle und der spezifischen Klassen kann das Generieren von Hardwaresimulationsergebnissen einheitlich und flexibel gestaltet werden.

In Abbildung 7.5 und Abbildung 7.6 wird veranschaulicht, wie einige Ergebnisse der gem5- und ZSim- Hardwaresimulationen auf drei Output-Parametern abgebildet werden. In Listing 7.4 werden 3 Parameter aus den Hardwaresimulationsergebnissen dargestellt. Im Unterabschnitt 6.1.2 wird die gesamten Hardwaresimulationsergebnisse vorgestellt. Diese Parameter

Listing 7.4: Output.json

```
"Cache Summary":{
 "Cache L1-D":{
  "num cache accesses": 15175,
  "num cache misses": 233,
  "miss rate": "1.54%"
              }
                   }
```

Listing 7.5: ZSimO

```

l1d:
  l1d-0: # Filter cache stats
    fhGETS: 8307 # Filtered GETS hits
    fhGETX: 6045 # Filtered GETX hits
    hGETS: 325 # GETS hits
    hGETX: 265 # GETX hits
    mGETS: 116 # GETS misses
    mGETXIM: 117 # GETX I->M misses
    mGETXSM: 0 # GETX S->M misses (upgrade misses)
```

Abbildung 7.5: Darstellung von 3 Output.json-Parametern aus den Ergebnissen der ZSim-Hardwaresimulationen

können von drei verschiedenen Hardwaresimulationen in der zweiten Klasse erzeugt werden, indem die Ausgabedaten der Hardwaresimulation auf die Hardwaresimulationsergebnisse abgebildet werden. In Abbildung 7.5 wird gezeigt, wie Hardwaresimulationsergebnisse aus anderen Parametern berechnet oder abgeleitet werden können, die in der Ausgabe der Hardwaresimulation enthalten sind. Die gegebenen Cache-Statistiken in Listing 7.5 für ZSim-Ergebnisse beziehen sich auf den L1-Daten-Cache und enthalten Informationen über die Anzahl der GETS- und GETX-Zugriffe sowie deren Treffer (Hits) und Fehlzugriffe (Misses). Hier ist nach [80] eine detaillierte Erklärung für GETS und GETX:
GETS (Get Shared):

- GETS-Anforderungen werden verwendet, wenn ein Prozessor eine Leseoperation (Read) für einen bestimmten Speicherblock ausführt. Der Prozessor möchte den Speicherblock lesen, aber plant nicht, den Block zu ändern (keine Schreiboperation).
- Wenn der Speicherblock bereits in einem anderen Prozessor-Cache vorhanden ist und sich im gemeinsamen (Shared) Zustand befindet, wird die GETS-Anforderung erfolgreich behandelt.
- Die Statistik „fhGETS“ zeigt die Anzahl der erfolgreich gefilterten GETS-Hits im Filter-Cache an. Die FilterCache-Abstraktion wurde speziell für die Effizienz der Simulation entwickelt und ist in der realen Hardware nicht vorhanden. Es handelt sich um einen direkt abgebildeten Cache mit der gleichen Satzanzahl wie der zugrunde liegende L1-Cache, jedoch nur einem einzigen Weg pro Satz. In diesem direkt abgebildeten Array des L1-Caches wird lediglich der zuletzt verwendete Weg in jedem Satz gespeichert.
- Die Statistik „hGETS“ zeigt die Anzahl der erfolgreich getroffenen GETS-Anforderungen im L1-Daten-Cache an, bei denen der gesuchte Speicherblock bereits im Cache im Shared-Zustand vorhanden war.

- Die Statistik „mGETS“ zeigt die Anzahl der verpassten GETS-Anforderungen im L1-Daten-Cache an, bei denen der gesuchte Speicherblock nicht im Cache vorhanden war und daher aus dem Hauptspeicher geladen werden musste.

GETX (Get Exclusive):

- GETX-Anforderungen werden verwendet, wenn ein Prozessor eine Schreiboperation (Write) für einen bestimmten Speicherblock ausführt. Der Prozessor möchte den Speicherblock schreiben und benötigt eine exklusive Kopie des Blocks, um die Schreiboperation durchzuführen.

- Wenn der Speicherblock bereits in einem anderen Prozessor-Cache vorhanden ist, wird die GETX-Anforderung verwendet, um alle anderen Kopien des Blocks in den anderen Prozessor-Caches ungültig zu machen und den Block exklusiv im Cache zu erhalten.

- Die Statistik „fhGETX“ im Filter-Cache erfasst erfolgreich gefilterte GETX-Hits, ähnlich wie bei „fhGETS“. Sie zählt GETX-Anfragen, bei denen das gewünschte Element im Filter-Cache gefunden wurde, und vermeidet somit Zugriffe auf den Haupt-Cache oder den Hauptspeicher. Die FilterCache-Abstraktion wurde speziell entwickelt, um die Simulationseffizienz zu verbessern und ist in der realen Hardware nicht vorhanden. Sie basiert auf einem direkt abgebildeten Cache mit derselben Anzahl von Sätzen wie der L1-Cache, aber nur einem Weg pro Satz. Dabei wird lediglich der zuletzt verwendete Weg im Array gespeichert.

- Die Statistik „hGETX“ zeigt die Anzahl der erfolgreich getroffenen GETX-Anforderungen im L1-Daten-Cache an, bei denen der gesuchte Speicherblock bereits im Cache im exklusiven oder geteilten Zustand vorhanden war.

- Die Statistik „mGETXIM“ zeigt die Anzahl der verpassten GETX I->M-Anforderungen im L1-Daten-Cache an, bei denen der gesuchte Speicherblock sich im Cache im Shared-Zustand befand und für die Schreiboperation in den exklusiven Zustand überführt werden musste (Invalidation und Modifizierung).

- Die Statistik „mGETXSM“ zeigt die Anzahl der verpassten GETX S->M-Anforderungen (Upgrade Misses) im L1-Daten-Cache an, bei denen der gesuchte Speicherblock sich im Cache im geteilten Zustand befand und für die Schreiboperation in den exklusiven Zustand überführt werden musste.

Zusammenfassend geben diese Statistiken Aufschluss über die Anzahl der Cache-Hits (hGETS und hGETX) und die Anzahl der Cache-Misses (mGETS, mGETXIM und mGETXSM) im L1-Daten-Cache für GETS- und GETX-Anforderungen.
Die Werte fhGETX, fhGETS, hGETS, hGETX, mGETS, mGETXIM und mGETXSM aus ZSim-Ergebnissen in 7.5 können zu **num cache accesses** addiert werden, da sie die Anzahl der Cache-Zugriffe für verschiedene Zugriffstypen im L1-Daten-Cache darstellen.

Die Werte L1Dcache.m_demand_accesses aus gem5-Ergebnissen in 7.7 für den L1-Daten-Cache in den beiden Cores (l1_controllers0 und l1_controllers1) können ebenfalls zu **num**

Listing 7.6: Output.json

```
"Cache Summary":{
 "Cache L1-D":{
  "num cache accesses": 49922,
  "num cache misses": 2060,
  "miss rate": "4.13%"
              }
                   }
```

Listing 7.7: gem5Txt

```
board.cache_hierarchy.ruby_system:{
    l1_controllers0.L1Dcache:{
        m_demand_accesses       49922
        m_demand_misses         2060
        },
    l1_controllers1.L1Dcache:{
        m_demand_accesses       0
        m_demand_misses         0
        }
}
```

Abbildung 7.6: Darstellung von 3 Output.json-Parametern aus den Ergebnissen der gem5-Hardwaresimulationen

cache accesses addiert werden, da sie die Anzahl der Cache-Zugriffe für jeden Core darstellen, Dabei wird die Anzahl der Cores in Tabelle 6.4 als Standardwerte festgelegt.
Dabei werden gem5 und ZSim Hardwaresimulationen durchgeführt, um die Ausführung der Binärdatei zur Berechnung der ersten 30 Fibonacci-Zahlen zu simulieren.
Die Ergebnisse der Hardwaresimulation werden in der Output-Datei als **num cache accesses** abgebildet. Dabei werden die entsprechenden Werte für jede Hardwaresimulation berechnet und in die Output-Datei eingetragen. Beachte, dass bei der Implementierung von Fibonacci nicht mehrere Kerne verwendet werden, sondern nur ein einzelner Kern. Daher ist einer der beiden Cores in gem5 null und kann zusammengefasst werden. Für **num cache misses** in der Output-Datei können die Werte aus den gem5- und ZSim-Ergebnissen wie folgt abgebildet werden. Der Wert m_demand_misses für den L1-Daten-Cache in den beiden Cores (l1_controllers0.L1Dcache und l1_controllers1.L1Dcache) stellt die Anzahl der Cache-Misses für jeden Core dar. Die Gesamtanzahl der Cache-Misses für den L1-Daten-Cache in gem5 7.7 ist daher die Summe der Werte von m_demand_misses für beide Cores. Die Werte mGETS, mGETXIM und mGETXSM stellen die Anzahl der Cache-Misses für verschiedene Zugriffstypen im L1-Daten-Cache dar. Die Gesamtanzahl der Cache-Misses für den L1-Daten-Cache in ZSim 7.5 ist daher die Summe der Werte von mGETS, mGETXIM und mGETXSM.
In den Hardwaresimulationsergebnissen von gem5 und ZSim sind keine expliziten Werte für die **miss rate** angegeben. Allerdings kann die miss rate für gem5 und ZSim berechnet werden, da aus den Ergebnissen beider Hardwaresimulationen die Anzahl der Cache-Misses und die Gesamtanzahl der Cache-Zugriffe bekannt sind. Die miss rate wird mit Hilfe der folgenden Formel berechnet:

$$\text{miss rate } (\%) = \frac{\text{num cache misses}}{\text{num cache accesses}} \times 100$$

Durch Anwenden dieser Formel auf die Hardwaresimulationsergebnisse von gem5 und ZSim kann die miss rate für beideHardwaresimulationen berechnet werden.

7.3 Struktur des gesamten Softwareentwurfs für die allgemeine Schnittstelle

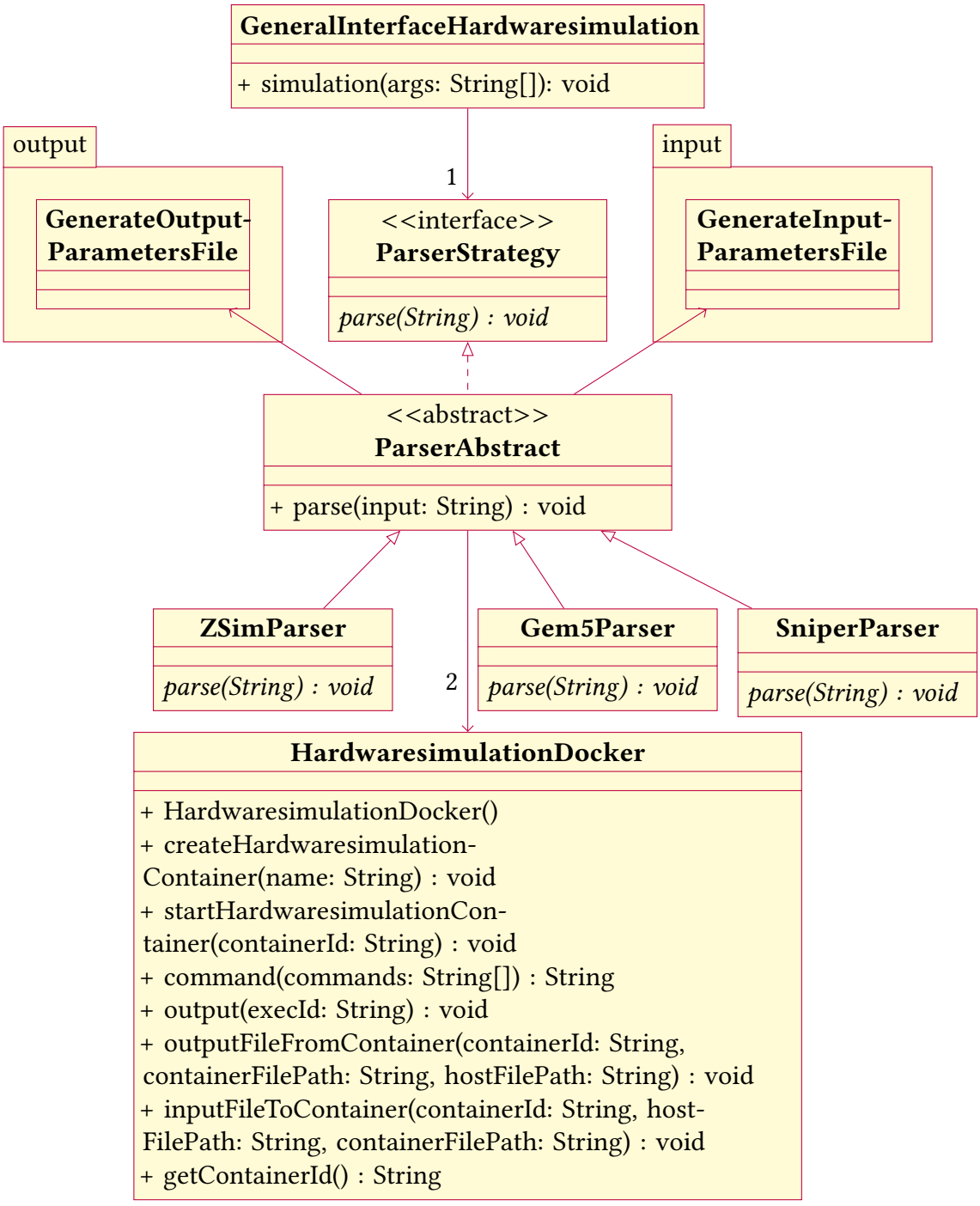

Abbildung 7.7: Struktur des gesamten Softwareentwurfs für die allgemeine Schnittstelle

Der Softwareentwurf für die allgemeine Schnittstelle besteht aus verschiedenen miteinander interagierenden Klassen, wie in Abbildung 7.7 dargestellt. Die zentrale Klasse ist

`GeneralInterfaceHardwaresimulation`, die die Methode `simulation(args: String[]): void` enthält. Die Methode `simulation` spielt eine entscheidende Rolle, da sie anhand des als Argument übergebenen Hardwaresimulationsnamens die entsprechende `ParserStrategy` auswählt, welche die Methode `parse` definiert. Die abstrakte Klasse `ParserAbstract` implementiert diese Schnittstelle und enthält die Methode `parse`. Es existieren drei konkrete Klassen, die von `ParserAbstract` abgeleitet sind: `SniperParser`, `Gem5Parser` und `ZSimParser`. Jede dieser Klassen implementiert die Methode `parse` auf ihre spezifische Art und Weise, da jede Hardwaresimulation spezifische Befehle ausführt und eine spezifische Konfigurationsdatei mit einem eigenen Format hat.
Die Klasse `HardwaresimulationDocker` stellt Methoden bereit, um die Hardwaresimulation in einem Docker-Container auszuführen. Dazu gehören Methoden wie `create-` und `start-HardwaresimulationContainer` und andere, die die Interaktion mit dem Container ermöglichen.
Die Pakete `input` und `output`, die in Abbildung 7.2 bzw. Abbildung 7.4 vorgestellt wurden, enthalten jeweils eine Klasse `GenerateInputParametersFile` und `GenerateOutputParametersFile`, die für das Erzeugen der Eingabe- und Ausgabeparameter-Dateien verantwortlich sind und mit den konkreten Parser-Klassen verbunden sind.
Die allgemeine Schnittstelle hat zwei Hauptaufgaben. Erstens dient sie dazu, die Eingabeparameter in das JSON-Format zu übersetzen und in eine Eingabekonfigurationsdatei für die ausgewählte Hardwaresimulation mithilfe des Input-Pakets zu überführen. Zweitens ermöglicht sie die Abbildung der Ergebnisse der Hardwaresimulation mithilfe des Output-Pakets in das JSON-Format, nachdem die Hardwaresimulation abgeschlossen ist. Zusammengefasst besteht die Hauptfunktion der allgemeinen Schnittstelle darin, die Kommunikation zwischen der Co-Simulation und der Hardwaresimulation zu erleichtern, indem sie die Eingabeparameter in das benötigte Format umwandelt und die Ergebnisse der Hardwaresimulation in ein strukturiertes JSON-Format überführt. Dies erleichtert den Datenaustausch und die Weiterverarbeitung der Ergebnisse. Die Verbindungen zwischen den Klassen zeigen die Abhängigkeiten und Interaktionen zwischen ihnen. Die Pfeile zeigen, welche Klassen welche Methoden aufrufen und wie die Daten zwischen den Klassen ausgetauscht werden. Insgesamt bietet dieser Softwareentwurf eine robuste Struktur für die allgemeine Schnittstelle der Hardwaresimulationen und ermöglicht eine einfache Erweiterung um weitere Hardwaresimulationen.

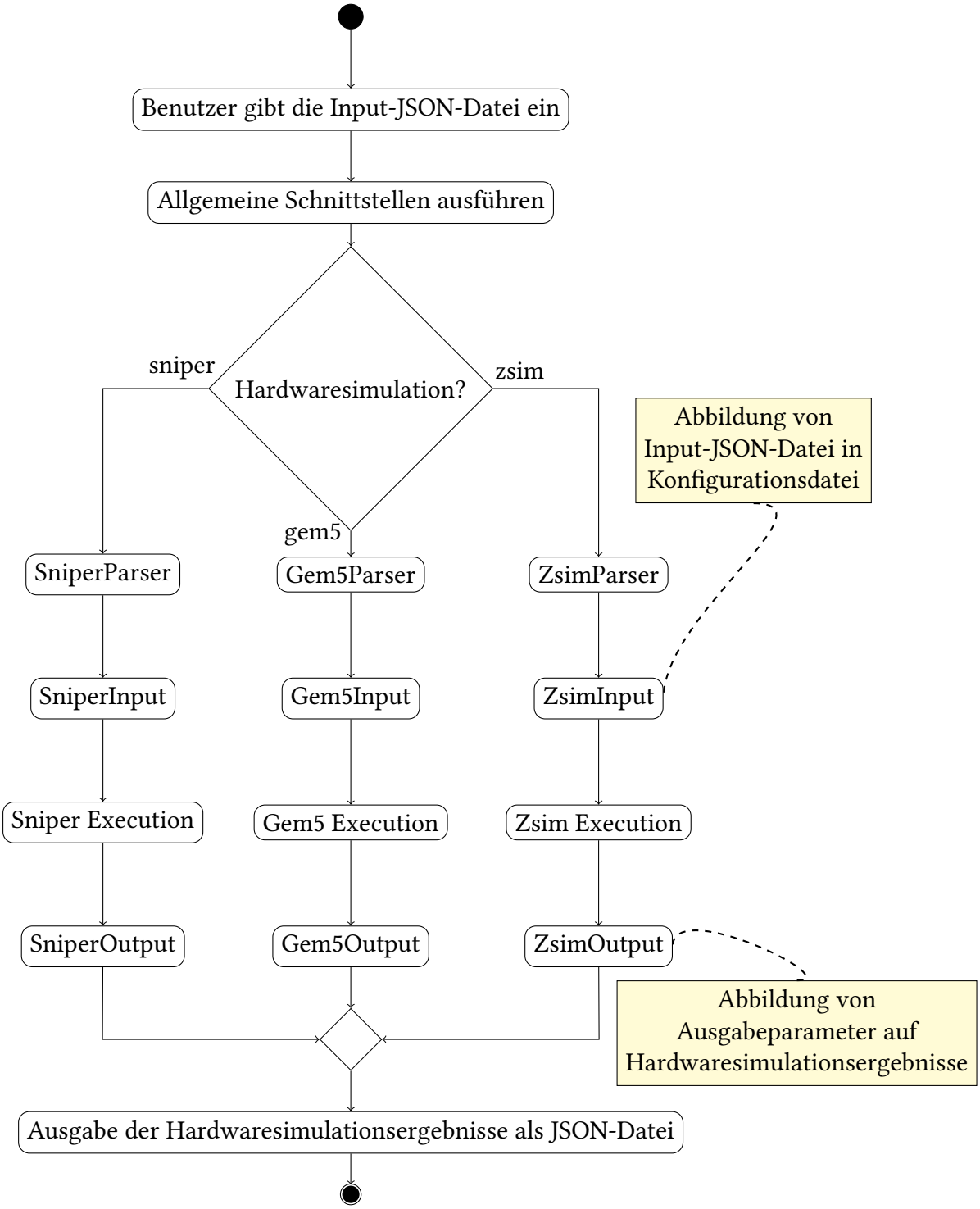

Abbildung 7.8: Prozessablauf der allgemeinen Schnittstelle mit Aktivitätsdiagramm

Die Abbildung 7.8 veranschaulicht das Aktivitätsdiagramm, welches den Ablauf der allgemeinen Schnittstelle für eine Klasse von Hardwaresimulationen zeigt. Der Prozess beginnt mit der Eingabe einer JSON-Datei durch den Benutzer. Diese Eingabe wird an die Durchführung der allgemeinen Schnittstellen weitergegeben. Daraufhin wird eine Entscheidung getroffen,

ob eine Hardwaresimulation durchgeführt werden soll oder nicht. Bei Auswahl der Hardwaresimulation wird entweder der *Gem5Parser*, *SniperParser* oder *ZsimParser* aktiviert, um die Eingabe zu analysieren. Nach der Eingabeanalyse wird eine Datei für die Eingabeparameter erstellt. Anschließend wird, je nach gewählter Hardwaresimulation (gem5, Sniper oder ZSim), fortgefahren. Die Schritte wie das Parsen der Eingabe, die Ausführung der Hardwaresimulation und die Erstellung der Ausgabe erfolgen bei allen jedoch jeweils mit eigener Implementierung. Nach der Hardwaresimulation werden die Ausgabeparameter in einer separaten Datei gespeichert. Abschließend erfolgt die Erstellung der Hardwaresimulationsergebnisse als JSON-Datei, womit der Prozess abgeschlossen wird.

Das Diagramm zeigt die einzelnen Schritte des Prozesses in Aktivitätskästchen und Entscheidungspunkte in Rauten. Die Pfeile verbinden die Aktivitäten und zeigen die Reihenfolge der Ausführung. Insgesamt bietet das Aktivitätsdiagramm eine Darstellung des Prozessablaufs der allgemeinen Schnittstelle für die Hardwaresimulation.

Der Code der allgemeinen Schnittstelle ist auf *https://github.com/* unter https://github.com/Mohammad-Nour-Dahi/GeneralInterfaceHardwaresimulation/tree/cef9a082d4ddd37bda617d95c03cc0c66a66bd81 zu finden.

8 Evaluation

Ziel	#	Frage	Metriken
Klassifizierung	1	Ist die Klassifizierung auf alle vorhandenen Hardwaresimulationen gut anwendbar?	Das Verhältnis von Hardwaresimulationen, die klassifiziert werden können.
	2	Inwiefern ähneln sich Schnittstellen von Hardwaresimulationen innerhalb einer Klasse?	Anzahl der gemeinsamen Parameter zwischen den Hardwaresimulationen innerhalb einer Klasse.
	3	Wie unterscheiden Hardwaresimulationen sich in Bezug auf ihre Zielsetzung innerhalb einer Klasse?	Anzahl der angebotenen Zielsetzungen zwischen den Hardwaresimulationen innerhalb einer Klasse.
Allgemeine Schnittstelle	4	Wie gut kann die Abbildung der Funktionen der Schnittstellen der verschiedenen Hardwaresimulationen durch eine allgemeine Schnittstelle erreicht werden?	Prozentsatz zwischen den Funktionen der Schnittstellen der Hardwaresimulationen und den Funktionen einer allgemeinen Schnittstelle.
	5	Ist der Aufwand für den Austausch von Hardwaresimulationen innerhalb einer bestimmten Klasse durch die Erstellung einer allgemeinen Schnittstelle für diese Klasse reduziert worden?	Anzahl der Änderungen an der Co-Simulation und an der Hardwaresimulation vor und nach Erstellung einer allgemeinen Schnittstelle.
	6	Wie gut ist die Performance der allgemeinen Schnittstelle?	Die Zeit, die die allgemeine Schnittstelle für die Abbildung und Ausführung der Hardwaresimulationen benötigt.
	7	Bleibt die Korrektheit der Hardwaresimulationsergebnisse erhalten?	Die Gleichheit der Hardwaresimulationsergebnisse bleibt unverändert, unabhängig davon, ob die allgemeine Schnittstelle genutzt wurde oder nicht.

Tabelle 8.1: GQM-Plan

Die Tabelle 8.1 zeigt den gesamten GQM-Plan mit den verschiedenen Zielen, Fragen und Metriken, die verwendet werden, um die gestellten Ziele zur Klassifizierung und Bewertung der allgemeinen Schnittstelle zu evaluieren. Der GQM-Plan dient dazu, die Untersuchung und Evaluation dieser beiden Ziele zu strukturieren und zu steuern. Im Folgenden wird eine Evaluation der Ergebnisse des GQM-Plans vorgenommen, um die Klassifizierung von Hardwaresimulationen und die allgemeine Schnittstelle zu analysieren und zu bewerten.

8.1 Klassifizierung

Die Evaluierung der Klassifizierung von Hardwaresimulationen ist von entscheidender Bedeutung, um sicherzustellen, dass die Klassifizierung ihre Ziele erreicht und zuverlässige Ergebnisse liefert. In Tabelle 8.1 des GQM-Modells werden spezifische Fragen formuliert, die sich auf die Klassifizierung beziehen und darauf abzielen, ihre Wirksamkeit und Aussagekraft zu messen. Diese Fragen werden im Folgenden analysiert und bearbeitet.

8.1.1 Die Anwendbarkeit der Klassifizierung

Die Anwendbarkeit der Klassifizierung auf alle existierenden Hardwaresimulationen zu untersuchen. Aufgrund der großen Vielfalt an Hardwaresimulationen, die in unterschiedlichen Bereichen eingesetzt werden können, stellt ihre Klassifizierung eine Herausforderung dar. Es ist daher schwierig, einheitliche Kriterien für die Klassifizierung aller 48 Hardwaresimulationen zu finden, die anhand dieser Arbeit gefunden wurden, in Tabelle 3.1 dargestellt. Um dieser Herausforderung zu begegnen, wurden im nächsten Schritt Kriterien für Hardwaresimulationen definiert. Diese Kriterien umfassen beispielsweise die Simulation der Gesamtfunktionalität und des Verhaltens der Standard-Hardware, die Unterstützung von Multicore, Execution-Driven, Aktualität innerhalb der letzten 5 Jahre und die Verfügbarkeit als Host für die x86-64-Architektur. Dabei wurden diejenigen Hardwaresimulationen ausgeschlossen, die diese Kriterien nicht erfüllen, um sie bei der Klassifizierung nicht weiter zu berücksichtigen. Die Anwendung dieser Kriterien ist von zentraler Bedeutung, um Hardwaresimulationen zu filtern. Danach kann die Klassifizierung auf die gefilterten Hardwaresimulationen angewandt werden. In Tabelle 8.2 werden die ungeeigneten Eigenschaften der Hardwaresimulationen aufgeführt, die nicht den gewünschten Kriterien entsprechen. Die Tabelle gibt auch die Anzahl der Hardwaresimulationen an, die diese ungeeigneten Eigenschaften aufweisen. Außerdem gibt es in Tabelle 8.2 zwei Hardwaresimulationen, die jeweils zwei ungeeignete Eigenschaften aufweisen.

Die ungeeigneten Eigenschaften	#
Grafische Hardwaresimulationen	2
Speicher-Hardwaresimulation	1
Cache-Hardwaresimulationen	2
Trace-Driven	7
Keine Commits seit über 5 Jahren	17
Single-Core-Prozessoren	7
Notwendige Informationen nicht öffentlich verfügbar	1
Keine Unterstützung für x86-64-Architektur als Host	4

Tabelle 8.2: Ungeeignete Eigenschaften und dazugehörige Anzahl der Hardwaresimulationen

Die Tabelle 8.3 fasst die Ergebnisse einer Untersuchung von Hardwaresimulationen zusammen. Es werden insgesamt 48 Hardwaresimulationen betrachtet. Von diesen Hardwaresimulationen weisen 41 ungeeignete Eigenschaften auf. Es gibt auch 2 Hardwaresimulationen, die mehr als eine ungeeignete Eigenschaft haben. Nach Ausschluss der Hardwaresimulationen mit ungeeigneten Eigenschaften werden insgesamt 9 Hardwaresimulationen für die Klassifizierung

verwendet. Mithilfe der auf ihnen basierenden Klassifikation wurden die neun Hardwaresimulationen klassifiziert. Die Klassifizierungsergebnisse sind in Tabelle 4.2 verfügbar.

Gesamtzahl der untersuchten Hardwaresimulationen	48
Die Gesamtzahl der Hardwaresimulationen, die ungeeignete Eigenschaften haben	41
Anzahl der Hardwaresimulationen, die mehr als eine ungeeignete Eigenschaft haben	2
Anzahl der Hardwaresimulationen, die auf die Klassifizierung angewendet werden	48 - 41 + 2 = 9

Tabelle 8.3: Anzahl der Hardwaresimulationen, die auf die Klassifizierung angewendet werden

8.1.2 Schnittstellenähnlichkeiten bei Hardwaresimulationen einer Klasse

Im Kapitel 5 wurden die Eingabeparameter und Hardwaresimulationsergebnisse verglichen, um die Ähnlichkeiten und Unterschiede zwischen den Hardwaresimulationen zu untersuchen. Anhand dieser Vergleiche wurden drei Klassen identifiziert und in Tabelle 5.10 dargestellt. Für die zweite Klasse von Hardwaresimulationen wurde festgestellt, dass sieben Eingabeparameter gemeinsam zwischen den Hardwaresimulationen „zsim“, „gem5“ und „sniper“ vorhanden sind, wie in Tabelle 8.4 dargestellt. Zusätzlich wurden für jede spezifische Hardwaresimulation unterschiedliche Parameter gefunden: „zsim“ hat fünf spezifische Parameter, „gem5“ hat einen und „sniper“ hat neun. Darüber hinaus verfügt „gem5“ über 5 Standardwerte, während „Sniper“ 55 und „ZSim“ 38 besitzt. Diese Standardwerte wurden nicht in die allgemeine Schnittstelle aufgenommen, da jede Hardwaresimulation spezifische Parameter mit detaillierten Standardwerten hat, die möglicherweise in anderen Hardwaresimulationen nicht vorhanden sind. Die Variation dieser spezifischen Parameter erfordert ein fundiertes Wissen über die jeweilige Hardwaresimulation, weshalb sie in dieser Arbeit nicht weiter untersucht wurde. Der Schwerpunkt lag hauptsächlich auf den allgemeinen Parametern, die in verschiedenen Hardwaresimulationen vorhanden sind.

Hardwaresimulation	**Eingabeparameter**		
	gemeinsamen	**spezifisch**	**standard**
gem5	7	1	5
Sniper		9	55
ZSim		5	38

Tabelle 8.4: Anzahl der Eingabeparameter

Im Hinblick auf die Hardwaresimulationsergebnisse wurden insgesamt 17 Parameter gefunden, die gemeinsam zwischen den Hardwaresimulationen existieren. Des Weiteren wurden spezifische Hardwaresimulationsergebnisse, die in anderen Hardwaresimulationen nicht vorhanden sind, bei der Analyse außer Acht gelassen. Diese Unterscheidung wird durch Tabelle Tabelle 8.5 veranschaulicht, in der gem5 1730 solcher spezifischen Ergebnisse aufweist, während Sniper 24 und ZSim sogar 2346 spezifische Hardwaresimulationsergebnisse aufweist.

Hardwaresimulation	Hardwaresimulationsergebnisse	
	gemeinsamen	spezifisch
gem5	17	1730
Sniper		24
ZSim		2346

Tabelle 8.5: Anzahl der Hardwaresimulationsergebnisse

In Tabelle 8.6 sind insgesamt 24 gemeinsame Parameter vorhanden, die sowohl Eingabeparameter als auch Hardwaresimulationsergebnisse umfassen. Dies bedeutet, dass diese 24 Parameter in allen betrachteten Hardwaresimulationen vorhanden sind und gemeinsam genutzt werden. Hingegen gibt es in dieser Klasse insgesamt 15 Unterschiede in den Parametern zwischen den Hardwaresimulationen. Diese Unterschiede beziehen sich auf spezifische Eingabeparameter, die in den verschiedenen Hardwaresimulationen variieren und nicht in allen Hardwaresimulationen vorhanden sind.

Die Gesamtzahl der spezifische Eingabeparameter	1 + 9 + 5 = 15
Eingabeparameter und Hardwaresimulationsergebnisse	7 + 17 = 24

Tabelle 8.6: Anzahl der Paramter

Für die erste Klasse und dritte Klasse liegt keine Implementierung der allgemeinen Schnittstelle vor, wodurch genaue Vorhersagen darüber, wie die Parameter im allgemeinen Schneidwert mit allen Parametern der Schneidwerte der Hardwaresimulationen in diese Klassen korrelieren, nicht möglich sind. Dennoch ist es möglich, qualitative Vergleiche anzustellen.

In der ersten Klasse gehören laut Tabelle 5.10 der Cache L1i und L1d sowie L2 zu allen Hardwaresimulationen in dieser Klasse. Zusätzlich wird Anzahl der Kerne und die Frequenz berücksichtigt. Bezüglich der Ausgabeparameter werden IPC, Cache, Zyklen und die Anzahl der Instruktionen betrachtet. Es werden sowohl spezifische als auch standardmäßige Parameter angegeben.

In der dritten Klasse gehören gemäß Tabelle Tabelle 5.10 der Cache L1i und L1d sowie L2 zu allen Hardwaresimulationen in dieser Klasse. Zusätzlich wird Anzahl der Kerne und die Frequenz berücksichtigt. In Bezug auf die Ausgabeparameter werden Simulationszeit, Host-Zeit und die Anzahl der Instruktionen betrachtet. Es werden sowohl spezifische als auch standardmäßige Parameter angegeben.

8.1.3 Hardwaresimulationen variieren in Zielsetzung

jedoch unterscheiden sich die Hardwaresimulationen hinsichtlich ihrer Zielsetzung innerhalb derselben Klasse. In allen drei Klassen in Tabelle 5.10 wurden jeweils alle möglichen Eigenschaften angeboten. In der zweiten Klasse sind sowohl das Full System als auch die Timing-Funktion und der User Mode verfügbar. Genauer gesagt unterstützt Gem5 das Full System und die Timing-Funktion auf Zyklen-Ebene, was eine detaillierte Analyse ermöglicht. Sniper ermöglicht hingegen den parallelen User Mode und die Timing-Funktion. ZSim++ bietet sowohl die parallele Timing-Funktion als auch den User Mode an. In dieser Klasse werden die Anforderungen an Genauigkeit und Geschwindigkeit je nach Anwendungsszenario sorgfältig

abgewogen.

8.2 Allgemeine Schnittstelle

Die Evaluation der allgemeinen Schnittstelle erfolgt durch gezielte Untersuchungen der Fragen in Tabelle 8.1 des GQM-Modells, die sich auf die Schnittstelle konzentrieren und darauf abzielen, ihre Entwicklung zu bewerten. Hierbei wird analysiert, inwieweit die Funktionen verschiedener Hardwaresimulationen mithilfe der allgemeinen Schnittstelle abgebildet werden können. Darüber hinaus wird geprüft, ob der Aufwand für den Austausch zwischen Hardwaresimulationen innerhalb einer Klasse durch die allgemeine Schnittstelle reduziert wird, wie effizient die Performance der allgemeinen Schnittstelle ist und inwiefern die Korrektheit der Ergebnisse der Hardwaresimulation sich darstellt.

8.2.1 Abbildung von Schnittstellen evaluieren

Das Ziel ist die Bestimmung der Fähigkeit einer allgemeinen Schnittstelle zur Abbildung der Funktionen von Schnittstellen verschiedener Hardwaresimulationen.

Die allgemeine Schnittstelle könnte sämtliche Funktionen der Hardwaresimulationen abdecken, wobei gemeinsame Parameter stärker berücksichtigt werden als spezifische Parameter für die einzelnen Hardwaresimulationen. Beispielsweise kann gem5 sowohl TIMING- als auch Full-System-Simulationen unterstützen, während die allgemeine Schnittstelle nur TIMING-Simulationen abbildet. Auch die Anzahl der Kerne wird möglicherweise nicht berücksichtigt, da die Ausgabe für Implementierungen von Sniper zeitaufwändig sein kann.

Die Tabelle 8.7 verdeutlicht, dass die allgemeine Schnittstelle nicht alle möglichen Konfigurationen der Hardwaresimulation abdeckt. Dabei werden mögliche Werte für einen Parameter aufgeführt. Die fett gedruckten Werte repräsentieren, welche Parameter von der allgemeinen Schnittstelle abgebildet werden können, während die nicht fett gedruckten Werte von der allgemeinen Schnittstelle nicht unterstützt werden.

Hardwaresimulation	**Parameter**	**Werte**
gem5	core type	ATOMIC, **TIM**, KVM, MINOR, O3
	num cores	1, **2**, 3, …
Sniper	core type	oneipc, **interval**, rob
	num cores	**1**, 2, 3, …
ZSim	core type	simpleCore, **skylake**, beefy, wimpy, nehalem
	num cores	**1**, 2, 3, …

Tabelle 8.7: Mehr mögliche Konfigurationen für einen Parameter, die von der allgemeinen Schnittstelle nicht unterstützt werden

Die allgemeine Schnittstelle kann nicht sämtliche Funktionen der Hardwaresimulationen innerhalb der Klasse vollständig abdecken. Dies ergibt sich aus den Unterschieden in Tabelle 8.7

anhand der Prozessortypen. Es existiert eine umfangreiche Anzahl von Parametern, die sich nicht sinnvoll auf eine allgemeine Schnittstelle übertragen lassen. Zudem wurde in dieser Bachelorarbeit nicht analysiert, ob diese Parameter – selbst wenn ihre Werte unterschiedlich sind – möglicherweise ein gemeinsames Verhalten aufweisen könnten. Ein Beispiel hierfür findet sich bei den Prozessortypen von gem5, wo ein möglicher Wert als *TIM* angegeben ist, während in Sniper der Wert *interval* genutzt wird. Die Ähnlichkeit oder Identität dieser Werte wurde nicht näher untersucht. Ungeachtet dessen bleibt die Frage offen, ob es bei ZSim Werte gibt, bei denen das Verhalten von *TIM* oder *interval* übereinstimmt. Aufgrund der Rahmenbedingungen der Arbeit war es nicht realisierbar, diese Frage zu untersuchen.

8.2.2 Austausch-Aufwand durch allgemeine Schnittstelle verringert

Vor der Erstellung einer allgemeinen Schnittstelle für den Austausch von Hardwaresimulationen (z. B. beim Wechsel von *gem5* zu *ZSim*) sind mehrere Änderungen erforderlich. Dazu gehören Anpassungen an den Namen der Eingabeparameter und Hardwaresimulationsergebnisse, da jede Hardwaresimulation ihre eigenen spezifischen Parameter, Ausgabeformate und Einheiten haben kann. Außerdem hat *gem5* eine Konfigurationsdatei als Python-Skript, wohingegen *ZSim* eine Konfigurationsdatei im Format cfg besitzt. Dies bedeutet, dass bei einem Wechsel von *gem5* zu *ZSim* Anpassungen an der Konfigurationsdatei vorgenommen werden müssen, um das richtige Format zu verwenden. Zusätzlich verfügt jede Hardwaresimulation über ihre eigene Methode zur Ausführung von Befehlen, was weitere Anpassungen erfordert, wenn man von der *gem5*-Konfigurationsdatei zu einer *ZSim*-Konfigurationsdatei übergeht. Es sind sowohl Änderungen an gemeinsamen Eingabeparametern als auch an spezifischen Eingabeparametern sowie an Standardwert-Eingabeparametern erforderlich. Die Tabelle 8.8 zeigt die Anzahl der Parameteränderungen zwischen den beiden Hardwaresimulationen. Es gibt insgesamt 50 Eingabeparameter, die sich aus 7 gemeinsamen Parametern, 5 spezifischen Parametern und 38 Standardwert-Eingabeparametern zusammensetzen, wie in Tabelle 8.4 dargestellt. Zusätzlich gibt es 17 Hardwaresimulationsergebnisse wie in Tabelle 8.5 und einen Ausführungsbefehl. Die Gesamtsumme der Änderungen beträgt 68, um zwischen den Hardwaresimulationen zu wechseln.

Die Gesamtzahl der Eingabeparameter	7 + 5 + 38 = 50
Hardwaresimulationsergebnisse	17
Anzahl der Befehle für die Ausführung	1
Summe der Änderungen	50 + 17 + 1 = 68

Tabelle 8.8: Anzahl der Parameteränderung von gem5 zu ZSim

Nach der Erstellung einer allgemeinen Schnittstelle wird der Aufwand für den Austausch erheblich reduziert. Es ist lediglich erforderlich, den Namen der aktuellen Hardwaresimulation durch den Namen der gewünschten Hardwaresimulation zu ersetzen. Durch diesen einfachen Schritt wird die Hardwaresimulation ausgetauscht, und dieser Prozess kann in nur einem Schritt abgeschlossen werden. Die Tabelle 8.9 vergleicht also die Anzahl der Änderungen, die ohne die allgemeine Schnittstelle gemacht wurden, mit der Anzahl der Änderungen, die mit der allgemeinen Schnittstelle gemacht wurden, um zwischen den Hardwaresimulationen zu wechseln.

Die Gesamtsumme der Änderungen ohne die allgemeine Schnittstelle	68
Die Gesamtsumme der Änderungen mit der allgemeinen Schnittstelle	1

Tabelle 8.9: Gesamtsumme der Änderungen beim Wechsel von gem5 zu ZSim

Die Gesamtzahl der Eingabeparameter	7 + 1 + 5 = 13
Hardwaresimulationsergebnisse	17
Anzahl der Befehle für die Ausführung	1
Summe der Änderungen	13 + 17 + 1 = 31

Tabelle 8.10: Anzahl der Parameteränderung von ZSim zu gem5

Wenn jedoch ein Wechsel von *ZSim* zu *gem5* ohne die allgemeine Schnittstelle vorgenommen werden soll, zeigt die Tabelle 8.10 die Anzahl der Parameteränderungen zwischen den Hardwaresimulationen *ZSim* und *gem5*. Die Gesamtzahl der Eingabeparameter beträgt 13, bestehend aus 7 gemeinsamen Eingabeparametern, einem spezifischen Eingabeparameter und 5 Standardwert-Eingabeparametern. Es werden 17 Hardwaresimulationsergebnisse und nur ein Ausführungsbefehl ermittelt. Die Summe der Änderungen von *ZSim* zu *gem5* beträgt 31. Daraus folgt, dass 31 Anpassungen erforderlich sind, wenn ein Wechsel von *ZSim* zu *gem5* vorgenommen werden soll. Die Tabelle 8.11 stellt die Gesamtzahl der Änderungen vor und nach der Erstellung einer allgemeinen Schnittstelle dar. Vor der Implementierung der Schnittstelle wurden 31 Änderungen durchgeführt, während danach nur eine zusätzliche Änderung erfolgte.

Die Gesamtsumme der Änderungen ohne die allgemeine Schnittstelle	31
Die Gesamtsumme der Änderungen mit der allgemeinen Schnittstelle	1

Tabelle 8.11: Gesamtsumme der Änderungen beim Wechsel von ZSim zu gem5

Wenn jedoch mehrere Austausche zwischen *gem5* und *ZSim* ohne allgemeine Schnittstelle stattfinden, werden auch die gleiche Anzahl an Änderungen bei jedem Wechsel vorgenommen. Wenn man zwischen drei Hardwaresimulationen wechseln will, ist der Aufwand höher. Im Gegensatz dazu benötigt die Verwendung einer allgemeinen Schnittstelle nur einen Schritt pro Austausch. Wenn man zwischen drei Hardwaresimulationen wechseln will, beträgt der Aufwand ebenfalls nur einen Schritt pro Austausch, da nur die Namen zwischen den drei Hardwaresimulationen, die in derselben Klasse liegen, geändert werden müssen. Dies erleichtert den Austausch von Hardwaresimulationen erheblich und reduziert den Aufwand, der zuvor für jede spezifische Simulation erforderlich war. Dadurch wird die Flexibilität erhöht und ermöglicht eine effiziente Nutzung verschiedener Hardwaresimulationen innerhalb derselben Klasse.

8.2.3 Performance der allgemeinen Schnittstelle

Die Zeit, die die allgemeine Schnittstelle für die Abbildung und Ausführung der Hardwaresimulationen benötigt, in der Tabelle 8.12 aufgeführt ist. Die Tabelle zeigt die Zeiten für die Abbildung und Ausführung von drei Hardwaresimulationen wie gem5, Sniper und ZSim sowie

die Zeit, die für die allgemeine Schnittstelle benötigt wird. Die Ausführungszeit, Hostzeit und die Zeit für die allgemeine Schnittstelle werden in Sekunden angegeben, wobei die allgemeine Schnittstelle im Durchschnitt 2,7 Sekunden benötigt. Die Tabelle gibt einen Überblick über die Performance der Hardwaresimulationen und den zusätzlichen Aufwand durch die allgemeine Schnittstelle.

Hardwaresimulation	**Ausführungszeit**	**Hostzeit**	**Zeit für die allgemeine Schnittstelle**
gem5	3,32	0,42	2,9
Sniper	5,56	4,18	1,38
ZSim	5,23	1,41	3,82
Durchschnitt benötigt die allgemeine Schnittstelle			2,7

Tabelle 8.12: Zeit für Abbildung und Ausführung der allgemeinen Schnittstelle

8.2.4 Korrektheit der Hardwaresimulationsergebnisse

Die Korrektheit der Hardwaresimulationsergebnisse bleibt unverändert, unabhängig von der Verwendung der allgemeinen Schnittstelle. In der Abbildung 8.1 werden die Ergebnisse der Sniper-Hardwaresimulation mit und ohne die allgemeine Schnittstelle verglichen. Die Ergebnisse sind bis auf die Reihenfolge identisch. Zusätzlich wird die HostNanoseconds-Zeit für Sniper mithilfe der allgemeinen Schnittstelle berechnet. Die konsistenten Ergebnisse der Hardwaresimulationen, unabhängig von der Verwendung der allgemeinen Schnittstelle, bestätigen die korrekte Implementierung und Zuverlässigkeit der Schnittstelle.

Der Fokus der Arbeit lag auf der Klassifizierung und der allgemeinen Schnittstelle. Eine Evaluierung der Genauigkeit der Simulationsergebnisse einzelner Hardwaresimulationen war nicht Teil der Arbeit, was eine gezielte Parameterauswahl für die Simulation erforderte. Die Abschätzung der Genauigkeit und Geschwindigkeit erfolgte in Abhängigkeit von der Granularität der Hardwaresimulation.

Listing 8.1: Output.json

```
"Cache Summary":{
 "Cache L1-D":{
    "miss rate": "3.79%",
    "mpki" : 13.85,
    "num cache accesses": 56818,
    "num cache misses": 2153
            },
 "Cache L1-I":{
    "miss rate" : "7.22%",
    "mpki" : 8.58,
    "num cache accesses" : 18479,
    "num cache misses" : 1334
    },
 "Cache L2":{
    "miss rate" : "89.13%",
    "mpki" : 20.24,
    "num cache accesses" : 3532,
    "num cache misses" : 3148,
    },
}
 "Cycles" : 505200,
 "HostNanoseconds" : 6521334600,
 "IPC" : 0.31,
 "Instructions" : 155507,
 "Time (ns)" : 168400
```

Listing 8.2: SniperOut

```
 Instructions                |      155507
  Cycles                     |      505200
  IPC                        |        0.31
  Time (ns)                  |      168400
Cache Summary                |
  Cache L1-I                 |
    num cache accesses       |       18479
    num cache misses         |        1334
    miss rate                |       7.22%
    mpki                     |        8.58
  Cache L1-D                 |
    num cache accesses       |       56818
    num cache misses         |        2153
    miss rate                |       3.79%
    mpki                     |       13.85
  Cache L2                   |
    num cache accesses       |        3532
    num cache misses         |        3148
    miss rate                |      89.13%
    mpki                     |       20.24
```

Abbildung 8.1: Vergleich der Sniper-Hardwaresimulation mit und ohne die allgemeine Schnittstelle

9 Verwandte Arbeiten

Im folgenden Kapitel werden verwandte Arbeiten präsentiert. Das Prinzip einer allgemeinen Schnittstelle wurde in der vorhandenen Literatur nicht angemessen für irgendeine Gruppe von Hardwaresimulationen behandelt. Gemäß meiner Recherche existiert einschlägige Literatur zur Klassifizierung von Hardwaresimulationen, auf der meine Arbeit basiert, insbesondere die von Akram und Sawalha. Zudem befasst sich auch Graef mit einer vergleichbaren Klassifizierung. Darüber hinaus stehen verschiedene Literaturquellen zur Verfügung, die unterschiedliche Hardwaresimulationen darstellen und diese miteinander vergleichen.

9.1 Klassifizierung von Hardwaresimulation

Im folgenden Abschnitt wird die Klassifizierung von Hardwaresimulationen vorgestellt. Dabei werden die Klassifizierungsansätze von Akram und Sawalha und Graef präsentiert. Es wird erläutert, welche Klassifizierung in meiner Arbeit verwendet wird und welche Unterschiede zu den Ansätzen von Akram und Greaf bestehen.

9.1.1 Klassifizierung von Akram und Sawalha

Der Artikel von Akram und Sawalha [7] bietet eine Übersicht über verschiedene Hardwaresimulationstechniken und untersucht mehrere Hardwaresimulationen, die in unterschiedliche Kategorien eingeteilt werden. Die Klassifizierung von Akram und Sawalha basiert auf drei wichtigen Faktoren: dem Detaillierungsgrad der Hardwaresimulationen, dem Zielbereich und der Eingabe für die Hardwaresimulationen. Im Folgenden werden diese Kategorien vorgestellt und näher erläutert.

Detaillierungsgrad der Hardwaresimulationen: Bezüglich des Detaillierungsgrads der Hardwaresimulationen werden Functional-, Timing- und Functional- /Timing- Hardwaresimulationen unterschieden.

- Functional (FUNC) Hardwaresimulationen: Eine funktionale Hardwaresimulation konzentriert sich ausschließlich auf die Implementierung der Architektur und zielt darauf ab, dieselbe Funktionalität wie die modellierte Architektur zu erreichen.

- Timing (TIM) Hardwaresimulationen werden die Mikroarchitektur von Prozessoren nachgebildet und liefern umfassende Statistiken zur Timing-Performance eines Systems. Sie ermöglichen die Analyse der Ausführungsgeschwindigkeit von Programmen und die Identifizierung von Optimierungsmöglichkeiten. Es gibt drei Typen davon: Zyklusebene-Hardwaresimulationen, Event-driven-Hardwaresimulationen und Interval-Hardwaresimulationen.

  - Zyklusebene-Hardwaresimulationen: imitieren den Betrieb eines Prozessors für jeden Zyklus, während zyklusgenaue Hardwaresimulationen die genaue Umsetzung jedes Zyklus mit Register-Transfer Level (RTL)-Implementierung nachahmen. Im Gegensatz zu den zyklusgenauen Hardwaresimulationen berücksichtigen Zyklusebene-Hardwaresimulationen nicht alle feinen Details der Hardware. Jedoch sind sie langsamer und benötigen mehr Speicher im Vergleich zu anderen Timing-Hardwaresimulationen. Laut Akram und Sawalha kann der funktionale Simulator „sim-fast“ für SimpleScalar Instruktionen 25-mal schneller simulieren als das detaillierte Zyklusebene-Simulationsmodell von SimpleScalar.
  - Event-Driven (EvDr) Hardwaresimulationen: simulieren Ziele aufgrund von Ereignissen anstelle von Zyklen und nutzen dabei Ereigniswarteschlangen, was Zeit einspart. Eine beispielhafte Hardwaresimulation wie SESC unterstützt die MIPS ISA und ermöglicht verschiedene Simulationmodelle.
  - Interval-Hardwaresimulationen: nutzen Fehlereignisse wie Cache-Misses und Verzweigungsmissschätzungen, um den Befehlsfluss in Intervalle zu unterteilen. Spezielle Teile des Simulators simulieren diese Fehlereignisse und ermitteln ihre genauen Zeitpunkte, um die Dauer jedes Befehlsintervalls abzuschätzen. Ein Beispiel hierfür ist Sniper, eine Hardwaresimulation, die in Multi-Core- und Many-Core-Systemen das Gleichgewicht zwischen Genauigkeit und Geschwindigkeit sucht.

- Integrierte Timing- und Funktional-Hardwaresimulationen ermöglichen ein flexibleres und genaueres Simulationsmodell. Sie können unabhängig voneinander arbeiten oder miteinander gekoppelt sein. Es gibt verschiedene Ansätze:
  - Timing-directed Hardwaresimulationen: nutzen die architektonischen Werte eines funktionalen Simulators für Timing-basierte Aufgaben, wenn benötigt.
  - Functional-first Hardwaresimulationen: führen funktionalen Simulator vor Timing-Simulator aus, generieren Befehlsstrom
  - Timing-first Hardwaresimulationen: laufen Timing-Hardwaresimulationen auf Zyklen-Ebene voraus, noch vor funktionalen Hardwaresimulationen.

Zielbereich

- Full System (FSys) Hardwaresimulation: ermöglicht das Ausführen von Betriebssystem-Binärdateien mit allen benötigten I/O-Geräten und Systemfunktionen in einer Hardwaresimulation.

- User Mode (UM) Hardwaresimulation: konzentrieren sich ausschließlich auf die Ausführung von Zielanwendungen und simulieren nur begrenzte Peripheriegeräte. Dabei werden in der Regel Systemaufrufe umgangen, und das Host-Betriebssystem wird genutzt.

Eingabe für die Hardwaresimulationen

- Trace-Driven (TD) Hardwaresimulationen: dienen als Eingabe für tracegesteuerte Simulatoren, um vorher aufgezeichnete Befehlsabläufe von Benchmarks mit festen Eingaben zu simulieren. Es ist jedoch zu beachten, dass diese Modelle keine Laufzeitänderungen von

mehrfädigen Anwendungen erfassen können, wodurch sie für parallele und zeitabhängige Systeme ungeeignet sind.

- Execution-Driven (EDr) Hardwaresimulationen: verwenden ausführbare Binärdateien und sind komplexer als trace-gesteuerte Hardwaresimulationen

Zusätzlich gibt es weitere Kategorien von Hardwaresimulationen wie Multiprozessor/Multicore-Hardwaresimulationen, spezialisierte/Beschleuniger-Hardwaresimulationen und modulare Hardwaresimulationen. Jede dieser Kategorien hat ihre eigenen spezifischen Anwendungsbereiche und ermöglicht detaillierte Untersuchungen und Analysen in verschiedenen Aspekten der Hardware- und Systemarchitektur. Darüber hinaus haben Akram und Sawalha einen detaillierten Vergleich der Hardwaresimulatoren, einschließlich gem5, MARSSx86, Multi2Sim, PTLsim, Sniper und ZSim, unter Berücksichtigung von Kriterien wie Genauigkeit, Leistung, Detailgrad und Entwicklungseinfachheit.

Die Klassifizierung von Hardwaresimulationen, wie von Akram und Sawalha vorgenommen, ist für diese Arbeit von großer Bedeutung, da sie sowohl bei der Auswahl der Kriterien zur Abgrenzung der Hardwaresimulationen als auch bei der eigentlichen Klassifizierung eine wichtige Rolle spielt. In dieser Arbeit wird die von Akram und Sawalha vorgeschlagene Klassifizierungsmethode angewendet und auf 9 Hardwaresimulationen erweitert. Ursprünglich beschränkte sich Akram und Sawalha auf die Klassifizierung nach Trace-Driven (TD) und Execution-Driven (EDr) basierend auf der Eingabe. In dieser Arbeit werden zusätzlich die Eingabe- und Ausgabeparameter berücksichtigt, um die Hardwaresimulationen mit verschiedenen Zielsetzungen zu klassifizieren. Diese Erweiterung ermöglicht eine genauere und umfassendere Klassifizierung der Hardwaresimulationen und trägt zur Entwicklung einer allgemeinen Schnittstelle bei.

9.1.2 Klassifizierung von Graef

In seiner Arbeit präsentiert Graef einen innovativen Ansatz, um einen Multicore-CPU-Simulator mit einer bereits existierenden Palladio-Komponente zu verbinden, mit dem Ziel, die Genauigkeit von Leistungsvorhersagen für Multicore-Systeme zu verbessern.

Palladio ist nach Reussner u. a. [63] ein bekanntes Werkzeug für diese Art von Vorhersagen, jedoch liefert es für parallele Anwendungen in Mehrkernsystemen ungenaue Ergebnisse, da es auf Single-Core-CPU-Architekturen aufbaut und nur die Taktrate als Metrik berücksichtigt. In dieser Arbeit werden Strategien vorgestellt, um diese Vorhersagemodelle für Mehrkernprozessoren zu erweitern, wie die Verwendung von Mehrkern-CPU-Simulatoren, Experimenten, multimetrischen Modellen und der Untersuchung von Software-Modellierungssprachen.

Außerdem hat Graef [28] einen Überblick über die Eigenschaften verschiedener Hardwaresimulationen bereitgestellt, darunter Sniper, Gem5, MaxSim, MARSSx86, ZSIM, Tejas und Multi2Sim. Er gibt die Unterstützung für Mehrkernsysteme, die Unterstützung für die x86-Befehlssatzarchitektur, die verwendete Programmiersprache, die Verwendung von Intel PIN, die Schwierigkeiten bei der Einrichtung des Simulators, die unterstützten Eingabeformate, die unterstützten Prozessormodelle, die Konfigurierbarkeit und die Unterstützung durch die Community für jeden Hardwaresimulationen an. Graef hat die Hardwaresimulationen anhand ihrer Fähigkeit zur direkten Ausführung von Java-Code, zur Simulation der x86 ISA und zur Mehrkernunterstützung klassifiziert. Die Untersuchung ergab, dass nicht alle Simulatoren

diese Eigenschaften erfüllen. Nach der Anwendung der Klassifizierungskriterien auf die Hardwaresimulationen zeigt sich, dass nur die Tejas-Java Hardwaresimulation und die MaxSim Hardwaresimulation diese Anforderungen erfüllen.

Die Klassifizierungskriterien basieren auf den Anforderungen von Palladio. Palladio hat zwei spezifische Ansätze: den Trace Simulation-Ansatz (SimuCom) und den Prototype Simulation Approach (ProtoCom). Für den SimuCom-Ansatz ist es wichtig, dass der Simulator Trace-Dateien aus der PCM-Instanz verarbeiten und die Ressourcennachfrage genau berechnen kann, sowie die x86 ISA simulieren und Java unterstützen kann. Im ProtoCom-Ansatz muss der Simulator den Java-Prototyp aus ProtoCom ausführen und die Leistung simulieren, ohne zusätzliche Abhängigkeiten zu haben. Auch hier sollte die x86 ISA simuliert und die Java-Programmierung unterstützt werden.

In seiner Arbeit hat Graef zusätzlich in Anhang A Dockerfiles für alle 6 Hardwaresimulationen präsentiert, die er in seiner Forschung behandelt hat. Diese Dockerfiles bieten eine unkomplizierte und standardisierte Methode, um die entsprechenden Hardwaresimulationen zu erstellen und zu verwenden, was die Einrichtung und Reproduzierbarkeit der Hardwaresimulationen erleichtert.

Die Klassifizierung von Hardwaresimulationen, wie von Graef durchgeführt, ist stark auf die Anforderungen von Palladio beschränkt, wodurch nur zwei spezielle Hardwaresimulationen diese Anforderungen erfüllen. Es ist auch festzustellen, dass diese beiden Hardwaresimulationen nicht alle Eigenschaften gemeinsam haben. Tejas-Java ist ein trace-basierte Hardwaresimulation, während MaxSim ein execution-basierte Hardwaresimulation ist. Obwohl Graef auch die Eingabeparameter berücksichtigt hat, gehören beide Hardwaresimulationen aufgrund der unterschiedlichen Eingabeformate (trace-basiert TD und execution-basiert EDr) zu verschiedenen Klassen sowohl in meiner Klassifizierung als auch in der Klassifizierung von Akram und Sawalha.

9.2 Vergleich von Hardwaresimulationen

In vielen wissenschaftlichen Arbeiten werden verschiedene Hardwaresimulationen verglichen. Die meisten dieser Arbeiten verwenden sowohl die Hardwaresimulationen als auch die Klassifizierungskriterien, die Akram und Sawalha in ihrem Papier zusammengefasst und gesammelt haben.

Die Klassifizierungskriterien von Urdén, Sanchez und Kozyrakis und Nikolic u. a. ermöglichen weder eine Klassifizierung noch die Erstellung einer allgemeinen Schnittstelle, da das Eingabeformat (Trace-Driven oder Execution-Driven) und die benötigten Eingabeparameter nicht berücksichtigt wurden.

9.2.1 Vergleich von Hardwaresimulationen gemäß Urdén

In der Untersuchung von Urdén [81] wurden fünf verschiedene Hardwaresimulationen ML-RSIM, SimICS, SimOS, SimpleScalar und VMWare hinsichtlich ihrer unterstützten Host-Plattformen, Host-Betriebssysteme, Target-Plattformen, Ziel-Betriebssysteme, Full System, Multiprozessor-Simulation und Out-Of-Order-Ausführung verglichen und klassifiziert. Die Klassifizierung erfolgt unabhängig vom Eingabeformat (Trace-Driven oder Execution-Driven)

und den benötigten Eingabeparametern, um die geeignetsten Optionen und Eigenschaften für verschiedene Anwendungsfälle zu ermitteln. Die Klassifizierungskriterien von Urdén ermöglichen weder eine Klassifizierung noch die Erstellung einer allgemeinen Schnittstelle, da das Eingabeformat und die benötigten Eingabeparameter nicht berücksichtigt wurden.

9.2.2 Vergleich von Hardwaresimulationen nach Sanchez und Kozyrakis

Sanchez und Kozyrakis [67] führten einen Vergleich verschiedener Hardwaresimulationen durch, darunter gem5/MARSS, CMPSim, Graphite, Sniper, HORNET, SlackSim und ZSim. In dieser Untersuchung wurden Aspekte wie Engine-Parallelisierung, detaillierte Kernsimulation, präzise Unicore-Simulation, Full System-Simulation, Multiprozess-Anwendungen berücksichtigt. Darüber hinaus wurden drei Techniken vorgestellt, die darauf abzielen, das Verhältnis zwischen Geschwindigkeit und Genauigkeit bei parallelen Hardwaresimulationen mit detaillierten architektonischen Modellen zu optimieren. Die Umsetzung dieser Ansätze führte zur Entwicklung der validierten zsim-Hardwaresimulation, der in Simulationen mit tausend Kernen Geschwindigkeiten von bis zu 1.500 MIPS erreicht. Die Kriterien zur Klassifizierung, die von Sanchez und Kozyrakis präsentiert wurden, erlauben weder eine angemessene Kategorisierung noch die Entwicklung einer umfassenden Schnittstelle, da sie das Eingabeformat und die erforderlichen Parameter für die Eingabe nicht einbezogen haben.

9.2.3 Vergleich von Hardwaresimulationen von Nikolic u. a.

Nikolic u. a. [57] führten eine Vergleichsstudie von unterschiedlichen Hardwaresimulationen wie ANT, CASLE, CCSTUDIO, CodeWarrior, CPU Sim, DigLC2, DLXview, Easy CPU, EDCOMP, ESCAPE, FastCache, HASE, HASE-Dinero, ISE Design Suite, JCachesim, JHDL, Logisim, M5, Quartus II, RM, RSIM, SIMCA, SimFlex, Simics, SimOS, SimpleScalar, SMOK und Virtual Vulcan durch. Hierbei wurden die Hardwaresimulationen anhand verschiedener Kriterien wie Verfasser, Zielgruppe, unterstützte Betriebssysteme, verwendete Programmiersprachen, Verfügbarkeit und Zielarchitektur der Computer, auf denen die Simulationen ausgeführt werden, verglichen. Die von Nikolic u. a. vorgestellten Kriterien sind nicht hinreichend, um eine Klassifizierung zu ermöglichen und eine allgemeine Schnittstelle zu entwickeln. Dies liegt daran, dass sie das Eingabeformat sowie die erforderlichen Eingabeparameter nicht berücksichtigt haben.

9.2.4 Vergleich von Hardwaresimulationen nach Alves u. a.

Alves u. a. haben in ihrer Arbeit [8] die wesentlichen Eigenschaften einer Vielzahl von bekannten Hardwaresimulationen wie SimAlpha, SimICS, SESC, GEMS, M5, Gem5, PTLsim, Multi2Sim, COTSon und MARSSx86 anhand von Kriterien wie FSys, Mikroarchitektur, Erweiterbarkeit, Netzwerk-on-Chip-Modell, Unterstützung von Nicht-einheitlichen Cache-Architekturen (NUCA) und Speichercontroller untersucht. Die präsentierten Kriterien von Alves u. a. genügen nicht, um eine Klassifizierung zu ermöglichen und eine allgemeine Schnittstelle zu gestalten, da sie das Eingabeformat sowie die benötigten Eingabeparameter nicht berücksichtigt haben.

9.2.5 Vergleich von Hardwaresimulationen von Schönwetter

Schönwetter [71] bietet eine umfassende Übersicht über Multi- und Many-Core-Hardwaresimulationen wie Graphite, Sniper, SoCLib, HORNET, QEMU und gem5 von Schönwetter. Diese Hardwaresimulationen sind in der Lage, kompilierten Applikationscode unabhängig von der Abstraktionsebene auszuführen und zu bewerten. Darüber hinaus vergleicht die Arbeit die Hardwaresimulationen hinsichtlich ihrer Simulationsperformanz, der Möglichkeit von Multi- und Many-Core-Simulationen, der Energieabschätzung und der Verfügbarkeit von eingebetteten Prozessormodellen. Die vorgestellten Klassifizierungskriterien durch Schönwetter sind nicht ausreichend für eine angemessene Einordnung und die Schaffung einer allgemeinen Schnittstelle, da sie das Eingabeformat und die notwendigen Parameter für die Eingabe nicht berücksichtigt haben.

10 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine allgemeine Schnittstelle entwickelt, um den Austausch von Hardwaresimulationen innerhalb einer Klasse zu ermöglichen, wobei diese Hardwaresimulationen unterschiedliche Zielsetzungen verfolgen. Einige Hardwaresimulationen konzentrieren sich auf Genauigkeit, während andere auf Geschwindigkeit setzen und dabei die Genauigkeit vernachlässigen. Um diese allgemeine Schnittstelle zu entwickeln, wurde ein Schnittstellenkonzept für die Co-Simulation von Hardwaresimulationen in zwei aufeinanderfolgenden Schritten entwickelt. Die erste Phase umfasst die Filterung, gefolgt von der zweiten Phase, der Klassifizierung.

Zu Beginn wurden Kriterien definiert, um Hardwaresimulationen zu unterscheiden und verschiedene Arten von Hardwaresimulationen zu filtern. Dadurch konnte eine gezielte Auswahl und Analyse relevanter Hardwaresimulationen durchgeführt werden.

Nach der Filterung wurden die verbleibenden Hardwaresimulationen anhand ihrer Eingabe- und Ausgabeparameter in verschiedene Klassen mit unterschiedlichen Zielsetzungen eingeteilt. Dabei wurde darauf geachtet, dass die Anzahl der gemeinsamen Parameter für alle Hardwaresimulationen in einer Klasse größer ist als die Gesamtanzahl der spezifischen Parameter, die nur für einzelne Hardwaresimulationen in dieser Klasse gelten.

Nachdem Hardwaresimulationen in Klassen eingeteilt wurden, kann eine allgemeine Schnittstelle für alle Hardwaresimulationen einer Klasse entwickelt werden. Für die Implementierung einer solchen allgemeinen Schnittstelle innerhalb einer Klasse fiel die Wahl auf die zweite Klasse, welche die Hardwaresimulationen gem5, Sniper und ZSim umfasst. Die allgemeine Schnittstelle wurde unter Berücksichtigung dieser Klassifizierung entwickelt, wobei besonderes Augenmerk auf die Eingabe- und Ausgabeparameter gelegt wurde, da sie eine grundlegende Rolle bei der Definition der Schnittstelle spielen.

Die entwickelte allgemeine Schnittstelle zeigt eine hohe Leistung bei der Austauschbarkeit von Hardwaresimulationen und ermöglicht gleichzeitig eine erhöhte Wartbarkeit. Dies führt zu einer signifikanten Steigerung der Flexibilität und Anpassungsfähigkeit der Co-Simulation. Durch die Verwendung dieser allgemeinen Schnittstelle wird der Aufwand für den Austausch erheblich reduziert. Es ist lediglich erforderlich, den Namen der aktuellen Hardwaresimulation durch den Namen der gewünschten Simulation zu ersetzen. Die Zeit, die von der allgemeinen Schnittstelle für die Abbildung und Ausführung der Hardwaresimulationen benötigt wird, beträgt im Durchschnitt 2,7 Sekunden. Die konsistenten Ergebnisse der Hardwaresimulationen, unabhängig von der Verwendung der allgemeinen Schnittstelle, bestätigen die korrekte Implementierung und Zuverlässigkeit dieser Schnittstelle. Dies unterstreicht die Genauigkeit und Verlässlichkeit ihrer Funktion.

Angesichts der Begrenzungen einer Bachelorarbeit war es nicht durchführbar, sämtliche denkbaren Hardwaresimulationen zu untersuchen. Dies schließt KI-, Grafik- und Single-Core-Simulationen sowie deren mögliche Klassifizierung mit ein. Des Weiteren wurde eine allgemeine Schnittstelle lediglich für Hardwaresimulationen der zweiten Kategorie entwickelt, nicht jedoch für solche der ersten und dritten Klassen.

11 Abkürzungsverzeichnis

ARM	Acorn RISC Machines
DBT	Dynamic Binary Translation
DRAM	Dynamic Random Access Memory
EDr	Execution-Driven
EvDr	Event-Driven
FCFS	First-Come, First-Served
FPGA	Field Programmable Gate Array
FSys	Full System
FUNC	Functional
GPGPU	General Purpose Graphics Processing Unit
HMP	Heterogeneous Multiprocessor
IO	In-Order
ISA	Instruction Set Architecture
LRU	Least Recently Used
MIPS	Microprocessor without Interlocked Pipeline Stages
MOD	Modular
NoC	Network-on-Chip
OOO	Out-Of-Order
OS	Operating System
Pin	Pin ist ein System zur dynamischen Instrumentierung von Binärdateien, das auf Linux und Windows ausgeführt werden kann und mit verschiedenen Intel-Prozessoren kompatibel ist.
QEMU	Quick Emulator
RTL	Register-Transfer Level
SMT	Simultaneous Multithreading
TD	Trace-Driven
TIM	Timing
UM	User Mode

Literatur

[1] Tor M Aamodt u. a. *GPGPU-Sim 3. x manual*. [Online; accessed 12-Juli-2023]. 2012. URL: http://gpgpu-sim.org/manual/index.php/Main_Page#Contributors_to_GPGPU-Sim_version_3.x.

[2] Daniel Aarno und Jakob Engblom. *Software and System Development Using Virtual Platforms Full-System Simulation with Wind River Simics. Full-System Simulation with Wind River Simics*. Elsevier Science Technology, 2014, S. 366. ISBN: 9780128007259. URL: https://dl.acm.org/doi/10.5555/2721547.

[3] Daniel Aarno und Jakob Engblom. *Software and System Development Using Virtual Platforms Full-System Simulation with Wind River Simics. Full-System Simulation with Wind River Simics*. [Online; accessed 23-Juli-2023]. 2014. URL: https://www.windriver.com/products/simics/knowledge-library.

[4] Jung Ho Ahn. *mcsim_public: McSimA+, modernized*. https://github.com/scale-snu/mcsim_public#mcsima-modernized. [Online; accessed 22-Juli-2023]. 2021.

[5] Jung Ho Ahn u. a. „McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling“. In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2013. DOI: 10.1109/ispass.2013.6557148.

[6] Jung Ho Ahn u. a. *McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling*. https://github.com/scale-snu/mcsim_public. [Online; accessed 27-Juli-2023]. Apr. 2013.

[7] Ayaz Akram und Lina Sawalha. „A Survey of Computer Architecture Simulation Techniques and Tools“. In: *IEEE Access* 7 (2019), S. 78120–78145. DOI: 10.1109/access.2019.2917698.

[8] Marco Antonio Zanata Alves u. a. „SiNUCA: A Validated Micro-Architecture Simulator“. In: *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, Aug. 2015. DOI: 10.1109/hpcc-css-icess.2015.166.

[9] E. K. Ardestani und J. Renau. „ESESC: A fast multicore simulator using Time-Based Sampling“. In: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2013. DOI: 10.1109/hpca.2013.6522340.

[10] Eduardo Argollo u. a. „COTSon“. In: *ACM SIGOPS Operating Systems Review* 43.1 (Jan. 2009), S. 52–61. DOI: 10.1145/1496909.1496921.

[11] T. Austin, E. Larson und D. Ernst. „SimpleScalar: an infrastructure for computer system modeling“. In: *Computer* 35.2 (2002), S. 59–67. DOI: 10.1109/2.982917.

[12] Jose Renau und Basilio Fraguela und James Tuck und Wei Liu und Milos Prvulovic und Luis Ceze und Smruti Sarangi und Paul Sack und Karin Strauss und Pablo Montesinos. *SESC simulator*. [Online; accessed 22-Juli-2023]. Jan. 2005. URL: http://sesc.sourceforge.net.

[13] Jose Renau und Basilio Fraguela und James Tuck und Wei Liu und Milos Prvulovic und Luis Ceze und Smruti Sarangi und Paul Sack und Karin Strauss und Pablo Montesinos. *SESC simulator*. https://github.com/kaien3/SESC. [Online; accessed 23-Juli-2023]. 2005.

[14] Jose Renau und Basilio Fraguela und James Tuck und Wei Liu und Milos Prvulovic und Luis Ceze und Smruti Sarangi und Paul Sack und Karin Strauss und Pablo Montesinos. *SESC simulator*. https://sesc.sourceforge.net/docs.html. [Online; accessed 23-Juli-2023]. 2005.

[15] Nathan Binkert u. a. *gem5*. https://github.com/gem5/gem5. [Online; accessed 27-Juli-2023]. 2011.

[16] Nathan Binkert u. a. „The gem5 simulator“. In: *ACM SIGARCH Computer Architecture News* 39.2 (Mai 2011), S. 1–7. DOI: 10.1145/2024716.2024718.

[17] Bobby R. Bruce, Jason Lowe-Power und mbabaie. *gem5-bootcamp-env*. https://github.com/gem5bootcamp/gem5-bootcamp-env. [Online; accessed 24-Juli-2023]. 2022.

[18] Trevor E. Carlson, Wim Heirman und Lieven Eeckhout. „Sniper“. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Nov. 2011. DOI: 10.1145/2063384.2063454.

[19] Trevor E. Carlson, Wim Heirman und Lieven Eeckhout. *Sniper*. https://github.com/Icarusradio/sniper. [Online; accessed 23-Juli-2023]. 2016.

[20] Bob Cmelik und David Keppel. „Shade: a fast instruction-set simulator for execution profiling“. In: *Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*. ACM, Mai 1994. DOI: 10.1145/183018.183032.

[21] *Comparison_of_instruction_set_architectures*. [Online; accessed 20-Mai-2023]. 2023. URL: https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures.

[22] Ashutosh Dhodapkar u. a. „TEM2P2EST: A Thermal Enabled Multi-model Power/Performance ESTimator“. In: *Power-Aware Computer Systems*. Springer Berlin Heidelberg, 2001, S. 112–125. DOI: 10.1007/3-540-44572-2_9.

[23] Jan Edler. „Dinero IV: Trace-driven uniprocessor cache simulator“. In: (1994). [Online; accessed 20-Juli-2023]. URL: https://pages.cs.wisc.edu/~markhill/DineroIV/.

[24] J. Emer u. a. „Asim: a performance model framework“. In: *Computer* 35.2 (2002), S. 68–76. DOI: 10.1109/2.982918.

[25] „Flexus“. In: (Aug. 2015). [Online; accessed 26-Juli-2023]. URL: https://parsa.epfl.ch/simflex/.

[26] Markus Frank. *Model-based performance prediction for concurrent software on multicore architectures - a simulation-based approach*. en. [Online; accessed 12-Juli-2023]. 2021. DOI: 10.18419/OPUS-11743.

[27] Cláudio Gomes u. a. „Co-Simulation: A Survey“. In: *ACM* (2018). URL: https://dl.acm.org/doi/abs/10.1145/3179993.

[28] Sebastian Graef. *Connecting Palladio with multicore CPU simulators*. en. [Online; accessed 11-Juli-2023]. 2018. DOI: 10.18419/OPUS-10188.

[29] Nikolaos Hardavellas u. a. „SimFlex“. In: *ACM SIGMETRICS Performance Evaluation Review* 31.4 (März 2004), S. 31–34. DOI: 10.1145/1054907.1054914.

[30] Sina Hassani, Gabriel Southern und Jose Renau. „LiveSim: Going live with microarchitecture simulation“. In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, März 2016. DOI: 10.1109/hpca.2016.7446098.

[31] Jian Huang. „The SImulator for Multi-threaded Computer Architecture (Release 1.2)“. In: *http://agassiz.cs.umn.edu/Tools/SIMCA/simca.html* (2000). URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=781ba597428cbeb3f4452f4079f50abebd48eb45.

[32] C.J. Hughes u. a. „Rsim: simulating shared-memory multiprocessors with ILP processors“. In: *Computer* 35.2 (2002), S. 40–49. DOI: 10.1109/2.982915.

[33] R. N. Ibbett. „HASE: A Flexible Toolset for Computer Architects“. In: *The Computer Journal* 38.10 (Okt. 1995), S. 755–764. DOI: 10.1093/comjnl/38.10.755.

[34] *Intel-Nehalem-Mikroarchitektur*. [Online; accessed 23-Juli-2023]. 2023. URL: https://de.wikipedia.org/wiki/Intel-Nehalem-Mikroarchitektur.

[35] *Intel-Skylake-Mikroarchitektur*. [Online; accessed 07-August-2023]. 2023. URL: https://de.wikipedia.org/wiki/Intel-Skylake-Mikroarchitektur.

[36] *Interval Simulation*. [Online; accessed 23-Juli-2023]. 2012. URL: http://snipersim.org/w/Interval_Simulation.

[37] Aamer Jaleel u. a. „CMP$im: A Pin-based on-the-fly multi-core cache simulator“. In: *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*. 2008, S. 28–36. URL: https://user.eng.umd.edu/~blj/papers/mobs2008.pdf.

[38] Kecheng Ji u. a. „An artificial neural network model of LRU-cache misses on out-of-order embedded processors“. In: *Microprocessors and Microsystems* 50 (Mai 2017), S. 66–79. DOI: 10.1016/j.micpro.2017.02.005.

[39] E. K. Ardestani und J. Renau. *ESESC: A Fast Multicore Simulator Using Time-Based Sampling*. https://github.com/masc-ucsc/esesc. [Online; accessed 23-Juli-2023]. 2013.

[40] Sagar Karandikar u. a. „FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud“. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Juni 2018. DOI: 10.1109/isca.2018.00014.

[41] Kenji Kise u. a. „The SimCore/Alpha Functional Simulator“. In: *Proceedings of the 2004 workshop on Computer architecture education held in conjunction with the 31st International Symposium on Computer Architecture - WCAE '04*. ACM Press, 2004. DOI: 10.1145/1275571.1275602.

[42] P.M. Kogge. „An Exploration of the Technology Space for Multi-Core Memory/Logic Chips for Highly Scalable Parallel Systems“. In: *Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'05)*. IEEE. DOI: 10.1109/iwia.2005.24.

[43] James Larus. „Spim: A mips32 simulator“. In: *Computer Science Department, University of Wisconsin–Madison*. (2005). [Online; accessed 10-Juli-2023]. URL: http://pages.cs.wisc.edu/~larus/spim.html.

[44] Sheng Li u. a. „McPAT“. In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, Dez. 2009. DOI: 10.1145/1669112.1669172.

[45] Imperas Software Limited. *OVP Guide to Using Processor Models*. https://github.com/riscv-ovpsim/imperas-riscv-tests. [Online; accessed 29-Juli-2023]. 2021.

[46] *Liste_von_Betriebssystemen*. [Online; accessed 20-Mai-2023]. 2023. URL: https://de.wikipedia.org/wiki/Liste_von_Betriebssystemen.

[47] Gabriel H. Loh, Samantika Subramaniam und Yuejian Xie. „Zesto: A cycle-level simulator for highly detailed microarchitecture exploration“. In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, Apr. 2009. DOI: 10.1109/ispass.2009.4919638.

[48] Jochen Ludewig. „Modelle im Software Engineering – eine Einführung und Kritik“. In: *Modellierung 2002, Modellierung in der Praxis – Modellierung für die Praxis*. Hrsg. von Martin Glinz und Günther Müller-Luschnat. Bonn: Gesellschaft für Informatik e.V., 2002, S. 7–22. URL: https://dl.gi.de/handle/20.500.12116/30650.

[49] Milo M. K. Martin u. a. „Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset“. In: *ACM SIGARCH Computer Architecture News* 33.4 (Nov. 2005), S. 92–99. DOI: 10.1145/1105734.1105747.

[50] Friedemann Mattern. „Modellbildung und Simulation.“ In: *Perspektiven der Informatik*. 1993, S. 56–64. URL: http://vs.inf.ethz.ch/publ/papers/Mod_Sim.pdf.

[51] I. McGregor. „The relationship between simulation and emulation“. In: *Proceedings of the Winter Simulation Conference*. IEEE. DOI: 10.1109/wsc.2002.1166451.

[52] Jason E Miller u. a. „Graphite: A distributed parallel simulator for multicores“. In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, Jan. 2010. DOI: 10.1109/hpca.2010.5416635.

[53] M. Moudgill. „Techniques for implementing fast processor simulators“. In: *Proceedings 31st Annual Simulation Symposium*. IEEE Comput. Soc. DOI: 10.1109/simsym.1998.668450.

[54] Aleksandar Milenkovic Mounika Ponugoti Amrish K. Tewar. *Multi2Sim Simulator: System Configuration*. http://lacasa.uah.edu/images/Upload/tutorials/m2s/Multi2Sim_SystemConfiguration.pdf. [Online; accessed 23-Juli-2023]. 2014.

[55] S.S. Mukherjee u. a. „Wisconsin Wind Tunnel II: a fast, portable parallel architecture simulator“. In: *IEEE Concurrency* 8.4 (Okt. 2000), S. 12–20. DOI: 10.1109/4434.895100.

[56] A.-T. Nguyen u. a. „The Augmint multiprocessor simulation toolkit for Intel x86 architectures“. In: *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*. IEEE Comput. Soc. Press. DOI: 10.1109/iccd.1996.563597.

[57] Bosko Nikolic u. a. „A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization“. In: *IEEE Transactions on Education* 52.4 (Nov. 2009), S. 449–458. DOI: 10.1109/te.2008.930097.

[58] Avadh Patel, Furat Afram und Kanad Ghose. „Marss-x86: A qemu-based microarchitectural and systems simulator for x86 multicore processors“. In: *1st International Qemu Users' Forum*. Citeseer. 2011, S. 29–30. DOI: 10.1109/ICPADS.2011.102. URL: https://www.researchgate.net/profile/Wei-Chung-Hsu-3/publication/241626652_PQEMU_A_parallel_system_emulator_based_on_QEMU/links/00b495324de9fec616000000/PQEMU-A-parallel-system-emulator-based-on-QEMU.pdf#page=33.

[59] Michael Pellauer u. a. „HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing“. In: *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, Feb. 2011. DOI: 10.1109/hpca.2011.5749747.

[60] D.G. Perez, G. Mouchard und O. Temam. „MicroLib: A Case for the Quantitative Comparison of Micro-Architecture Mechanisms“. In: *37th International Symposium on Microarchitecture (MICRO-37'04)*. IEEE. DOI: 10.1109/micro.2004.25.

[61] Pengju Ren u. a. „HORNET: A Cycle-Level Multicore Simulator“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.6 (Juni 2012), S. 890–903. DOI: 10.1109/tcad.2012.2184760.

[62] Jose Renau. *ESESC*. https://github.com/masc-ucsc/esesc. [Online; accessed 25-Juli-2023]. 2021.

[63] Ralf H. Reussner u. a. *Modeling and Simulating Software Architectures. The Palladio Approach*. MIT Press, 2016, S. 400. ISBN: 9780262034760. URL: https://dl.acm.org/doi/book/10.5555/3036121.

[64] Andrey Rodchenko u. a. „MaxSim: A simulation platform for managed applications“. In: *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2017. DOI: 10.1109/ispass.2017.7975286.

[65] Felipe Rosa u. a. „Instruction-driven timing CPU model for efficient embedded software development using OVP“. In: *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, Dez. 2013. DOI: 10.1109/icecs.2013.6815549.

[66] M. Rosenblum u. a. „Complete computer system simulation: the SimOS approach“. In: *IEEE Parallel &amp Distributed Technology: Systems &amp Applications* 3.4 (1995), S. 34–43. DOI: 10.1109/88.473612.

[67] Daniel Sanchez und Christos Kozyrakis. „ZSim“. In: *ACM SIGARCH Computer Architecture News* 41.3 (Juni 2013), S. 475–486. DOI: 10.1145/2508148.2485963.

[68] Daniel Sanchez und Christos Kozyrakis. *ZSim++*. https://github.com/dzhang50/zsim-plusplus. [Online; accessed 23-Juli-2023]. 2015.

[69] Smruti R. Sarangi u. a. „Tejas: A java based versatile micro-architectural simulator“. In: *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, Sep. 2015. DOI: 10.1109/patmos.2015.7347586.

[70] Lambert Schaelicke und Mike Parker. „The design and utility of the ML-RSIM system simulator“. In: *Journal of Systems Architecture* 52.5 (Mai 2006), S. 283–297. DOI: 10.1016/j.sysarc.2005.07.001.

[71] Dominik Schönwetter. „Performante Simulation von echtzeitfähigen Applikationen und Präzisierung der Laufzeitvorhersage auf Basis instruktionsgenauer Simulationstechnik“. Diss. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2018. URL: https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/9758.

[72] Joseph Sharkey, Dmitry Ponomarev und Kanad Ghose. „M-sim: a flexible, multithreaded architectural simulation environment“. In: *Technical report, Department of Computer Science, State University of New York at Binghamton* (2005). [Online; accessed 10-Juli-2023]. URL: https://www.researchgate.net/publication/228912540_Abstract_M-SIM_A_Flexible_Multithreaded_Architectural_Simulation_Environment.

[73] Herbert Stachowiak. *Allgemeine modelltheorie*. Springer, 1973. URL: https://archive.org/details/Stachowiak1973AllgemeineModelltheorie/page/n61/mode/2up.

[74] Mark Sutherland. *Parsa-EPFL/flexus*. https://github.com/parsa-epfl/flexus. [Online; accessed 23-Juli-2023]. 2015.

[75] Dean M. Tullsen, Susan J. Eggers und Henry M. Levy. „Simultaneous multithreading“. In: *25 years of the international symposia on Computer architecture (selected papers)*. ACM, Aug. 1998. DOI: 10.1145/285930.286011.

[76] Rafael Ubal u. a. „Multi2Sim“. In: *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, Sep. 2012. DOI: 10.1145/2370816.2370865.

[77] Rafael Ubal u. a. „Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors“. In: *19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*. IEEE, Okt. 2007. DOI: 10.1109/sbac-pad.2007.17.

[78] Rafael Ubal u. a. *Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors*. https://github.com/Multi2Sim/multi2sim. [Online; accessed 23-Juli-2023]. 2007.

[79] Rafael Ubal u. a. *The Multi2Sim Simulation Framework A CPU-GPU Model for Heterogeneous Computing*. http://www.multi2sim.org/. [Online; accessed 23-Juli-2023]. 2013.

[80] *Understanding Cache System Simulation in zSim*. [Online; accessed 27-Juli-2023]. 2019. URL: https://wangziqi2013.github.io/article/2019/12/25/understand-zsim-ccsim.html.

[81] Ulf Urdén. *A Comparison of Three Computer System Simulators*. [Online; accessed 13-Juli-2023]. 2004. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A829764&dswid=-3410.

[82] M. Vachharajani u. a. „Microarchitectural exploration with Liberty“. In: *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.* IEEE Comput. Soc. DOI: 10.1109/micro.2002.1176256.

[83] J.E. Veenstra und R.J. Fowler. „MINT: a front end for efficient simulation of shared-memory multiprocessors“. In: *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE. DOI: 10.1109/mascot.1994.284422.

[84] David Wang u. a. „DRAMsim“. In: *ACM SIGARCH Computer Architecture News* 33.4 (Nov. 2005), S. 100–107. DOI: 10.1145/1105734.1105748.

[85] Kenzo Van Craeynest Wim Heirman Trevor E. Carlson. *Sniper*. http://snipersim.org/w/Tutorial:IISWC_2013_Sniper. [Online; accessed 23-Juli-2023]. 2016.

[86] Matt T. Yourst. „PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator“. In: *2007 IEEE International Symposium on Performance Analysis of Systems &amp Software*. IEEE, Apr. 2007. DOI: 10.1109/ispass.2007.363733.