Ein organisches Taskverarbeitungssystem für zuverlässige Multi-Core SoC-Architekturen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Benjamin Betting
aus Lörrach

Frankfurt 2016
(D 30)

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe-Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Uwe Brinkschulte
Gutachter:	Prof. Dr. Uwe Brinkschulte Prof. Dr. Lars Hedrich Prof. Dr. Jürgen Teich
Datum der Disputation:	13. Juni 2016

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Professur für Eingebettete Systeme des Instituts für Informatik an der Johann Wolfgang Goethe-Universität in Frankfurt am Main.
Im Zuge dessen möchte ich den folgenden Leuten meinen Dank aussprechen:

Allen voran meinem Chef und fachlichem Mentor Herr Prof. Uwe Brinkschulte für die Möglichkeit zur Durchführung dieser Dissertation, bei der er mir mit wertvollen Anregungen und Hilfestellungen geholfen hat.

Herrn Prof. Lars Hedrich danke ich, das er die Rolle des Zweitgutachters übernahm und mir während der Projektphase bei technischen Fragen geholfen hat. Ebenfalls danke ich Herrn Prof. Teich in seiner Rolle als Drittgutachter.

Meinen Kollegen Mathias Pacher, Andreas Lund und Frau Linda Stapleton danke ich, das Sie mir den zeitlichen Freiraum für die Beendigung dieser Arbeit eingeräumt haben.

Besonders danken möchte ich meinen Eltern Claudia und Lutz Betting sowie meinem Bruder Florian, dass Sie mir mein Studium ermöglicht haben. Zum Schluss möchte ich noch ganz herzlich meiner Freundin Katharina danken, die mich bei der Durchführung dieser Arbeit ideelle und mit sehr viel Geduld unterstützt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Geschichtliche Entwicklung - Vom Mikroprozessor zum SoC

Den Grundstein für die Entwicklung von SoCs wurde mit der Herstellung des ersten Mikroprozessors, der sogenannten zentralen Recheneinheit eines Computersystems (CPU), in den 70er Jahren gelegt. Dabei wurden die aus Röhren bestehenden analogen Signalprozessoren zunehmend durch Transistor behaftete Typen ersetzt. Bereits in den späten 60er Jahren entwickelten die beiden damaligen, Weltmarkt führenden und konkurrierenden Halbleiter-Firmen Texas Instruments (gegründet als Geophysical Service) und Intel Corporation (gegründet als Fairchild Semiconductor) parallel die Architektur eines Ein-Chip Mikroprozessors der vollständig als integrierter Schaltkreis (IC) gefertigt werden sollte. In Serie schaffte es dann im Jahr 1971 als erstes der von Intel eingeführte i4004, kurz nach dem Texas Instruments sein Modell des TMS1000 wenige Monate zuvor vorgestellt hatte, der aber erst 1972 als SR10-Taschenrechner und anschließend 1974 als eigenständiges Produkt auf den Markt kam [FSH+09, SBN81]. Beide Prozessoren waren auf einfache Steuerungsaufgaben ausgelegt und umfassten ein 4-bit breites Rechen- und Steuerwerk das je mit einer Taktrate von 500 und 750 kHz überzeugte und der Harvard-Architektur entsprach [BU07]. Im Gegensatz zum i4004 umfasste das Schaltungsmodell des TMS1000 neben dem Mikroprozessor auch den Instruktions- (ROM) und Datenspeicher (RAM). Diese Art der Integration läutete zugleich die Ära des Ein-Chip Mikrorechnersystems ein, dass neben den zentralen Recheneinheiten auch die Peripheren-Funktionen des Computersystems mit auf dem IC integrierte und als Mikrocontroller (MCU) bezeichnet wird. Daher gilt in der Literatur häufig der i4004 als erster Mikroprozessor und der TMS1000 als erster Mikrocontroller [Aug83]. Hinsichtlich der Halbleiter-Fertigung (Planarprozess) verfügten die zugehörigen ICs beider Modelle mit der damaligen Fertigungstechnik von 10 µm zwischen 2300 - 8000 integrierten Transistoren. Wie bereits von Gordon Moore im Jahr 1965 prognostiziert, verlief die weitere Entwicklung der nachfolgenden Mikroprozessor- und Mikrocontrollertypen bezüglich der integrierten Transistormenge (Integrationsdichte) gemäß dem von ihm dargelegten Mooreschen Gesetz (siehe [Moo09]). Dies sah eine stetige Verdopplung der integrierten Transistormenge von ICs im wiederkehrenden Zeitraum von 2 Jahren voraus (exponentielles Wachstum). Dementsprechend umfasste beispielsweise der 1978 eingeführte i8086 von Intel bereits knapp 30.000 Transistoren, bei einer Ferti-

gungstechnik von 3 µm und einer internen Taktfrequenz von 5 MHz [MD03]. Abbildung 1.1 zeigt die zeitliche Entwicklung von Mikroprozessoren und deren Systeme (MUCs, SoCs) gemessen an der Integrationsdichte von Transistoren, ausgehend von den frühen 70er Jahren bis zum Beginn des 21.ten Jahrhunderts im Vergleich zu Moores prognostiziertem Verlauf.

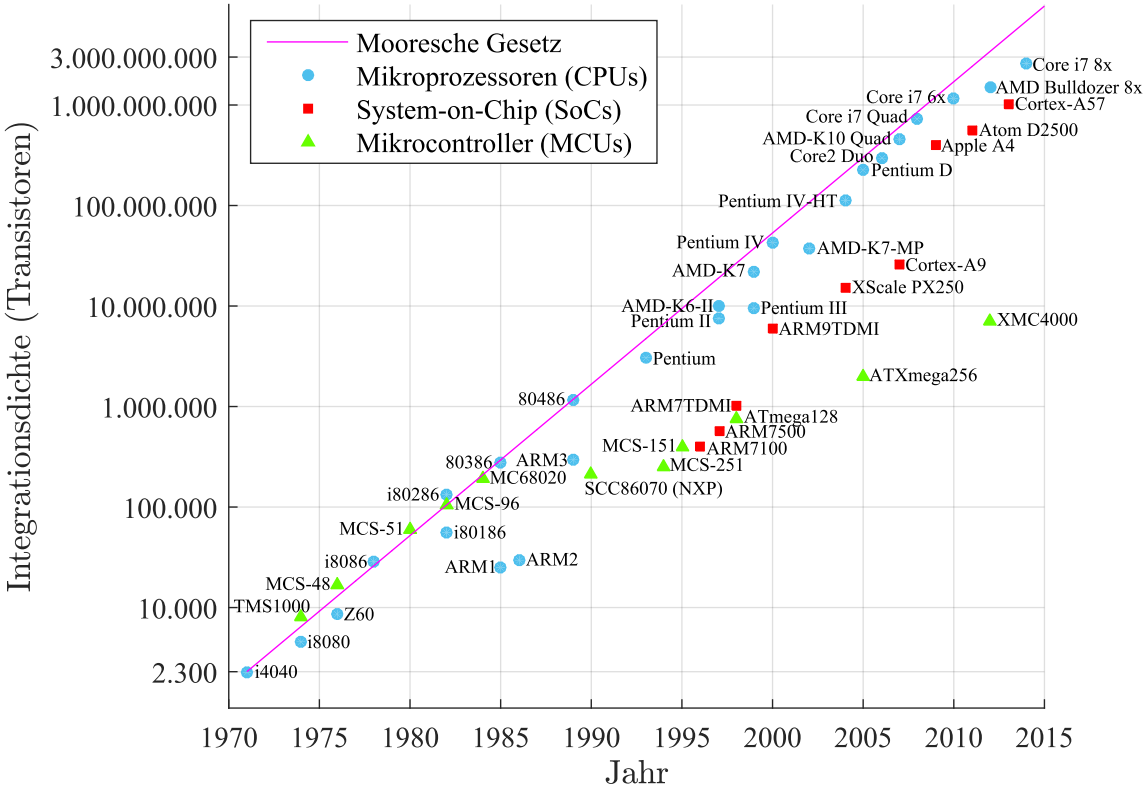


Abbildung 1.1.: Zeitliche Entwicklung der Integrationsdichte bei CPUs, SoCs und MUCs im Vergleich zu Moores Prognose

Bereits kurz nach dem bei Texas Instruments im Jahr 1974 der erste TMS1000 vom Band lief, begann man bei Intel ebenfalls mit der Entwicklung eines Mikrorechners als IC. Dieser feierte mit dem MCS-48 (auch 8048) im Jahr 1976 seine Veröffentlichung. Neben einem 8-bit Mikroprozessor verfügte der IC des 8048 ebenfalls wie der TMS1000 über ein integriertes ROM (1024 Byte) und RAM (64 Byte) [Wak79]. Damit wurde erstmals bei Intel-Chips das Anwenderprogramm direkt auf dem IC gespeichert. Im Jahr 1980 wurde von Intel die Nachfolge-Familie des MCS-51 (8051) eingeführt [Wal08]. Mit dieser Serie kam es zum endgültigen kommerziellen Durchbruch des Mikrocontrollers, da Intel einerseits den integrierten Speicher verdoppelte und andererseits die lizenzierte Fertigung der MCS-Familie an weitere Firmen (Atmel, Infineon, NXP etc.) abgab [Rot02]. Anfangs fanden jedoch aufgrund der hohen Fertigungskosten nur wenige Mikrocontroller den Weg in den Alltag, was

sich aber bei der Einführung der kostengünstigen Varianten 8020, 8021 und 8022 änderte und von da an Mikrocontroller auch zunehmend in Haushaltsgeräten integriert wurden.

Ab den 80er Jahren konzentrierten sich die Halbleiterhersteller zweigleisig auf der Entwicklung von immer leistungsfähigeren Mikroprozessoren und umfangreicheren Mikrocontrollern. Im Jahr 1982 läutete die von David Patterson und Carlo Séquin konzipierte RISC Instruktionssatzarchitektur (ISA) einen Umbruch in der Mikroprozessor-Entwicklung ein [SP82]. Diese zielte darauf ab, die damals in den Mikroprozessoren ausschließlich verwendete komplexe CISC-Architektur durch eine stark vereinfachte ISA zu ersetzen. Dadurch sollte einerseits die Dekodierungs- und Ausführungszeit von Befehlen verkürzt werden und anderseits die im Entwurf benötigte Transistormenge stark reduziert werden. Inspiriert durch dieses Konzept, begann der britische Computerhersteller Acorn, unter der Leitung von Steve Furber und Sophie Wilson, mit der Entwicklung eines leistungsfähigen RISC-Mikroprozessors. Dieser hatte mit der Einführung des ARM1 im Jahr 1985 Weltpremiere und beruhte auf der gleichnamigen ARMv1-Architektur. Mit einer Taktfrequenz von 4 MHz war der ARM1 aufgrund der vereinfachten Befehlsdekodierung mit 2 Millionen verarbeiteten Instruktionen pro Sekunde (2 MIPS) doppelt so schnell wie das Takt gleiche Modelle des MC68000 von Motorolla (1 MIPS). Des Weiteren umfassten die ICs des ARM1 bei einer Fertigungstechnik von 3 µm insgesamt 25.000 Transistoren, was gerade mal 1/3 der Transistormenge des MC68000 entsprach [Fur89]. Dadurch konnten die ICs der ARM1-Serie weitaus mehr zusätzliche Peripherie-Komponenten mit auf dem Chip integrieren, als die bis dahin vertriebenen Mikrocontroller der Hersteller Intel und Motorolla. In den folgenden Jahren wurden die in Serie produzierten Nachfolgemodelle ARM2 und ARM3 in den von Acorn vertriebenen Heimcomputern (Archimedes etc.) eingesetzt, was zu einer zunehmenden Bekanntheit der ARM-Architektur führte. Ende der 80er Jahre war der kommerzielle Erfolg der ARM-Mikroprozessoren bereits so groß, dass sich Acorn mit seinen Partner-Firmen Apple und VLSI zusammenschloss, um von nun an alle zukünftigen ARM-Architekturmodelle (z. B. als IP Cores) ausschließlich als Lizenz zu vertreiben. Im Zuge dessen gründeten sie das ausgelagerte Unternehmen Advanced RISC Machines (kurz ARM Ltd.), welches seitdem den lizenzierten Vertrieb aller ARM-Architekturen und -Designs betreibt [Lan14] . Von da an konnten auch andere Halbleiterhersteller wie Philips, Sony aber auch Intel und Texas eigene, ARM basierte RISC-Mikroprozessoren in Lizenz produzieren.

Der endgültige Durchbruch gelang ARM mit der Einführung der ARM7-Serie, die mit den Modellen ARM7100 und ARM7500 die weltweit ersten SoC Designs auf den Markt brachte (damals noch als Mikrocontroller bezeichnet). Diese SoCs wurden aufgrund ihres geringen Energieverbrauchs in zahlreichen Mobiltelefonen (z. B. Nokia 6110) und anderen portablen Kommunikationsgeräten (z. B. PDAs) eingebaut. Beide SoCs umfassten neben einem 32-bit ARM710a-Mikroprozessor (16 MHz) einen eigenen LCD-Controller (GPU), eine eigene Speicherverwaltungseinheit mit 8

Kilobyte Cache (MMU) sowie zahlreiche weitere Mikrocontroller-Bestandteile (serielle Schnittstellen, Zähler-/Zeitgebereinheit, AD/DA Wandler etc.) [Ltd96]. Mit der Einführung des letzten Modells ARM7TDMI der 7er-Serie wurde dann endgültig zur Jahrtausendwende die Ära der SoCs eingeläutet. Das ARM7TDMI umfasste neben dem eigentlichen RISC-Kern eine leistungsstarke Co-Prozessor Schnittstelle, so dass es mit anderen CISC basierten Prozessoren erweitert werden konnte und daher in zahlreichen Multimediageräten wie dem Gameboy Advance, dem Nintendo DS und den Apple Ipods der 1.Generation verbaut wurde (max. 1.000.000 Transistoren bei 0.35 µm mit 66 MHz) [Fur00]. Mit der Einführung des Synthese fähigen Modells ARM7TDMI-S im Jahr 2001, konnte die ARM-Architektur auch erstmals als Hardwarebeschreibungsmodell (Verilog) erworben werden [Ltd01].

Seite Beginn des 21.ten Jahrhunderts hat die Anzahl der weltweit vermarkteten SoCs stark zugenommen. Laut einer Analyse von Transparency Market Research, lag der weltweite SoC-Marktanteil im Jahr 2014 bei 36 Milliarden US-Dollar. Für das Jahr 2021 wird ein Marktanteil von 80 Milliarden US-Dollar prognostiziert [(TM15]. Heute im Jahr 2016 wird die SoC-Entwicklung und -Produktion von einer Vielzahl unterschiedlicher Halbleiterhersteller übernommen, deren Designs größtenteils auf den neuen Generationen der ARM-Architekturen ARMv5-ARMv8 basieren. Laut einem Bericht von IC Insights, wurden im Jahr 2013 über 95% der Tablets und Smartphones mit ARM basierten SoCs ausgerüstet [II13]. Zu den bekanntesten gehören unter anderem die von Intel und Marvell produzierten XScale und Atom Designs sowie die derzeit leistungsstärksten, ARMv8 basierten SoCs von Qualcomm, wie beispielsweise das Snapdragon 808 und 810 das einen 64-bit ARM Cortex-A57 Mikroprozessor integriert (2.3 GHz, über 1 Milliarde Transistoren bei 22 nm Fertigung) [ANS04, Yiu13].

Laut ARM Ltd. wurden einschließlich bis zum Jahr 2013 weltweit 50 Milliarden ARM-basierte Mikrochips (CPUs, SoCs, MCUs) produziert und verkauft. Davon entfallen 58% auf Mobilgeräte (z. B. Smartphones, Tablets), 20% auf Eingebettete Systeme (z. B. Touchscreen Controller, Smartcards), 16% auf Industriegeräte (z. B. Festplatten, Netzwerk-Interfaces) und 16% auf Heimgeräte (z. B. Kameras, Smart TVs. etc) [Ltd14, MS14].

1.2. Aktuelle Entwicklung - Steigerung der Zuverlässigkeit und Ausfallsicherheit

Durch die rasante Entwicklung der Integrationsdichte (siehe Abbildung 1.1) in den letzten 40 Jahren, hat die Rechenleistung von Mikrochips (SoCs, CPUs, MCUs) immens zugenommen. Was noch in den 90ern Jahren ausschließlich von riesigen Super- oder Desktop-Computern geleistet werden konnte, kann heute von portablen Geräten (Tablets, Smartphones etc.) übernommen werden, die in einem Rucksack oder einer Hosentasche Platz finden. Auch wenn sich die Integrationsdichte in den kommenden

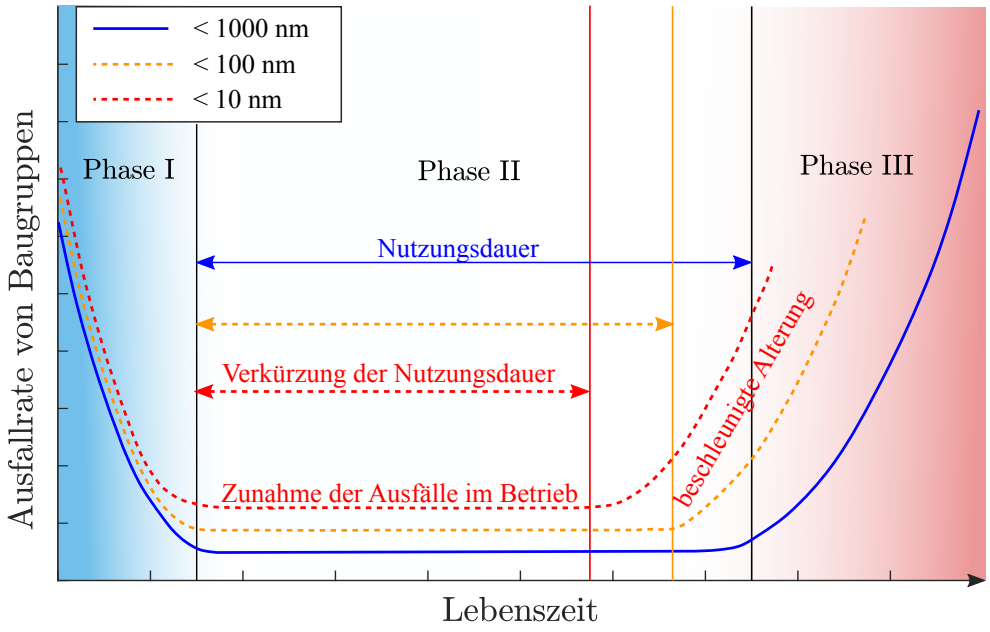


Abbildung 1.2.: Lebenszeit von Mikrochips unter Berücksichtigung der Ausfallrate von Baugruppen bei unterschiedlichen Fertigungsgrößen

Jahren zunehmend nicht mehr nach dem von Moore prognostiziertem Verlauf entwickeln wird [Moo15], so treten bereits bei den jetzigen Strukturgrößen der planaren Fertigungstechnik zunehmend Nebeneffekte auf, welche die Zuverlässigkeit und Lebensdauer von Mikrochips herabsetzen. Am einfachsten lassen sich diese Nebeneffekte anhand der Ausfallrate einzelner Baugruppen von Mikrochips veranschaulichen, verteilt über den gesamten Lebenszyklus. Dabei wird die Ausfallrate der integrierten Baugruppen durch die Einwirkung der folgenden beiden Fehlerquellen beeinflusst:

- Frühausfälle, die auf Fehler im Entwurfs- und Fertigungsprozess zurückzuführen sind (Fehler behaftete oder anfällige Baugruppen).
- Spätausfälle, die auf einen alterungsbedingten Material-Verschleiß zurückzuführen sind (Degradation von Baugruppen).

Dadurch lässt sich die Lebensdauer von Mikrochips anhand der Ausfallrate der integrierten Baugruppen modellieren und auch unter Einschränkungen bedingt vorhersagen. Abbildung 1.2 zeigt die graphische Darstellung des Sachverhalts für unterschiedliche Fertigungsgrößen (1000, 100, 10 nm). Wie deutlich zu erkennen ist, ist die Ausfallrate bei allen drei Fertigungsgrößen zu Beginn und am Ende des Lebenszyklus am höchsten. Die mittlere Zyklusphase entspricht der eigentlichen Nutzungsdauer des Mikrochips, weshalb die Ausfallrate in diesem Zeitraum am geringsten ist (annähernd konstant). Nach der Inbetriebnahme in Phase I, ist bei allen drei Größen der Ausfall an Produktionsfehler behafteten Baugruppen ähnlich hoch und

stetig abnehmend. Der erneute Anstieg der Ausfallrate am Ende der Nutzungsdauer in Phase II, ist auf die zunehmende Degradation von Baugruppen zurückzuführen, was dann auch letztlich zum totalen Ausfall des Mikrochips am Ende von Phase III führt. Diese Degradation verläuft bei immer kleineren Fertigungsgrößen (< 100 nm) tendenziell schneller. Des Weiteren ist bei kleineren Fertigungsgrößen die konstante Ausfallrate während der Nutzungsdauer höher, so dass bereits in Phase II mit einem erhöhten Ausfall von Baugruppen zu rechnen ist [Dhi13]. Diese negativen Auswirkungen lassen sich unter anderem auf die folgenden Nebeneffekte zurückführen, die bei den Fertigungsgrößen heutiger Chip-Designs zunehmend zu beobachten sind:

Zunahme von Elektromigration: Die Fertigung kleinerer Strukturgrößen führt zum verstärkten Auftreten von Elektromigration. Dieses Phänomen wurde bereits im frühen 20.ten Jahrhundert entdeckt und trat erstmals bei der kommerziellen Fertigung von ICs in den 70er Jahren in Erscheinung [Ble76]. Damals wurde beobachtet, dass die im Zuge der Miniaturisierung ansteigende Stromdichte die natürliche Diffusion der dotierten Leitermaterialien in ICs verstärkt. Dies hat vor allem auf die Struktur der Schaltungs verknüpfenden Leiterbahnen deformierende Auswirkungen. Der Effekt ist einfach zu beschreiben: Durch den sich verringernden Querschnitt von Leiterbahnen, nehmen die Kollisionen zwischen den beweglichen Elektronen und den statischen Metallionen der Leiterbahnen stark zu. Dabei erfolgt eine Impulsübertragung von Elektronen auf die Metallionen, wodurch ein passiver Metalltransport entsteht [Kim11]. Dies führt bei zunehmender Nutzungsdauer zu einer Ausdünnung und Anlagerung von Ionen an bestimmten Stellen, was letztlich zu einer strukturellen Unterbrechung der Leiterbahn führt. Nach der Beobachtung dieses Effekts, wurde die Elektromigration durch die Verwendung von resistenten Leitermaterialien (z. B. Kupfer, Gold, Silber) weitestgehend beherrschbar [Ryu99]. Durch den enormen Sprung des planaren Fertigungsprozess in den zweistelligen Nanometer-Bereich, tritt die Diffusion jedoch auch bei resistenten Leitermaterien zunehmend in Erscheinung, so dass die Elektromigration in der heutigen Halbleiterfertigung erneut an Bedeutung gewinnt.

Sensitivität gegenüber Strahlung: Die kleineren Strukturgrößen führen zu einer erhöhten Störanfälligkeit der Baugruppen gegenüber ionisierender Strahlung (z. B. kosmische Strahlung, atomare Strahlung). Diese Strahlung emittiert Ionen (geladene Teilchen), die beim Durchdringen der Chipstruktur (DIE) ihre Energie an die Baugruppen abgeben. Wird dabei der energetische Schwellwert (LET_{th}) überschritten, ruft dies einen entsprechenden Schaltungs-Effekt hervor, dessen Auswirkungen entweder zu einer vorübergehenden Störung oder einer dauerhaften Schädigung der ionisierten Baugruppe führt. Bei kleineren Fertigungsgrößen nimmt der energetische Schwellwert stetig ab, so dass während des Lebenszyklus zunehmend mit Beeinträchtigungen durch ionisierende Strahlung auftreten, die im Allgemeinen als Single

Event Effects (SEE) bezeichnet werden [Ibe15]. Zu den wichtigsten im digitalen Schaltungsbereich zählen:

- **Single Event Upset (SEU):** Verursacht eine willkürliche Zustandsänderung, sogenannte Bit-Flips, in Speicherzellen oder Registern. Diese können dann zu einem Fehler auf Softwareebene führen (fehlerhafter Programmablauf). Ein SEU kann aber auch zum vorübergehenden Systemausfall führen, wenn durch die Ionisierung spezifische Steuer- oder Kontrollmechanismen (State Machines, Steuerungs-Agenten etc.) in einen undefinierten Zustand versetzt werden, der einen Neustart des Systems erfordert [Ada84].

- **Single Event Latchup (SEL):** Verursacht das Zünden (Durchschalten) parasitärer Thyristor-Strukturen[1] des Chips, was einen Kurzschluss der Versorgungsspannung von Baugruppen hervorruft (Latch-Up-Effekt). Der dadurch fließende Strom kann zur thermischen Überbelastung und dauerhaften Zerstörung der Baugruppen führen [NCW+92].

- **Single Event Burnout (SEB):** Verursacht das ungewollte Durchschalten von gesperrten (Leistungs-)Transistor-Baugruppen bei gleichzeitig hoch anliegender Versorgungsspannung. Der überhöhte Stromfluss kann dann zur thermischen Überbelastung und dauerhaften Zerstörung der Baugruppe führen, falls die hohe Versorgungsspannung nicht zurückgefahren wird [Joh93].

- **Single Event Gate Rupture (SEGR):** Verursacht die dauerhafte Zerstörung der Potential-Barriere in Transistor-Baugruppen (Gateoxid) wodurch eine leitende Verbindung entsteht. Der dadurch fließende Strom kann ähnlich wie bei SEB zur thermischen Überbelastung und dauerhaften Zerstörung der Baugruppe führen [SPJ94].

Zunahme thermischer Verlustleistung: Bei kleineren Fertigungsgrößen nimmt die Anzahl an Leckströmen zu. Dadurch erhöht sich die thermische Verlustleistung, welche die Baugruppen zusätzlich beansprucht. Leckströme treten vermehrt bei kleineren Strukturgrößen auf, da die parasitären Schaltungskomponenten im Zuge der niedrigeren Versorgungsspannung häufiger zünden. Obwohl durch die Einführung des Tri-Gate Transistors die Anzahl an Leckströmen deutlich reduziert werden konnte [Vai10], wird aufgrund der weiter steigenden Integrationsdichte die durch Leckstrom verursachte Verlustleistung zunehmen. Des Weiteren führen die neueren 3D-Integration dazu, dass die Abfuhr der Verlustleistung im Chip zusätzlich erschwert wird. Dies führt im Gesamten zu einem höheren Temperaturverhalten von Baugruppen, wodurch im Chip sogenannte Hot-Spots entstehen können [EFH09], die entweder zu einer unmittelbaren Zerstörung (z. B. Temperatur-Overblow) oder

[1] Ein parasitärer Thyristor ist eine ungewollte, Transistor ähnliche Schaltungskomponente, die bei der Dotierungen von Feldeffekttransistoren entsteht.

einer langfristigen Schädigung (positiver Rückkopplungseffekt auf Elektromigration und Strahlungs-Sensitivität) von Baugruppen führt [KT98].

1.3. Ziele und Struktur der Arbeit

Das Ziel der vorliegenden war die Entwicklung und Evaluation eines organischen Taskverarbeitungssystems, welches die Zuverlässigkeit von Multi-Core basierten SoC Designs gegenüber Hardware- und Software seitigen Fehler und Ausfällen erhöht. Dabei übernimmt das System einerseits die intelligente Verwaltung der System-Aufgaben und -Ressourcen und andererseits die reaktive und proaktive Adaption von Systemveränderungen, die im Zuge der Miniaturisierung zunehmend durch Fehler und Ausfälle herbeigeführt werden (siehe Abschnitt 1.2). Hierfür nutzt das System Techniken die sich am Verhalten der natürlichen Systeme (endokrine Hormonsystem) inspirieren, wodurch ein organisches Funktionsprinzip mit adaptierten, natürlichen Fähigkeiten entsteht. Diese Arbeit ist daher im Bereich des Organic Computing anzusiedeln (siehe DFP SPP 1183 [Sch11]) und entstand im Rahmen des MixedCoreSoC-Forschungsprojekts [vRSH+15], welches als Teil des DFG-Schwerpunktprojektprogramms SPP 1500 [HBB+11] gefördert wurde. Dieses Projektprogramm hat es sich zum Ziel gemacht, die zunehmend negativen Auswirkungen der Miniaturisierung zu erforschen um geeignete Architekturen und Methoden auf Hardware- und Softwareebene zu entwickeln, welche die Zuverlässigkeit von eingebetteten Systemen nachhaltig verbessern sollen. Zur Steigerung der Fehler- und Ausfalltoleranz nutzt das System ein dezentrales Regelungskonzept, das die folgenden aufgeführten Selbst-X Eigenschaften umsetzt:

- Selbst-Konfiguration: Das System findet selbständig nach der Inbetriebnahme eine gültige Konfiguration unter Berücksichtigung und Einhaltung der Anforderungen und Aufgaben.

- Selbst-Heilung: Das System ist in der Lage auf etwaige Fehler und Ausfälle von Komponenten (z. B. Kerne, Tasks etc.) mittels geeigneter Maßnahmen zu reagieren, so dass erneut ein funktionsfähiger Systemzustand gefunden wird.

- Selbst-Optimierung: Das eingeschwungene System passt sich selbstständig an Veränderungen an die durch interne oder externe Einflüsse verursacht werden. Dadurch wird stetig versucht die bereits gefundene Konfiguration zur verbessern.

Des Weiteren sind einige Implementierungs bezogene Teile der Arbeit in Kooperation mit dem Remis-Projekt entstanden [LPB11], welches sich mit der Entwicklung eines Regelungssystems zur Steuerung von Durchsatzraten in mehrfädigen Multi-Core Architekturen beschäftigt.

Der Aufbau der Arbeit ist wie folgt: Kapitel 1 motiviert das Thema. Hierbei wird ein Rückblick über die bisherigen und aktuellen Entwicklungen im Bereich von SoCs gegeben, wobei das Problem der abnehmenden Zuverlässigkeit der Systemen im Zuge der Miniaturisierung erläutert wird. In Kapitel 2 wird die Konzeption und Spezifikation des biologisch inspirierten Taskverarbeitungssystem beschrieben. Ausgehend von der grundlegenden SoC Hardwarearchitektur, werden die Anforderungen und Aufgaben des Systems abgeleitet. Anschließend wird deren Umsetzung mittels dezentralem Steuerungskonzept erörtert, wobei bestimmte Eigenschaften des menschlichen, endokrinen Hormonsystems in adaptierter Form übernommen werden. Dabei werden auch die Unterschiede der adaptierten Funktionsweise aufgezeigt. Das organische Funktionsprinzip des Systems wird in Kapitel 3 vorgestellt. Neben grundlegenden Begriffs- und Notationsdefinitionen wird die dezentrale Funktionsweise der zuverlässigen Verwaltung und Verarbeitung von Tasks dargelegt. Kapitel 4 umfasst die theoretischen Grundlagen des Systems. Dabei wird das hormongestützte Modell der Taskverarbeitung vorgestellt, sowie dessen Funktionsweise und Echtzeiteigenschaften formal analysiert und bewiesen. In Kapitel 5 wird die Implementierung des Systems vorgestellt. Neben dem synchronen Zeitmodell, wird die Umsetzung der unterschiedlichen Komponenten auf Hardware- und Systemebene gezeigt. Abschließend erfolgt eine quantitative Analyse des Worst-Case Zeitverhalten in Abhängigkeit der frei wählbaren Konfigurationsparameter, wodurch das Zeitverhalten des Systems auch rechnerisch ermittelt werden. Kapitel 6 zeigt die Auswertung der Selbst-X Eigenschaften des Systems auf einem voll entwickelten, FPGA-basierten Prototypen mit mehreren Messreihen. Der aktuelle Stand der Forschung, sowie eine quantitative Vergleichsanalyse gegenüber anderen Systemen wird in Kapitel 7 gegeben. In Kapitel 8 erfolgt die abschließende Zusammenfassung über die erzielten Ergebnisse der Forschungsarbeit sowie ein Ausblick auf zukünftige Entwicklungen. Der Abschluss der Arbeit erfolgt in Anhang A, in welchem die nebensächlichen Informationen zu den simulierten Fehlermodellen in den Messreihen von Kapitel 7 aufgeführt sind.

2. Biologisch inspiriertes System

In diesem Kapitel soll das Konzept des biologisch inspirierten Taskverarbeitungssystem vorgestellt werden. Ausgehend von der grundlegenden SoC Hardware-Architektur in Abschnitt 2.1, werden die zentralen Aufgaben und Anforderungen des Systems in Abschnitt 2.2 abgeleitet. Abschließend wird in Abschnitt 2.3 die Übertragung bestimmter Eigenschaften aus dem menschlichen endokrinen Hormonsystem auf das organische System erörtert.

2.1. Grundlegende SoC Architektur

Für das in dieser Arbeit vorgestellte organische Taskverarbeitungssystem erfolgt die Betrachtung und Definition einer generalisierten SoC-Architektur. Diese stellt beim Systementwurf die Ausgangssituation auf Hardwareebene dar und berücksichtigt die Umsetzung einer zuverlässigen SoC-Applikation als Redundanz gestützte, heterogene Multi-Core Architektur. Ausgehend von der eigentlichen SoC-Applikation, erfolgt die Distribution der gesamten SoC-Funktion in lokale Recheneinheiten, den sogenannten *Kernen*. Diese übernehmen die Ausführung der eigentlichen System-Aufgaben, den sogenannten *Tasks*, in integrierten Schaltkreisen und Funktionseinheiten. Dadurch erfolgt eine Umsetzung der SoC-Applikation als Network-on-Chip (NoC), wobei jeder Kern einen selbständigen Rechenknoten darstellt, der unabhängigem Zugriff auf ein globales Verbindungsnetzwerk erhält, um mit anderen Kernen zu kommunizieren. Abbildung 2.1 zeigt eine Darstellung der SoC-Architektur mit einer beispielhaften Applikation als autonome Fahrtregelung. Unter Berücksichtigen dieser abstrahierten Sichtweise der SoC-Applikation kann eine generalisierte SoC-Architektur definiert werden, die sich im Wesentlichen aus den folgenden beiden Systemkomponenten zusammensetzt:

- **Task:** Repräsentiert eine (Teil-)Aufgabe der SoC-Applikation, die den Kernen des Systems zugeordnet wird.
- **Kern:** Repräsentiert eine lokale Recheneinheit, welche die Ausführung von einer oder mehrerer zugeordneter Tasks im System übernimmt.

Während des Systembetriebes erfolgt also eine Zuteilung von Tasks auf Kerne. Um diese Zuteilung zu ermöglichen, ist je nach SoC-Applikation eine unterschiedliche Ordnung (Typisierung) zwischen Tasks und Kernen erforderlich. Dadurch können

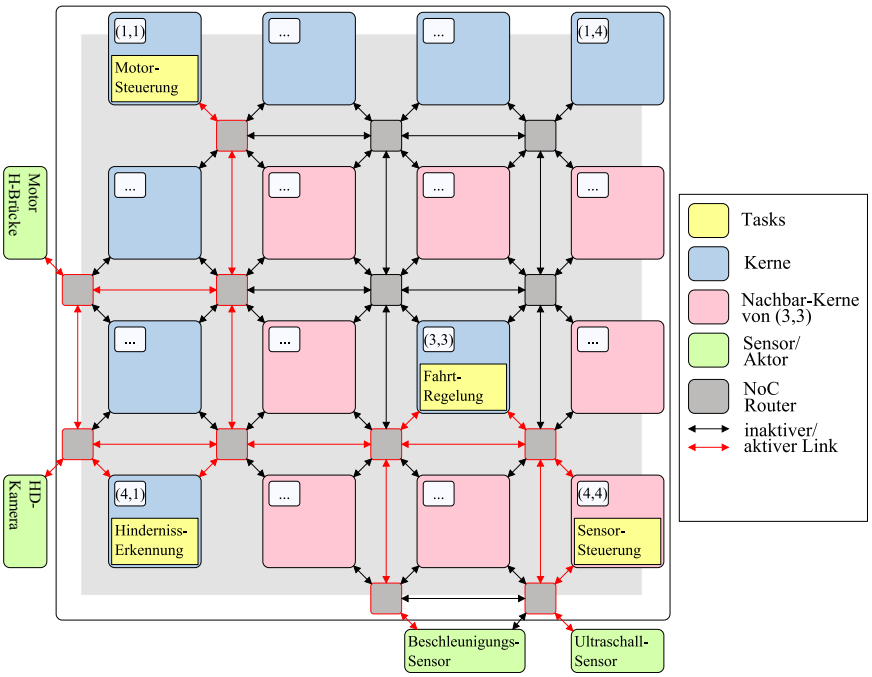


Abbildung 2.1.: Beispiel SoC-Applikation mit generalisierter Architektur

die funktionalen Eigenschaften der Kerne individuell den Anforderungen der Tasks angepasst werden, so dass für die Ausführung einer bestimmten Task eine bestimmte Menge an Kernen zur Verfügung steht. In Publikation [BvRBH13] wird eine solche Ordnung für die Umsetzung einer Mixed-Signal SoC-Applikation vorgestellt. Unter Berücksichtigung des obigen Generalisierungskonzepts erfolgt eine Abstraktion der SoC-Funktion in analoge und digitale Kerne. Weiterführend erfolgt eine Typisierung von Tasks und Kernen, die eine eindeutige Zuordnung zwischen beiden Komponenten vollzieht und eine robuste Taskverarbeitung gegenüber hardwareseitigen Ausfällen in beiden Signalbereichen ermöglicht.

Der Entwurf des organischen Taskverarbeitungssystem erfolgt universell für die obige, generalisierte Systemarchitektur. Im weiteren Verlauf wird deshalb von einer Ordnung zwischen Tasks und Kernen ausgegangen, die eine Zuteilung aller Tasks auf allen Kerne des Systems ermöglicht, wobei die lokale Task-Aufnahmekapazität eines Kerns begrenzt ist. Die Umsetzung der Zuverlässigkeit innerhalb der generalisierten Systemarchitektur erfolgt somit durch die redundante Verfügbarkeit von Kernen. Je nach Höhe der Redundanz kann somit eine unterschiedliche Toleranz des Systems gegenüber den folgenden Ausfällen erzielt werden:

- **Taskausfall:** Ein Taskausfall definiert den vorzeitigen Abbruch der Ausführung einer bestimmten Task auf einem bestimmten Kern. Tritt dieses Ereignis ein, wird die lokal erzeugte Instanz der Task auf dem zugeordneten Kern terminiert und der Kern verliert gleichermaßen die Zuordnung der Task.

- **Kernausfall:** Ein Kernausfall definiert den dauerhaften oder vorübergehenden Ausfall eines bestimmten Kerns. Als Folge dessen wird die Ausführung aller auf diesem Kern zugeordneten Tasks vorzeitig abgebrochen, so dass für die betroffene Taskmenge eine Folge von Taskausfällen entsteht. Handelt es sich um einen vorübergehenden Ausfall (z. B. infolge einer Überhitzung des Kerns), so erfolgt nach diesem die Reaktivierung des betroffenen Kerns und eine erneute Zuordnung von Tasks auf den Kern ist möglich.
  $\Rightarrow$ Der Ausfall eines Kerns führt gleichermaßen zum Ausfall dessen zugeordneter Taskmenge.

Beide Ausfallarten können durch Störungen auf Hardwareebene (z. B. lokale Bit-Flips, Alterung von Funktionseinheiten) verursacht werden, die gemäß dem Fehlerkonzept in Literatur [HBB+11] zunächst zu einem lokalen Fehler (z. B. Rechenfehler einer Task) und letztlich zum jeweiligen Ausfall führen können. Bei dem in dieser Arbeit vorgestellten Taskverarbeitungssystem kann die Detektion von Task- und Kernausfällen einerseits passiv durch das organische Funktionsprinzip in Kapitel 3 erfolgen und anderseits mittels zusätzlicher Monitoring-Einheiten.

Unter Berücksichtigung der generalisierten Architektur und den oben genannten Ausfallarten, können für die Umsetzung der Selbst-X Eigenschaften -Konfiguration, -Optimierung und - Heilung die folgenden Funktionsweisen abgeleitet werden:

- **Selbst-Konfiguration:** Bei der Selbst-Konfiguration erfolgt die initiale, selbständige Zuordnung aller Tasks auf die verfügbaren Kerne des Systems. Diese ist abgeschlossenen sobald sich alle Tasks auf den Kernen in der Ausführung befinden.

- **Selbst-Optimierung:** Bei der Selbst-Optimierung erfolgt die globale Optimierung der aktuellen Taskverteilung. Hierbei wird die Zuordnung von Tasks überprüft und gegebenenfalls durch Migration der betroffenen und bereits ausgeführten Tasks auf andere Kerne optimiert. Die Optimierung ist beendet sobald die Migration der betroffenen Taskmenge beendet ist.

- **Selbst-Heilung:** Bei der Selbst-Heilung erfolgt eine re- oder proaktive Neuzuordnung der ausfallbehafteten Tasks auf die verbleibenden Kerne des Systems. Die Heilung ist abgeschlossen, sobald sich alle ausfallbehafteten Tasks erneut in der Ausführung befinden.

2.2. System Anforderungen und Aufgaben

Unter Berücksichtigung der generalisierten SoC Architektur aus Abschnitt 2.1 können die folgenden System-Anforderungen und -Aufgaben festgehalten werden.

Das System soll die folgenden Anforderungen erfüllen:

- Es soll die vorgegebene Redundanz von Kernen maximal ausnutzen, so dass der Systembetrieb sichergestellt ist, solange genügend Task verarbeitende Kerne im System verfügbar sind.
- Es soll die Selbst-X Eigenschaften -Konfiguration, -Optimierung und Heilung gemäß den in Abschnitt 2.1 vorgestellten Funktionsprinzipien umsetzen.
- Es soll vollständig auf den Einsatz zentraler Steuerungskonzepte verzichten, so dass der Bestandteil eines Single of Point Failure vollständig vermieden wird.
- Es soll Veränderungen (z. B. Ausfälle von Tasks u. Kernen) bezüglich des Systemzustands selbstständig erkennen und mittels der Selbst-X Eigenschaften -Heilung und -Optimierung re- oder proaktiv adaptieren.
- Es soll die hier aufgeführten Anforderungen Ressourcen schonend umsetzen (geringer Speicherverbrauch, geringe Rechenzeit, Flächenaufwand etc.), so dass ein Einsatz auf eingebetteten Systemen möglich ist.

Des Weiteren lassen sich für das System die folgenden, zentralen Aufgaben ableiten:

- Die dezentrale Regelung der Verteilung aller Tasks auf die vorhandenen Kerne im System, unter Berücksichtigung des aktuellen Systemzustands.
- Die dezentrale Regelung der Verarbeitung von Tasks auf den Kernen gemäß der aktuellen Taskverteilung. Hierbei sollen die Tasks dynamisch zwischen den Kernen im System transferiert werden, so dass eine zügige Synchronisation zwischen der Zuordnung und Ausführung von Tasks erfolgt.

2.3. Dezentrales, Hormon gestütztes Systemkonzept

Für die Umsetzung der in Abschnitt 2.1 definierten Anforderungen und Aufgaben erfolgt der Entwurf eines dezentralen Steuerungskonzepts für die Verteilung und Verarbeitung von Tasks, das sich an dem menschlichen, endokrinen Hormonsystem inspiriert und versucht bestimmte Eigenschaften in adaptierter Form auf die SoC-Architektur zu übertragen.

2.3.1. Funktionsprinzip und Eigenschaften des endokrinen Hormonsystems

Das endokrine Hormonsystem übernimmt neben dem Nervensystem die Koordination von Aufgaben im menschlichen Körper, die wiederum im inneren von Zellen und Organen ausgeführt werden. Es repräsentiert daher ein globales Nachrichtensystem, das den Austausch von Informationen zwischen Körperzellen ermöglicht, um somit deren Aktivität zu koordinieren [Gol04]. Dies erfolgt über die Ausschüttung von Botenstoffen, den sogenannten Hormonen. Dies sind chemisch niedermolekulare Verbindungen (Signale) die über Hormondrüsen (z. B. Hypothalamus) erzeugt und in den Blutkreislauf ausgeschüttet werden, um sich im ganzen Körper verteilen zu können. Dieser Vorgang dauert mehrere Minuten. Ist ein Hormon an einer Zelle angekommen wird es über ein zugehöriges Rezeptorprotein (Sensoren) aufgenommen, das je nach Zelltyp unterschiedlich auf der Zellmembran ausgeprägt ist. Dadurch wird das Reaktionsverhalten von Zellen gegenüber Hormonen gesteuert. Besitzt die Zelle für das Hormon einen zugehörigen Rezeptor, so wird es über diesen aufgenommen und über weitere Kanäle ins Zellinnere (Zellkern) transportiert. Andernfalls zeigt die Zelle bezüglich des Hormons keinerlei Reaktionsverhalten. Im Zellkern angekommen entfaltet das Hormon seine Wirkung, in dem es eine spezifische Funktion der Zelle aktiviert. Um eine Regelung der Zellfunktion zu realisieren, besitzt das Hormonsystem mehrere direkte und indirekte Rückkopplungsmechanismen. Diese regulieren den Ausstoß des entsprechenden Hormons und somit letztlich dessen Konzentration im Blutkreislauf. Wurde ein hinreichend hohes Reaktionsverhalten von Zellen registriert, erfolgt eine Reduktion des Hormonausstoßes in den zugehörigen Drüsen. Andernfalls erfolgt ein erhöhter Hormonausstoß. So wird beispielsweise der Blutzuckerspiegel (Glucosekonzentration) des Menschen durch die Ausschüttung eines entsprechenden Proteo-Hormons (Insulin) in der Bauchspeicheldrüse reguliert. Dabei aktiviert das ausgeschüttete Insulin in den Zellen den Transport von Glucose. Dadurch erfolgt eine Einlagerung der im Blut verfügbaren Glucose in die Zellen, wodurch im Gegenzug die Glucosekonzentration im Blut absinkt. Dabei induziert und hemmt Insulin gleichermaßen die Glucose-Produktion (Glykogensynthese) und -Speicherung in den Zellen.

Die Funktionsweise des Hormonsystems ist vollständig dezentral. Zellen arbeiten zwar gemeinsam, aber in selbständiger und unabhängiger Form an einer oder mehreren Aufgaben. Die Koordinaten dieser Aufgaben erfolgt über die Ausschüttung von Hormonen, wodurch die benötigte Funktion in den Zellen aktiviert wird. Die Regelung dieser Zellfunktion erfolgt aber nicht zentral in einer einzigen Zelle, sondern gemeinsam in allen, weshalb alle Zellen im System sich wechselseitig beeinflussen können. Über einen entsprechenden Rezeptor auf der Membran kann jede Zelle dann selbst entscheiden, ob Sie auf ein bestimmtes Hormon reagiert oder nicht. Dadurch ist das Hormonsystem sehr robust, weil der Ausfall einer einzelnen Zelle einerseits nicht den Informationsaustausch zwischen den anderen Zellen stört und an-

derseits nicht die Umsetzung der eigentlichen Aufgabe blockiert, solange genügend weitere Zellen mit der erforderlichen Zellfunktion zur Verfügung stehen. Folglich lassen sich anhand des hier beschriebenen, endokrinen Funktionsprinzips zwei herausragende Eigenschaften isolieren, die für das organische Taskverarbeitungssystem wünschenswert sind:

- **Dezentrale Regelung der Zellaktivität:** Durch die Ausschüttung von Hormonen erfolgt eine dezentrale Regelung der Aktivität von Zellen im Körper.
- **Toleranz gegenüber Zellausfällen:** Funktioniert eine Zelle gar nicht oder nur teilweise, so wird die Aufgaben von einer funktionsfähigen Zelle gleichen Typs übernommen und fortgesetzt.

2.3.2. Adaption der endokrinen Systemeigenschaften

Eine direkte Übertragung der oben aufgeführten Eigenschaften auf das organische Taskverarbeitungssystem ist aufgrund der komplexen, technischen Anforderungen sowie der begrenzt verfügbaren System-Ressourcen nicht möglich. Daher werden bei der Umsetzung der zentralen Aufgaben die folgenden Funktionsprinzipien verwendet, welche die endokrinen Eigenschaften in abgewandelter Form übernehmen. Dabei gilt zwischen den zentralen Komponenten des organischen und endokrinen Systems, die folgende Abstraktion:

- **Kern $\Leftrightarrow$ Zelle:** Die Kerne repräsentieren abstrakt die Körper-Zellen, die in der Lage sind die Aufgaben (Tasks) im System auszuführen.
- **Task $\Leftrightarrow$ Zellaufgabe:** Die Tasks repräsentieren die Zellaufgaben die von den Kernen ausgeführt werden.
- **Netzwerknachrichten $\Leftrightarrow$ Hormone:** Die Netzwerknachrichten die zwischen Kernen und Tasks kommuniziert werden, repräsentieren die ausgeschütteten Hormone.

Adaptionsform für die dezentrale Taskverteilung: Für die Umsetzung der dezentralen Taskverteilung erfolgt eine Adaption der endokrinen Eigenschaften nach dem folgendem Schema: Alle Kerne im System übernehmen gleichermaßen die Verteilung aller Tasks, wobei im Konkreten jeder Kern selbst über die lokale Zuordnung seiner Taskmenge entscheidet. Dazu schütten alle Kerne Hormone aus, die Informationen enthalten für welche Tasks jeder Kern sich interessiert, wie dessen funktionale Eigenschaften sind und wie der aktuelle Kernzustand ist. Auf Basis dieser Informationen, entscheidet dann jeder Kern für sich selbst und unter Berücksichtigung des Entscheidungsverhaltens aller anderen Kerne, welche Task er übernimmt. So entsteht eine globale Taskverteilung durch das Mitwirken aller Kerne und es lassen sich

die folgenden Parallelen (•) und Unterschiede (◦) zur endokrinen Funktionsweise ableiten:

- Die negative Rückkopplung innerhalb der Taskverteilung erfolgt durch die zunehmende Ausschüttung weiterer Hormone, so dass die Verteilung zum Erliegen kommt, sobald alle Tasks verteilt wurden.

- Das Senden von Hormonen erfolgt über das globale Kommunikationsnetzwerk. Dabei kann ein Hormon entweder an alle oder nur eine Gruppe von Tasks und Kernen gesendet werden.

- Das Reaktionsverhalten eines Kerns auf ein bestimmtes Hormonen ist davon geprägt, ob sich der Kern für die Task interessiert, für welche das Hormon bestimmt ist. Andernfalls wird das Hormon von dem jeweiligen Kern ignoriert.

- Fällt ein Kern aus, so wird die Taskverteilung nicht negativ tangiert, solange genügend andere Kerne im System vorhanden sind. Durch den Ausfall eines Kerns, wird die hormonelle negative Rückkopplung vermindert was eine erneute Verteilung der betroffenen Taskmenge auf die restlichen Kerne hervorruft.

- ◦ Da das globale Kommunikationsnetzwerk im Gegensatz zum menschlichen Blutkreislauf nur über eine begrenzte Bandbreite verfügt, erfolgt das Senden von Hormonen stark begrenzt (zyklisch). Damit ein Hormon dennoch ab dem ersten Empfang seine volle Wirkung bezüglich seiner zugehörigen Task auf einem bestimmten Kern entfaltet, wird dessen Intensität über einen entsprechenden Hormonwert reguliert, der in der Hormon-Nachricht abgelegt ist. Im Gegensatz zum endokrinen System wird die Intensität der Hormon-Wirkung somit nicht über die Quantität des Hormons sondern über dessen Wert reguliert.

Adaptionsform für die dezentrale Taskverarbeitung: Für die Umsetzung der dezentralen Taskverarbeitung erfolgt eine Adaption der endokrinen Eigenschaften nach dem folgendem Schema: Alle Kerne übernehmen die Verarbeitung von Tasks entsprechend der aktuellen Zuordnung in zugehörigen Instanzen, die lokal auf den Kernen ausgeführt werden. Für die Erzeugung einer lokalen Instanz, muss die Task auf dem jeweiligen Kern in gespeicherter Form verfügbar sein. Da aber die Verteilung von Tasks dynamisch über den Austausch von Hormonen erfolgt und die Task-Aufnahmekapazität von Kernen begrenzt ist, erfolgt eine verteilte Speicherung aller Tasks auf allen Kernen. Dabei wird jede Task primär auf einem bestimmten Kern initial gespeichert und sekundär auf verschiedenen Kernen mehrfach gesichert. Dies erfolgt unabhängig davon, ob ein Kern seine gespeicherten Tasks zu einem späteren Zeitpunkt tatsächlich selbst übernimmt und ausführt. Damit alle Tasks trotz der verteilten Speicherung global für alle Kerne verfügbar sind, werden Hormone gesendet

die den Transfer der Tasks zwischen den Kernen ermöglichen. Des Weiteren werden diese Hormone dazu verwendet um die Kommunikation zwischen kooperierenden Tasks und den Austausch von Zustandsdaten zu ermöglichen. Dadurch entsteht eine globale Verarbeitung aller Tasks in lokalen Instanzen und es lassen sich die folgenden Parallelen (•) und Unterschiede (◦) zur endokrinen Funktionsweise ableiten:

- Tasks werden als Hormone aufgefasst, die für die Erzeugung und Ausführung innerhalb einer lokalen Instanz global über das Kommunikationsknetwerk transferiert werden.
- Alle Tasks sind trotz verteilter Speicherung global für alle Kerne verfügbar. Fällt ein Kern aus, wird dessen initial gespeicherte Taskmenge aus der verteilten Sicherungsmenge der restlichen Kerne bezogen.
- Tasks können durch den Austausch von Hormonen berechnete Ergebnisse und Zustandsdaten mit anderen kooperierenden Tasks austauschen, wodurch die Umsetzung einer gemeinsamen Aufgabe erfolgt.
- Die Ausführung von Tasks erfolgt auf den Kernen in lokalen Instanzen. Dadurch werden alle zugehörigen Task-Informationen direkt auf dem zugeordneten Kern gespeichert, so dass unter anderem eine mehrfache und unabhängige Ausführung der selben Task im System möglich ist, falls gewünscht.
- ◦ Für den Transfer von Tasks müssen Hormone Informationen aufnehmen, die zwar das konkrete Funktionsverhalten eines Kerns nicht definieren aber durch entsprechende Direktiven (Blaupause der Task) instruieren um die Erzeugung und Ausführung lokaler Instanzen zu ermöglichen. Deshalb werden im Gegensatz zum endokrinen System die Hormone auch dazu verwendet, um Informationen bezüglich der eigentlichen Task-Struktur zwischen den Kernen zu übertragen.

3. Organische System Architektur

3.1. Das organische Taskverarbeitungssystem

Das organische Taskverarbeitungssystem setzt sich aus dem Zusammenspiel zweier Mechanismen zusammen, die separiert die Zuordnung und Verarbeitung von Tasks auf der in Kapitel 2 vorgestellten zuverlässigen SoC-Architektur regelt und überwacht:

Das *künstliche Hormonsystem* (KHS[1]) ist eine Middleware die im Rahmen der Forschungsarbeiten [vR12] und [BBB+06] entwickelt wurde und die Verteilung von Tasks auf die im Netz vorhandenen Kerne übernimmt. Sie ist somit für die Umsetzung der *Taskverwaltung* im System verantwortlich und wird im Folgenden auch als solche assoziiert. Die Taskverteilung erfolgt durch das Mitwirken aller verfügbaren Kerne vollständig dezentral. Dabei tauschen die Kerne untereinander ihre Taskeignung mittels Hormonen aus, um sich so gegenseitig für eine Menge an Tasks zu bewerben. Auf Basis dieser Hormone entscheiden dann alle Kerne lokal, ob deren Eignung zur Übernahme der beworbenen Tasks berechtigt. Dadurch erfolgt die Taskzuordnung stets gegenüber den Kernen, die aus Sichtweise der Taskausführung die bestgeeignete Wahl im Netz darstellen. Dieser Prozess erfolgt schrittweise innerhalb einer primären, geschlossenen Regelschleife, die lokal auf jedem Kern ausgeführt und als *Hormonzyklus* (H-Zyklus) bezeichnet wird.

Die *hormongeregelte Taskverarbeitung* (HTV[2]) ist ein hoch dynamisches und zuverlässiges Task-Transfermodell, das eigens im Rahmen dieser Arbeit entwickelt wurde und den Verarbeitungsprozess nach der Zuordnung von Tasks durch das KHS übernimmt. Sie ist daher für die Umsetzung der *Taskverarbeitung* im System verantwortlich und wird im Folgenden auch als solche assoziiert. Gleichermaßen wie beim KHS erfolgt die Verarbeitung dezentral und knüpft nahtlos an dessen Regelkonzept an. Nach der Zuordnung von Tasks durch den H-Zyklus, werden die Tasks gemäß ihrer Verfügbarkeit im Netz zwischen den Kernen über weitere Hormone transferiert. Anschließend erfolgt die lokale Erzeugung und Ausführung der Task-Instanz auf dem zugeordneten Kern. Dabei werden die im Netz verfügbaren Zustandsdaten von Tasks und Kernen berücksichtigt. Dieser Prozess wird durch

[1] Im Englischen wird das KHS als *Artificial Hormone System* kurz AHS bezeichnet.
[2] Im Englischen wird die HTV als *Task Processing System* kurz TPS bezeichnet.

das Handlungsprinzip des H-Zyklus initiiert und synchronisiert. Die Umsetzung erfolgt in einer sekundären, geschlossenen Regelschleife die lokal auf jedem Kern ausgeführt und als *Verarbeitungszyklus* (V-Zyklus) bezeichnet wird.

Um das gemeinsame Funktionsprinzip der beiden Mechanismen KHS und HTV zu erläutern, werden in den folgenden beiden Abschnitten zunächst grundlegende Begrifflichkeiten und Notationsdefinitionen vorgestellt. Die Erläuterung des Funktionsprinzips von KHS und HTV erfolgt dann in Abschnitt 3.1.6.

3.1.1. Begriffsdefinitionen

System, Netz, Grid: Das *System* bezeichnet und umfasst alle implementierten Komponenten (Tasks, Kerne, Verbindungsnetzwerke, Applikationen etc.). Das *Netz* bezeichnet den Zusammenschluss aller Task verarbeitenden Kerne, die über ein Verbindungsnetzwerk miteinander verbunden sind. Das *Grid* bezeichnet eine spezifische Netzart, die eine rechteckige Anordnung von Kernen in horizontale Zeilen und vertikale Spalten definiert. Die beiden Begriffe Netz und Grid sind also gleichbedeutend, wobei letzterer eine Präzisierung darstellt.

Software- u. Hardware-Task: Tasks repräsentieren komplexe (Teil-) Aufgaben die vom System ausgeführt und auf zwei Arten implementiert werden:

- **Software-Task:** Bei einer Software implementierten Task wird die Aufgabe durch ein dynamisches Programm im System abgebildet (lineare Abfolge von Befehlen).
- **Hardware-Task:** Bei einer durch Hardware implementierten Task wird die Aufgabe durch einen statischen Schaltkreis oder mehrere Funktionseinheiten im System abgebildet.

Jede Task des Systems ist bekannt und wird durch mindestens eine Implementierung abgebildet.

Verwandtschaft von Tasks: Tasks, die während ihrer Ausführung gemeinsam an einer Aufgabe arbeiten und miteinander kommunizieren, werden als verwandt bezeichnet. Eine Task kann mit einer oder mehreren Tasks im System verwandt sein. Die Stärke der Verwandtschaft (Grad) richtet sich nach der Menge der Zusammenarbeit und wird unter anderem anhand der untereinander kommunizierten Daten abgeleitet.

Nachbarschaft von Kernen: Kerne, die aufgrund ihrer Position im Grid eine Hop-Distanz von ≤ 1 zueinander haben, werden als benachbart bezeichnet. Bei einem Grid definiert die Nachbarschaft alle Kerne, deren Position beidseitig in horizontaler, vertikaler und diagonaler Richtung mit $\leq -/+1$ entfernt sind. (äußerer Kern-Ring). Es gilt daher, dass auch ein einziger Kern zu sich selbst im Grid benachbart ist.

Komponenten einer Task: Eine Task besteht aus zwei Komponenten: Der Taskbeschreibung und dem optionalen Taskzustand.

- **Beschreibung:** Die Beschreibung übernimmt die Darstellung und Verfügbarkeit einer Task auf allen Systemebenen. Ihr Inhalt ist abhängig von der Art der Task-Implementierung. Bei einer Software-Task umfasst die Beschreibung den kompilierten Zwischen- oder ausführbaren Maschinencode. Bei einer Hardware-Task umfasst die Beschreibung formale Eingabe- oder Konfigurationsparameter, die für die Initialisierung und Steuerung der Prozess-Sequenz im zugehörigen Schaltkreis oder in den Funktionseinheiten benötigt wird (z. B. Datenraten, Timings, indirekte Steuerbefehle etc.). Die Beschreibung einer Task ist zu jedem Systemzeitpunkt global im Netz verfügbar und wird auf mindestens einem Kern gespeichert.
  $\Rightarrow$ Bei einer Software-Task umfasst die Beschreibung die eigentliche Implementierung (Programmcode) und bei einer Hardware-Task formale Parameter die den Hardware-Prozess initialisieren und steuern.

- **Zustand:** Der Zustand repräsentiert den aktuellen Fortschritt der Ausführung einer bestimmten Task auf einem bestimmten Kern im Netz. Der Zustand umfasst neben Prozess-Kontextdaten (Befehlszähler, Statusregister etc.) auch I/O-Daten die von der Task verarbeitet wurden (Zwischenergebnisse, Messergebnisse etc.). Im Startzustand sind keine Zustandsdaten vorhanden, weshalb der initiale Zustand einer Task im Normalfall eine Datenmenge von 0 Bytes aufweist. Im Endzustand ist für jede Task eine unterschiedlich große Zustandsdatenmenge im Netz vorhanden. Diese wird im Normalfall von dem Kern im Netz gespeichert, auf welchem die Task aktuell oder zuletzt ausgeführt wurde.

Bildung von Task-Organen: Tasks die zueinander verwandt sind, werden gemäß dem Grad der Verwandtschaft auf umliegende, benachbarte Kerne zugeordnet. Dadurch entsteht eine Gruppe lokal kooperierender Tasks (Task-Cluster), die als Task-Organ bezeichnet wird. Durch das gruppieren von Tasks in zugehörige Organe wird die Netz-Distanz zwischen den Task-Instanzen minimiert, so dass sich der Kommunikationsaufwand auf einen bestimmten Grid-Bereich begrenzt (Multicast).

Sendearten von Hormonen: Hormone können im Netz auf drei Arten von Kernen und Tasks gesendet werden:

- **Unicast:** Bezeichnet das direkte Senden von Hormonen zwischen einem dedizierten Sender und Empfänger.

- **Multicast:** Bezeichnet das Senden von Hormonen zwischen einem Sender und einer begrenzten Gruppe von Empfängern.

- **Broadcast:** Bezeichnet das Senden von Hormonen an alle Teilnehmer im Netz.

3.1.2. Notationsdefinitionen

In den folgenden Kapiteln werden die hier aufgeführten Definitionen für Indizes, Mengen und Konstanten verwendet. Sie bilden die Grundlage für das Taskverarbeitungssystem.

Ω bezeichnet die Menge aller Kerne im Netz

ω bezeichnet die Anzahl aller Kerne im Netz

$$\omega := |\Omega|$$

NxM bezeichnet die rechteckige Netz-Anordnung (Grid) von ω Kernen in N-Zeilen und M-Spalten

$I\Omega$ bezeichnet die Indexmenge aller Kerne, wobei jeder Index der Menge die Position eines Kerns im Grid definiert

$$I\Omega := \{(1,1), ..., (n_N, m_M)\}$$

C_γ bezeichnet den Kern mit dem Index (Position) $\gamma \in I\Omega$ im Grid

Für die Menge der Kerne Ω gilt somit:

$$\Rightarrow \quad \Omega := \left\{C_{(1,1)}, ..., C_{(n_N,n_M)}\right\} = \left\{ C_\gamma \mid \gamma \in I\Omega \right\}$$

Φ_γ bezeichnet die Menge an Kernen, die mit einer Hop-Distanz von 0 oder 1 zu einem Kern C_γ benachbart sind (also auch C_γ selbst)

$$\Phi_\gamma := \{C_\delta \mid \delta \in I\Omega \text{ und } C_\delta \text{ ist ein Nachbar von } C_\gamma\}$$

φ_γ bezeichnet die Anzahl an Kernen, die zu einem Kern C_γ benachbart sind

$$\varphi_\gamma := |\Omega|$$

M bezeichnet die Menge aller Tasks

m bezeichnet die Anzahl aller Tasks

$$m := |M|$$

IM bezeichnet die Indexmenge aller Tasks, wobei jeder Index der Menge die Ordnungsnummer einer Task definiert

$$IM := \{1, ..., m\}$$

T_i bezeichnet die Task mit dem Index (Ordnungsnummer) $i \in IM$

Für die Menge der Tasks M gilt somit:

$$\Rightarrow \quad M := \{T_1, ..., T_m\} = \{ T_i \mid i \in IM\}$$

$\boldsymbol{M_\gamma}$ bezeichnet die Taskmenge, für die sich ein Kern C_γ interessiert

$$\Rightarrow \quad M_\gamma := \{T_j \mid j \in IM \text{ und Kern } C_\gamma \text{ ist interessiert an } T_j\}$$

$\boldsymbol{m_\gamma}$ bezeichnet die Taskanzahl, für die sich ein Kern C_γ interessiert

$$m_\gamma := |M_\gamma|$$

$\boldsymbol{V_i}$ bezeichnet die Taskmenge, die zu einer Task T_i verwandt ist

$$V_i := \{T_j \mid j \in IM \text{ und } T_j \text{ ist verwandt zu } T_i\}$$

$\boldsymbol{v_i}$ bezeichnet die Taskanzahl, die zu einer Task T_i verwandt ist

$$v_i := |V_i|$$

$\boldsymbol{E_\gamma}$ bezeichnet die Taskmenge, die auf Kern C_γ ausgeführt wird

$$E_\gamma := \{T_j \mid j \in IM \text{ und } T_j \text{ wird auf } C_\gamma \text{ ausgeführt}\}$$

$\boldsymbol{e_\gamma}$ bezeichnet die Taskanzahl, die auf Kern C_γ ausgeführt wird

$$e_\gamma := |E_\gamma|$$

$\boldsymbol{O_\gamma}$ bezeichnet die Taskmenge, die auf Kern C_γ optimiert wird

$$O_\gamma := \{T_j \mid j \in IM \text{ und } T_j \text{ wird auf } C_\gamma \text{ optimiert}\}$$

$\boldsymbol{o_\gamma}$ bezeichnet die Taskanzahl, die auf Kern C_γ optimiert wird

$$o_\gamma := |O_\gamma|$$

$\boldsymbol{L_\gamma}$ bezeichnet die Taskmenge, die initial auf Kern C_γ gespeichert wird

$$L_\gamma := \{T_j \mid j \in IM \text{ und } T_j \text{ wird initial auf } C_\gamma \text{ gespeichert}\}$$

$\boldsymbol{l_\gamma}$ bezeichnet die Taskanzahl, die initial auf Kern C_γ gespeichert wird

$$l_\gamma := |L_\gamma|$$

$\boldsymbol{R_\gamma}$ bezeichnet die Taskmenge, die aufgrund der redundanten Task-Sicherung zusätzlich auf Kern C_γ gespeichert wird

$$R_\gamma := \{T_j \mid j \in IM \text{ und } T_j \text{ wird redundant auf } C_\gamma \text{ gespeichert}\}$$

$\boldsymbol{r_\gamma}$ bezeichnet die Taskanzahl, die aufgrund der redundanten Task-Sicherung zusätzlich auf Kern C_γ gespeichert wird

$$r_\gamma := |R_\gamma|$$

3.1.3. Definitionen zu Hormonen

Im Allgemeinen werden Hormone innerhalb des Systems dazu verwendet, Informationen und Daten jeglicher Art zwischen Tasks und Kernen zu kommunizieren. Daher können Hormone abstrakt als Nachrichten aufgefasst werden, die eine bestimmte Informations- oder Datenmenge des Absenders (Kern, Task) aufnehmen und an einen oder mehrere Empfänger im Netz übermitteln. Für das KHS und die HTV werden zwei Klassen von Hormonen unterschieden, die im Folgenden detailliert vorgestellt werden.

3.1.4. Steuerhormone des KHS

Steuerhormone werden für die Umsetzung der Taskzuordnung durch das KHS gesendet und beinhalten neben einer Absenderkennung (Header), einen Hormontyp und den eigentlichen Hormonwert (Ganzzahl ≥ 0). Im KHS werden drei Typen von Steuerhormonen unterschieden:

Eignungswerte: Dieser Hormontyp repräsentiert die Eignung von Kernen gegenüber der Ausführung von Tasks. Ein hoher Eignungswert bedeutet, dass ein bestimmter Kern zur Ausführung einer bestimmten Task besser geeignet ist, als ein Kern, der für die gleiche Task einen niedrigeren Wert aufweist. Innerhalb der Taskzuordnung werden zwei konkrete Eignungswerte unterschieden:

- **Lokale Eignungswert - $E_{i\gamma}$:** Dieses Hormon repräsentiert die Eignung eines Kerns C_γ eine bestimmte Task T_i im Netz ausführen zu können. Der Wert dieses Hormons wird statisch aufgrund der funktionalen Eigenschaften von C_γ bestimmt und lokal gespeichert. Falls C_γ keine Eignung gegenüber T_i besitzt, ist der Wert Null ($E_{i\gamma} = 0$).

- **Modifizierte Eignungswert - $Em_{i\gamma}^{i\Omega}$:** Dieses Hormon repräsentiert die aktuelle Eignung von C_γ zur Ausführung von Task T_i. Der Hormonwert wird durch Verrechnung des lokalen Eignungswerts $E_{i\gamma}$ mit allen auf C_γ empfangen Hormonen bestimmt, wobei Akzeleratoren addiert und Suppressoren subtrahiert werden. Dieses Hormon wird global an die Task T_i auf allen Kernen im Netz gesendet.

Suppressoren: Dieser Hormontyp verringert die Eignung von Kernen hinsichtlich der Ausführung von Tasks und übernimmt die negative Rückkopplung im H-Zyklus. Dieser Typ wird maßgeblich dazu verwendet, die lokale Übernahme einer Task global im Netz zu kommunizieren, um dadurch weitere Übernahmen zu verhindern oder zu begrenzen. Des Weiteren werden durch die Ausschüttung lokaler Suppressoren negative Zustandsänderungen des Systems (z. B. eine hohe Kern-Auslastung oder

-Temperatur) erfasst und bei der Taskverteilung mit einbezogen. Innerhalb des KHS werden drei Suppressoren unterschieden:

- **Übernahme-Suppressor -** $\boldsymbol{S_{i\gamma}^{i\Omega}}$: Dieses Hormon wird global von Kern C_γ an die Task T_i auf allen Kernen im Netz gesendet, wenn T_i von Kern C_γ übernommen wurde. Dadurch wird die Eignung aller Kerne gegenüber T_i gemäß der Stärke des Hormonwerts vollständig oder nur teilweise unterdrückt, wodurch eine einfache oder mehrfache Übernahme von T_i im Netz erfolgt.

- **Last-Suppressor -** $\boldsymbol{Sl_{i\gamma}^{M\gamma}}$: Dieses Hormon wird lokal an alle Tasks $T_j \in M_\gamma$ von Kern C_γ gesendet, solange T_i auf Kern C_γ ausgeführt wird. Der Hormonwert repräsentiert die Kernauslastung, die im Zuge der Ausführung von T_i auf C_γ verursacht wird. Dadurch wird die Anzahl der auf C_γ ausgeführten Tasks e_γ begrenzt.

- **Monitoring-Suppressor -** $\boldsymbol{Sm^{M\gamma}}$: Dieses Hormon wird von den lokalen Monitoring-Einheiten an alle Tasks $T_j \in M_\gamma$ auf Kern C_γ gesendet. Durch den Hormonwert wird der aktuelle Zustand von Kern C_γ (Verlauf der Kerntemperatur, Taskfehler etc.) bei Zuordnung von Tasks berücksichtigt. Ähnlich wie der Last-Suppressor verringert dieses Hormon die Eignung von C_γ, je schlechter das Monitoring dessen Zustand bewertet.

Akzeleratoren: Dieser Hormontyp verbessert die Eignung von Kernen hinsichtlich der Ausführung von Tasks und übernimmt die positive Rückkopplung im H-Zyklus. Dieser Typ wird maßgeblich dazu verwendet, um eine Zuordnung von Tasks in Organe zu ermöglichen und um die aktuelle Task-Zuordnung im Netz zu festigen. Des Weiteren werden durch die Ausschüttung lokaler Akzeleratoren positive Zustandsänderungen des Systems (z. B. eine geringe Auslastung etc.) erfasst und bei der Taskverteilung mit einbezogen. Innerhalb des KHS werden drei Suppressoren unterschieden:

- **Angebots-Akzelerator -** $\boldsymbol{A_{i\gamma}^{i\gamma}}$: Dieses Hormon wird lokal von Task T_i an sich selbst gesendet, wenn T_i im aktuellen H-Zyklus von C_γ im Netz neu angeboten wird. Da im Falle einer Neuzuordnung die Optimierung von T_i zusätzliche Transferkosten aufwirft, stärkt dieses Hormon die aktuelle Zuordnung von T_i auf C_γ. Dadurch wird sichergestellt, dass die Optimierung von T_i nur dann erfolgt, wenn im Netz eine bessere Eignung verfügbar ist, die oberhalb des Schwellwerts von $A_{i\gamma}^{i\gamma}$ liegt. Somit repräsentiert der eigentliche Hormonwert die Kosten die im Zuge einer Optimierung von T_i anfallen.

- **Organ-Akzelerator -** $\boldsymbol{A_{i\gamma}^{V_i\Phi_\gamma}}$: Dieses Hormon wird von T_i an alle verwandten Tasks $T_j \in V_i$ auf allen benachbarten Kernen $C_\delta \in \Phi_\gamma$ gesendet, wenn T_i von Kern C_γ übernommen wurde. Dadurch erfolgt die Gruppierung aller zu

T_i verwandten Tasks in ein lokales Organ, wodurch der Kommunikationsaufwand zwischen den betroffenen Tasks global minimiert wird. Der Hormonwert repräsentiert den Grad der Verwandtschaft und wird lokal auf den Kernen gespeichert.

- **Monitoring-Akzelerator - $\boldsymbol{A_{i\gamma}^{M\gamma}}$**: Dieses Hormon wird von den lokalen Monitoring-Einheiten an alle Tasks $T_j \in M_\gamma$ auf Kern C_γ gesendet und stellt das Gegenstück zum Monitoring-Suppressor dar. Der Hormonwert verbessert daher die Eignung aller Tasks auf Kern C_γ, je besser das Monitoring den Zustand bewertet. Die Verwendung dieses Hormons ist aber optional, da eine Verbesserung wechselseitig durch den Rückgang des Monitor-Suppressors modelliert werden kann.

3.1.5. Datenhormone der HTV

Innerhalb der HTV erfolgt der Transfer von Daten ausschließlich durch das Senden von Datenhormonen. Dadurch werden Tasks und deren Kommunikation abstrakt als Hormone aufgefasst, die gemäß der aktuellen Taskzuordnung zwischen verschieden Kernen und Tasks im Netz gesendet werden. Ein Datenhormon beinhaltet neben einer Absenderkennung den Hormontyp und die eigentlichen Nutzdaten. Die Menge an Nutzdaten ist begrenzt, so dass für größere Datenmengen eine Aufteilung in mehrere Datenhormone erfolgt, die sequentiell gesendet werden. Die Anzahl der benötigten Hormone ist abhängig von der Datenmenge sowie der Nutzlast eines einzelnen Datenhormons. Innerhalb der HTV werden drei Typen von Datenhormonen unterschieden:

- **Beschreibungs-Hormon - $\boldsymbol{B_{i\delta}^{i\gamma}}$:** Dieses Hormon wird für den sequentiellen Transfer der Beschreibung einer Task T_i gesendet, wenn T_i von Kern C_γ übernommen wurde. Kern C_δ speichert die Beschreibung von T_i und sendet die zugehörige Datenmenge mittels einem oder mehreren Beschreibungs-Hormonen an Task T_i auf Kern C_γ.

- **Zustands-Hormon - $\boldsymbol{Z_{i\delta}^{i\gamma}}$:** Dieses Hormon wird für den Transfer des aktuellen Zustands einer Task T_i gesendet, wenn T_i von Kern C_γ übernommen wurde. Kern C_δ speichert den Zustand von T_i und sendet die zugehörige Datenmenge mittels eines oder mehrerer Zustands-Hormone an Task T_i auf Kern C_γ.

- **Kommunikations-Hormon - $\boldsymbol{K_{i\gamma}^{V_i\Phi_\gamma}}$:** Dieses Hormon wird für die Kommunikation zwischen einer Task T_i mit allen verwandten Tasks $T_j \in V_i$ auf allen benachbarten Kernen $C_\delta \in \Phi_\gamma$ gesendet, wenn T_i auf Kern C_γ ausgeführt wird. Die Menge der gesendeten Kommunikations-Hormone ist abhängig von der Kommunikationsrate von Task T_i.

3.1.6. Organisches Funktionsprinzip

Im Folgenden wird das organische Funktionsprinzip des Systems durch das Zusammenspiel von Hormonzyklus (H-Zyklus) und Verarbeitungszyklus (V-Zyklus) vorgestellt. Abbildung 3.1 zeigt eine vereinfachte Darstellung beider Zyklen, die in zugehörigen Instanzen auf allen Kernen lokal ausgeführt werden. Weitere Details bezüglich der Umsetzung sowie des Zeitverhaltens beider Zyklen werden in den nachfolgenden Kapiteln 4 und 5 erläutert. Die Dauer des H-Zyklus ist statisch und die Auswertung erfolgt periodisch. Hingegen ist die Dauer des V-Zyklus dynamisch und die Auswertung einer Iteration erfolgt in Synchronisation mit dem H-Zyklus sobald die Zuordnung einer Task erfolgt ist.
⇒ Nach der (Neu-)Zuordnung einer Task T_i durch den periodischen H-Zyklus erfolgt der (Neu-)Start des zugehörigen V-Zyklus von T_i im System.

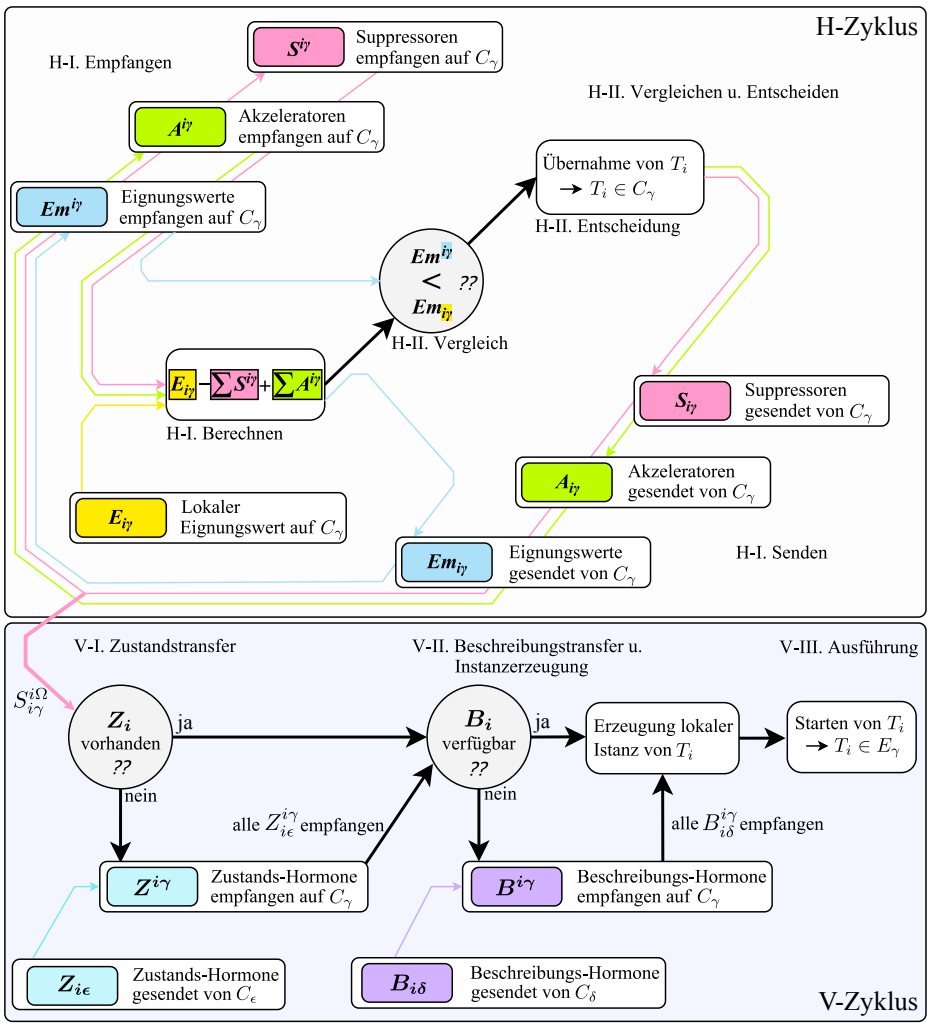


Abbildung 3.1.: H-Zyklus und V-Zyklus des Systems

Die Funktionsweise des H-Zyklus gliedert sich in zwei Phasen:

Der H-Zyklus startet in der ersten Phase (Abbildung 3.2) mit dem nebenläufigen Senden und Empfangen von Steuerhormonen. Diese werden für die Berechnung der modifizierten Eignungswerte aller Tasks $T_i \in M_\gamma$ benötigt, auf die sich ein Kern C_γ bewirbt. Dabei werden für jede Task T_i die empfangenen Suppressoren $S^{i\gamma}$ und Akzeleratoren $A^{i\gamma}$ mit dem lokalen Eignungswert $E_{i\gamma}$ verrechnet, wobei Suppressoren subtrahiert und Akzeleratoren addiert werden. Anschließend wird der berechnete, modifizierte Eignungswert $Em_{i\gamma}$ für Task T_i per Broadcast an alle KHS-Instanzen auf allen Kernen im Netz gesendet, sofern der Hormonwert $Em_{i\gamma}$ größer Null ist. Diese Phase endet nachdem alle fremden Hormone empfangen und alle modifizierten Eignungswerte berechnet und gesendet wurden.

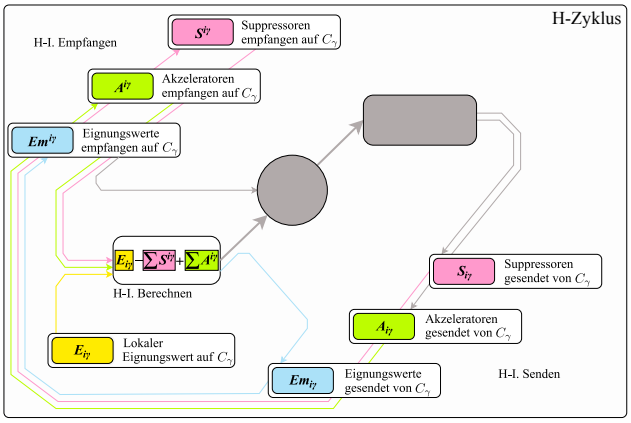


Abbildung 3.2.: 1. Phase H-Zyklus: Senden u. Empfangen

In der zweiten Phase (Abbildung 3.3) erfolgt die Entscheidung über die lokale Zuordnung von Tasks. Dabei werden für eine ausgewählte Task $T_i \in M_\gamma$ alle empfangenen modifizierten Eignungswerte $E^{i\gamma}$ mit dem gesendeten Eignungswert verglichen. Ist der eigene, gesendete Wert $Em_{i\gamma}$ größer als der höchstwertige, empfangene Eignungswert $Em^{i\gamma}$, dann ist Kern C_γ für Taskausführung am besten geeignet und übernimmt folglich die Task T_i. Dafür sendet Kern C_γ in der ersten Phase des darauffolgenden H-Zyklus einen globalen Übernahme-Suppressor aus, um die Übernahme auf allen Kernen im Netz mitzuteilen. Des Weiteren sendet C_γ durch die Übernahme zusätzliche Organ-Akzeleratoren für T_i aus, um die verwandten Tasks auf den benachbarten Kernen innerhalb eines Organs zu allokieren (Task-Clustering). Diese Phase endet

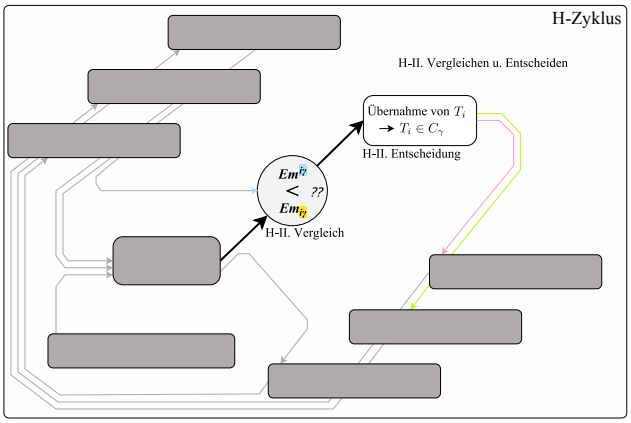


Abbildung 3.3.: 2. Phase H-Zyklus: Vergleichen u. Entscheiden

nachdem die Entscheidung über Taskzuordnung von T_i gefällt ist. Das Senden des Übernahme-Suppressors sowie der Organ-Akzeleratoren für T_i erfolgt in der ersten Phase des darauffolgenden H-Zyklus.

Die Funktionsweise des V-Zyklus gliedert sich in drei Phasen, die je nach Verfügbarkeit von Taskkomponenten auch wahlweise in der Sequenz übersprungen werden können.

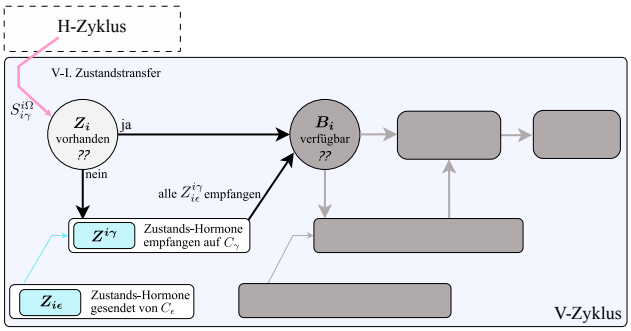


Abbildung 3.4.: 1. Phase V-Zyklus: Zustandstranfer

Der V-Zyklus für eine Task T_i startet in der ersten Phase (Abbildung 3.4) mit dem Transfer des Taskzustands von T_i, nachdem der zugehörige Übernahme-Suppressor $S_{i\gamma}$ durch Kern C_γ im Netz vollständig kommuniziert wurde. Ist zum Zeitpunkt der Übernahme kein Zustand von Task T_i im Netz vorhanden (bei Systemstart) oder wird der Zustand bereits auf Kern C_γ gespeichert, so wird diese Phase übersprungen. Andernfalls beginnt Kern C_ϵ, welcher den aktuellen Zustand von T_i speichert, nach dem Empfang des Übernahme-Suppressors mit dem Senden der Zustands behafteten Datenhormone $Z_{i\epsilon}^{i\gamma}$ an Kern C_γ. Befindet sich die Task T_i zum Zeitpunkt des Suppressor-Empfangs aufgrund einer vorhergehenden Taskzuordnung in der Ausführung auf Kern C_ϵ, so wird diese vor dem Zustandstransfer abgebrochen. Diese Phase endet, nachdem der Zustand von T_i vollständig an Kern C_γ transferiert wurde.

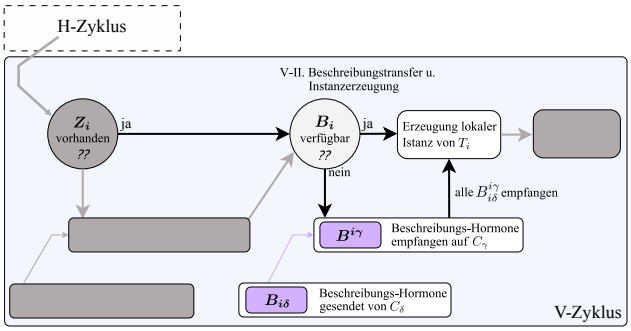


Abbildung 3.5.: 2. Phase V-Zyklus: Instanzerzeugung

In der zweiten Phase (Abbildung 3.5) erfolgt der Transfer der Taskbeschreibung sowie die eigentliche Erzeugung der lokalen Taskinstanz von T_i auf Kern C_γ. Wird die die Beschreibung bereits auf Kern C_γ gespeichert, wird der Transfer übersprungen und direkt mit der Instanzerzeugung begonnen. Andernfalls sendet Kern C_δ, welcher die Beschreibung von T_i initial im Netz speichert, die zugehörigen Datenhormone $B_{i\gamma}^{i\gamma}$ an Kern C_γ. Nachdem der Transfer beendet ist,

erfolgt unter Berücksichtigung der optional vorhandenen Zustandsdaten aus der vorangegangenen ersten Phase die Erzeugung der lokalen Instanz von T_i auf C_γ. Diese Phase endet sobald die Instanzerzeugung abgeschlossen ist.

In der letzten Phase (Abbildung 3.6) erfolgt das Starten der Taskausführung von T_i auf Kern C_γ. Durch den lokalen Scheduler erfolgt die Zuordnung des eigentlichen Task-Prozess von T_i auf die Recheneinheiten von Kern C_γ. Nachdem die Task gestartet wurde, verweilt der Zyklus weiterhin in dieser Phase. Dies gilt solange bis eine Neuzuordnung von T_i durch den H-Zyklus erfolgt, was den Start des Folgezyklus initiiert (Rückkopplung auf die erste Phase des V-Zyklus).

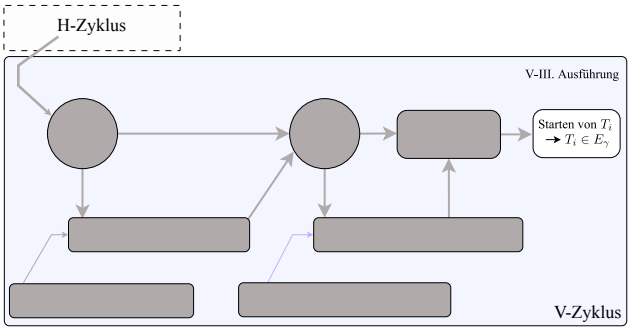


Abbildung 3.6.: 3. Phase V-Zyklus: Ausführung von Task T_i

4. System Eigenschaften

4.1. Eigenschaften der hormongeregelten Taskverarbeitung

Das KHS übernimmt die Aufgabe der dezentralen Zuordnung von Tasks gegenüber den Kernen. Die Funktionsweise dieser Taskverwaltung wurde in Kapitel 3.1.6 dargelegt. Die formalen Beweise und Analysen bezüglich der Korrektheit für die Selbst-Konfiguration, -Heilung und -Optimierung wurden in Forschungsarbeit [vR12] gezeigt. Hierbei wurde die eigentliche Verarbeitung von Tasks ausgegliedert, da diese nicht integrativer Bestandteil und Aufgabe der Taskverwaltung ist. Im folgenden Abschnitt wird nun das formale Modell der hormongeregelten Taskverarbeitung ergänzt (HTV). Dieses knüpft nahtlos an die Taskverwaltung des KHS an und behandelt die Definition von zeitlichen Abläufen, Zuständen und Ereignissen die nach der erfolgreichen Übernahme von Tasks durch das KHS erfolgen. Für die Koordination der verteilten Taskverarbeitung und die Synchronisation gegenüber der Taskzuordnung, werden ausschließlich die durch das KHS propagierten Hormone genutzt. Das Modell verzichtet somit vollständig auf die aktive Verwendung eigener Kontroll- und Synchronisationsmechanismen um negative Auswirkungen gegenüber der Taskverwaltung auszuschließen. Das Gesamt-Zeitverhalten des Modells wird in Abschnitt 4.2 analysiert.

4.1.1. Ergänzende Notationsdefinition

In den folgenden Abschnitten und Kapiteln werden die hier aufgeführten Definitionen für Indizes, Mengen und Konstanten verwendet. Diese bilden die Grundlage für das dezentrale Taskverarbeitungskonzept.

$\boldsymbol{I_{i\gamma}}$	bezeichnet die lokale Instanz von Task T_i auf Kern γ
$\boldsymbol{Cz_i}$	bezeichnet den Kern, welcher den aktuellen Zustand von Task T_i speichert
$\boldsymbol{Cb_i}$	bezeichnet den Kern, welcher die Beschreibung von Task T_i speichert
$\boldsymbol{ts_{max}}$	ist das größtmögliche Intervall für die Bedienzeit einer Task im Netz (maximale Prozessorzeitscheibe)
$\boldsymbol{m_{max}}$	ist die größtmögliche Anzahl an Tasks auf die sich ein Kern bewirbt $:= \max_{C_\gamma \in \Omega}\{m_\gamma\}$
$\boldsymbol{Dh}$	bezeichnet die maximale Nutzdatenmenge eines Daten-Hormons im Netz (mit $Dh > 0$ Byte)
$\boldsymbol{D_{hv}}$	bezeichnet die maximale Datenmenge des Hormonwerts von Steuerhormonen im Netz (mit $D_{hv} \geq 1$ Byte)
$\boldsymbol{D_{hs}}$	bezeichnet die maximale Datenmenge einer Hormonwert-Summe aus Steuerhormonen im Netz (mit $D_{hs} \geq 2$ Byte)
$\boldsymbol{Dz_i}$	bezeichnet die Datenmenge des Zustands von Task T_i (mit $Dz_i \geq 0$ Byte)
$\boldsymbol{Db_i}$	bezeichnet die Datenmenge der Beschreibung von T_i (mit $Db_i > 0$ Byte)
$\boldsymbol{Dz_{max}}$	ist die maximale Datenmenge eines Taskzustand im Netz (mit $Dz_{max} \geq 0$ Byte) $:= \max_{T_i \in M}\{Dz_i\}$
$\boldsymbol{Db_{max}}$	ist die maximale Datenmenge einer Taskbeschreibung im Netz (mit $Db_{max} > 0$ Byte) $:= \max_{T_i \in M}\{Db_i\}$

4.1.2. Definition von Vorgängen und Zeiten innerhalb der Taskverarbeitung

Das Modell greift das Verhalten von Tasks unmittelbar nach der Zuordnung durch das KHS auf. Ausgehend vom Zeitpunkt der Entscheidung über die Taskübernahme im jeweiligen Hormonzyklus, ist eine unterschiedliche Abfolge der folgenden Schritte zu berücksichtigen, bevor die eigentliche Taskausführung durch den Kern gestartet werden kann. Die Definition und Betrachtung unterschiedlicher Szenarien von Taskübernahmen erfolgt in Abschnitt 4.1.3.

Zustandstransfer - ($\dot{\mathbf{Z}}$) Dieser Schritt umfasst den Transfer des Zustandes der übernommenen Task zum Kern. Der Transfer erfolgt hormongestützt und endet sobald die zugehörige Datenmenge Dz_i vollständig übertragen wurde.

Instanzerzeugung - ($\dot{\mathbf{E}}$) Bei der Erzeugung wird die lokale Instanz der übernommenen Task anhand der Taskbeschreibung generiert. Sofern zuvor ein Zustand übertragen wurde, wird dieser von der Instanzerzeugung berücksichtigt und konsistent wiederhergestellt.

Ausführung - ($\dot{\mathbf{A}}$) In diesem Schritt erfolgt die Ausführung der Task aus Sichtweise der HTV. Abhängig vom Kontrollfluss der Task befindet sich der Prozess entweder im aktiven, bereiten oder blockierten Zustand.

Instanztermination - ($\dot{\mathbf{E}}^{-1}$) Die Termination umfasst das endgültige Löschen einer Taskinstanz im Falle einer Übernahme durch einen anderen Kern nachdem die Ausführung abgebrochen wurde. Sie bildet somit die Umkehroperation zur Instanzerzeugung, da Sie nur bei Abgabe einer Task an einen anderen Kern auftritt und sich mit dieser zur Laufzeit vollständig überdeckt oder nebenläufig erfolgt. Sie stellt daher keinen echten Vorgang der Taskverarbeitung dar und gilt nur als Synonym für die Instanzerzeugung aus Sichtweise des Kerns welcher die Task abgibt.

Zusätzlich zu den oben genannten Schritten existieren innerhalb der Taskverarbeitung noch die folgenden, sogenannten Wartezeiten, die als Übergang zwischen bestimmten Schritten auftreten können.

Start der Taskausführung - ($t_{\dot{E}\dot{A}}$) Dieser Schritt beinhaltet den erstmaligen Startvorgang der Taskausführung nach Übernahme. Es erfolgt die Eingliederung der Task in die vorhandene Verarbeitungssequenz des jeweiligen Kerns. Dieser Schritt endet, sobald der zugehörige Prozess der Task zugeteilt wird und dieser in den aktiven Zustand wechselt.

Abbruch der Taskausführung - ($t_{\dot{S}}$) Hier wird die Task in ihrer Ausführung unterbrochen. Die aktuelle Prozessausführung wird gestoppt und die Task wechselt in den inaktiven Zustand.

4.1.2.1. Details zur Komplexität von Schritten und Wartezeiten

Es folgt die Analyse von Worst-Case Laufzeiten der im vorangegangenen Abschnitt definierten Abläufe.

Der Overhead für den Zustandstransfer in $\dot{\text{Z}}$ einer Task ist proportional zum Aufwand, der für die Übertragung der zugehörigen Datenmenge Dz des Zustands benötigt wird. Unter der Berücksichtigung der maximalen Hormongröße Dh kann die Laufzeit daher als vielfaches der maximalen Kommunikationszeit t_K angenommen werden.

$$\text{Worst-Case Zeitverhalten} \quad t_{\dot{Z}} = \left\lceil \frac{Dz_{max}}{Dh} \right\rceil \cdot t_K \tag{4.1}$$

Das Laufzeitverhalten für die Instanzerzeugung in $\dot{\text{E}}$ einer Task wird dominiert von der Zeitdauer des Transfers der Taskbeschreibung. Diese umfasst, neben der eigentlichen Beschreibung, die Adressierung aller kontextbezogenen Daten- und IO-Ströme. Der Aufwand definiert sich über die Gesamtdatenmenge Db der Taskbeschreibung. Die Laufzeit berechnet sich somit äquivalent zum Zustandstransfer als das vielfache der Kommunikationszeit tk bis alle Hormone vollständig übertragen wurden.

$$\text{Worst-Case Zeitverhalten} \quad t_{\dot{E}} = \left\lceil \frac{Db_{max}}{Dh} \right\rceil \cdot t_K \tag{4.2}$$

Die Wartezeit $t_{\dot{E}\dot{A}}$ für den Start der Taskausführung umfasst die Zeitspanne, ausgehend vom Zeitpunkt nach der Instanzerzeugung hinweg, über die Eingliederung des zugehörigen Task-Prozess in die Warteschlange des Scheduler, bis hin zur (Wieder-) Aufnahme der Ausführung am zuletzt betrachteten Haltepunkt. Hier dominiert die maximale Zuordnungszeit ts_{max} (Zeitscheibenintervall), die ein Prozess vom Scheduler des Kerns erhalten kann. Unter der Berücksichtigung einer fairen und prioritätsgeführten Prozessausführung sowie der maximalen Taskanzahl m_{max} für die sich ein Kern interessiert, ist im Worst Case Fall mit einer vorangehenden Sequenz von $m_{max} - 1$ Zuteilungen zu rechnen, bevor der Prozess zum ersten mal in den aktiven Zustand wechselt.

$$\text{Worst-Case Zeitverhalten} \quad t_{\dot{E}\dot{A}} = ts_{max} \cdot (m_{max} - 1) \tag{4.3}$$

Die Wartezeit $t_{\dot{S}}$ für den Abbruch einer Taskausführung erfolgt entweder im Falle einer selbstbestimmten Terminierung oder bei einer erneuten Zuordnung der Task durch das KHS. Anders als beim Startvorgang befindet sich in beiden Fällen der zugehörige Task-Prozess entweder im aktiven oder blockierten Zustand. Somit muss im

schlimmsten Fall eine maximale Zuordnungszeit ts_{max} abgewartet werden, falls der Prozess aufgrund einer kritischen Tasksequenz nicht vorzeitig unterbrochen werden kann.

$$\text{Worst-Case Zeitverhalten} \quad t_{\dot{S}} = ts_{max} \tag{4.4}$$

4.1.3. Kontrolle der dynamischen Aspekte bei der Taskverarbeitung

Abbildung 4.1 zeigt den Zyklus der hormongesteuerten Taskverarbeitung (HTV) einer Task aus Sichtweise des Kerns, dessen Hormonzyklus (H-Zyklus) die Entscheidung über die Taskübernahme getroffen hat. (V-Zyklus).

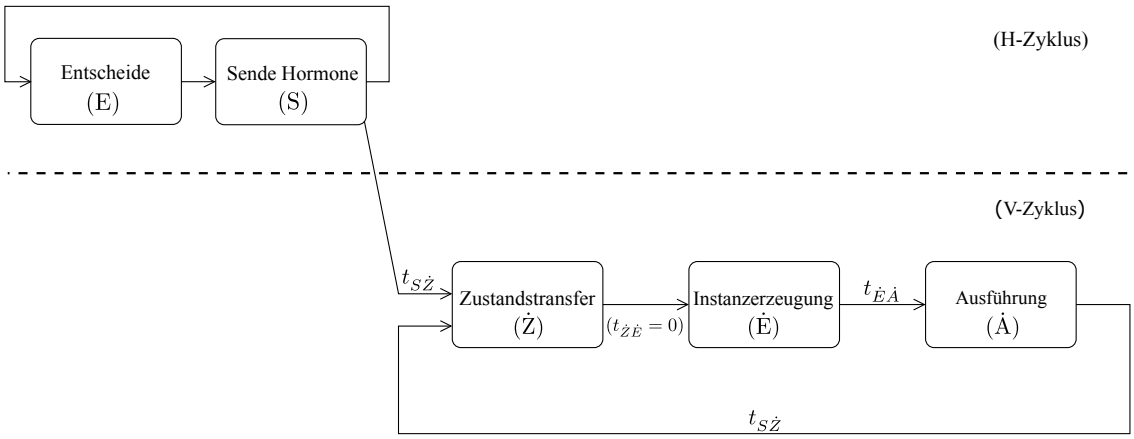


Abbildung 4.1.: Verarbeitungszyklus (V-Zyklus)

Wie bereits in Abschnitt 4.1 angekündigt, knüpft die Sequenz der Taskverarbeitung direkt an den Entscheidungsschritt des H-Zyklus an. Nach dem Entscheiden wechselt der H-Zyklus in den Sendeschritt (S). Im Falle der Taskübernahme werden in diesem Übernahme-Suppressoren ausgesendet. Dieses Ereignis definiert den Übergang und die Schnittstelle zum Task-Verarbeitungszyklus Die Sequenz des V-Zyklus sieht wie folgt aus:

1. $t_{S\dot{Z}}$ - Reaktionszeit auf Taskübernahme
2. $\dot{Z}$ - Zustandstransfer
3. $\dot{E}$ - Instanzerzeugung
4. $t_{\dot{E}\dot{A}}$ - Warten bis Ausführung gestartet wurde
5. $\dot{A}$ - Ausführung
6. $t_{S\dot{Z}}$ - Warten bis Ausführung beendet wurde

Der V-Zyklus einer Task beginnt, nachdem sich die Information der Taskübernahme im Netz verteilen konnte und Kern Cz_i, welcher den aktuellen Zustand der Task speichert, informiert wird. Diese Wartezeit $t_{S\dot{Z}}$ in 1., wird als *Reaktionszeit auf Taskübernahme* des Zyklus bezeichnet. Nach Ablauf von $t_{S\dot{Z}}$ startet der Zyklus mit dem Zustandstransfer ($\dot{\text{Z}}$) durch das Aussenden der zustandsbehafteten Datenhormone von Cz_i. Nach dem Zustandstransfer beginnt in 3. die Erzeugung der lokalen Taskinstanz ($\dot{\text{E}}$), die durch das Aussenden der Hormone von Kern Cb_i, welcher die Taskbeschreibung speichert, eingeleitet wird. Nachdem die Erzeugung abgeschlossen ist, wartet der Zyklus in 4. auf den Start der Taskausführung. Diese Wartezeit $t_{\dot{E}\dot{A}}$ umfasst die Zeitspanne nach der Erzeugung der lokalen Taskinstanz bis zum Start der Taskausführung. Nach Ablauf der Wartezeit befindet sich die Taskverarbeitung dann im Schritt 5. in der Ausführung ($\dot{\text{A}}$). Hier angekommen verweilt der Zyklus solange in diesem Schritt, bis die Ausführung durch eine Neuzuordnung des H-Zyklus oder durch die Task selbst abgebrochen werden muss. Die Wartezeit in 6. entspricht erneut der in 1. aufgeführten Reaktionszeit $t_{S\dot{Z}}$ und umfasst den Zeitraum bis der V-Zyklus erneut gestartet werden kann, was beispielsweise im Falle einer Neuzuordnung der Task durch den H-Zyklus erfolgt.

Abgeleitet von der obigen Sequenz erschließt sich die Definition der folgenden Zeiten, die bei der Verarbeitung des V-Zyklus auftreten:

Reaktionszeit auf Taskübernahme: Die Reaktionszeit $t_{S\dot{Z}}$ in 1. und 6. definiert die Wartezeit bis zum Start oder Neustart des V-Zyklus. Die zugehörige Zeitspanne umfasst daher den Sequenz-Abschnitt ausgehend vom Broadcast des Übernahme-Suppressor (S) bis zum Start oder Neustart des Zustandstransfer ($\dot{\text{Z}}$) in 2. Im Falle eines Neustart des V-Zyklus in 6., erfolgt innerhalb der Zeitspanne noch zusätzlich ein Abbruch der Taskausführung.

Vorlaufzeit auf Taskausführung: Die Vorlaufzeit $t_{\dot{Z}\dot{A}}$ definiert die Verweilzeit des V-Zyklus bis zum Erreichen der Taskausführung. Die Zeitspanne $t_{\dot{Z}\dot{A}}$ definiert den Sequenz-Abschnitt ausgehend vom Beginn des Zustandstransfer ($\dot{\text{Z}}$) in 2. bis zum Beginn der Taskausführung ($\dot{\text{A}}$) in 5.

$$t_{\dot{Z}\dot{A}} = t_{\dot{Z}} + t_{\dot{E}} + t_{\dot{E}\dot{A}} \tag{4.5}$$

V-Zykluszeit: Aufgrund der KHS oder der Task selbstbestimmten Zeitdauer $t_{\dot{A}}$, die der Zyklus in der Ausführung verweilt, umfasst die Zykluszeit im Gegensatz zu der des H-Zyklus nicht den Ablauf der vollständigen Sequenz 1. - 6., sondern die Zeitspanne ausgehend vom Senden (S) der Taskübernahme in 1. bis zum Erreichen der Taskausführung ($\dot{\text{A}}$) in 5. Die V-Zykluszeit ergibt sich daher als Akkumulation

aus Reaktions- und Vorlaufzeit.

$$t_{VC} = t_{S\dot{A}} = t_{S\dot{Z}} + t_{\dot{Z}\dot{A}} \tag{4.6}$$

Die Zykluszeit der Taskverarbeitung ist somit dynamisch und definiert sich unabhängig vom System und dessen Zeitbedingung weitestgehend anhand der taskgebundenen Komplexität der Schritte 1. - 5., die über die Reaktions- und Vorlaufzeit erfasst werden. Dennoch sollte für eine möglichst reaktionsfähige Verarbeitung von Tasks gegenüber der Taskverwaltung versucht werden, die beiden Zeiten nach Möglichkeit zu minimieren. Für die weitere Analyse, ob und unter welchen Handlungsvorschriften innerhalb des V-Zyklus diese Minimierung eingehalten und die Taskverarbeitung stets konsistent zur aktuellen Zuordnung einer Task T_i nur anhand der ausgetauschten Hormone erfolgen kann, werden die genannten Zeiten $t_{S\dot{Z}}$ und $t_{\dot{Z}\dot{A}}$ des V-Zyklus nun genauer untersucht.

4.1.3.1. Betrachtung und Analyse der Reaktionszeit

Die Länge der Reaktionszeit ist fallabhängig und unterscheidet sich zu welchem Zeitpunkt der Taskverarbeitung die Taskübernahme, die den V-Zyklus initiiert, erfolgt. Unabhängig in welcher Selbst-X-Phase der zugehörige H-Zyklus ausgeführt wird, so sind immer nur die folgenden beiden Fälle der Taskübernahme zu unterscheiden:

1. Fall: Inaktive Taskübernahme Die inaktive Übernahme definiert die Zuordnung der Task T_i, während die zugehörige Taskverarbeitung *inaktiv* ist. Zum Zeitpunkt der Übernahme wird der V-Zyklus also nicht ausgeführt und T_i befindet sich auch in keiner weiteren Instanz in der Verarbeitung. Ein Abbruch der Taskausführung ist daher nicht erforderlich und der V-Zyklus ($j = 1$) startet direkt mit dem Zustandstransfer ($\dot{\text{Z}}$). Dieser umfasst in diesem Fall entweder den initialen oder den zuletzt gespeicherten Zustand von T_i. Eine inaktive Übernahme tritt nur bei initialer oder erneuter Zuordnung von T_i durch das KHS in den Phasen der Selbst-Konfiguration oder Selbst-Heilung auf. Abbildung 4.2 stellt den zeitlichen Ablauf einer initialen Taskübernahme dar. C_γ übernimmt T_i während der V-Zyklus noch

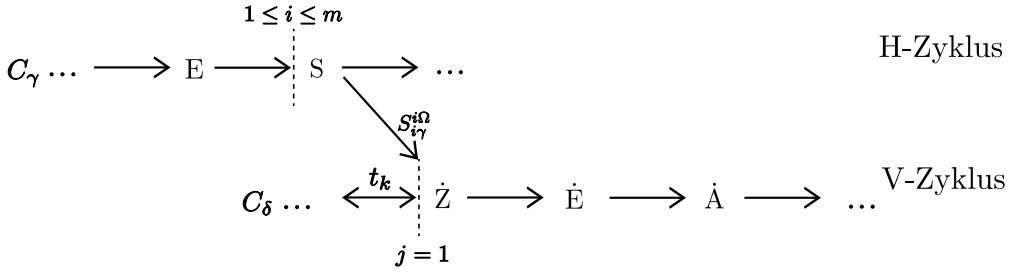


Abbildung 4.2.: Zeitverlauf des V-Zyklus bei inaktiver Taskübernahme

inaktiv ist und sendet $S_{i\gamma}^{i\Omega}$ aus, so dass unmittelbar nachdem Empfang des Suppressor auf $C_\delta = Cz_i$ der Zustandstransfer in $\dot{\mathrm{Z}}$ beginnt. Die Reaktionszeit $t_{S\dot{Z}}$ bei inaktiver Übernahme beschränkt sich somit auf die Zeitdauer, die für das Senden des Übernahme-Suppressor benötigt wird. Unter der Annahme das C_γ und $C_\delta = Cz_i$ im Netz maximal weit voneinander entfernt sein können, gilt für die Reaktionszeit bei inaktiver Taskübernahme die größtmögliche Kommunikationszeit t_K im Netz.

$$\text{Reaktionszeit bei inaktiver Taskübernahme} \quad t_{S\dot{Z}} \leq t_K \tag{4.7}$$

2. Fall: Aktive Taskübernahme Die aktive Übernahme definiert die Zuordnung der Task T_i während die zugehörige Taskverarbeitung *aktiv* ist. Zum Zeitpunkt der Übernahme wird der V-Zyklus also bereits ausgeführt ($1 \geq j$) und die Task T_i befindet sich in einer anderen Instanz auf Kern $C_\delta = Cz_i = Cb_i$ ($\gamma \neq \delta$) in der Verarbeitung. Befindet sich der V-Zyklus während der Übernahme in der Ausführung ($\dot{\mathrm{A}}$), so erfolgt zunächst der Abbruch der Ausführung innerhalb der Wartezeit $t_{\dot{S}}$, bevor der Zustandstransfer zu C_γ im darauf folgenden V-Zyklus ($j + 1$) beginnt. Andernfalls wird die verbleibende Sequenz des V-Zyklus weiter fortgeführt, bis auch Sie nach dem Abbruch der Ausführung terminiert und der Zustandstransfer im Folgezyklus beginnt. Der zeitliche Ablauf ist in Abbildung 4.3 dargestellt. Dieser Fall

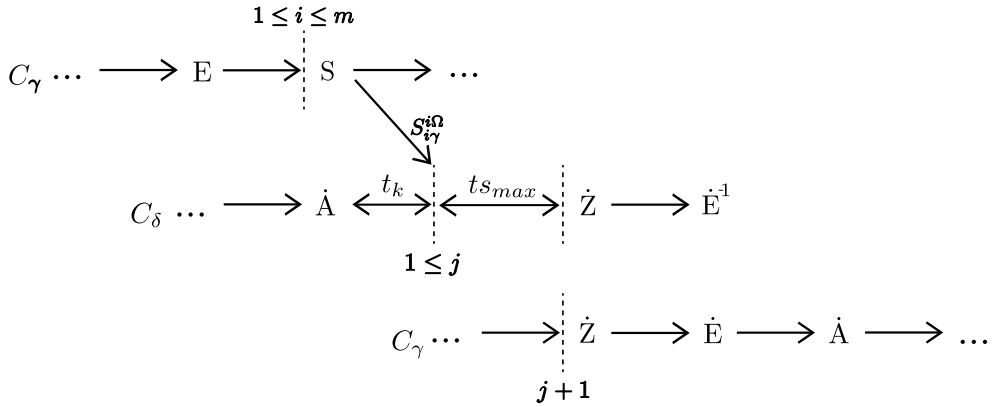


Abbildung 4.3.: Zeitverlauf des V-Zyklus bei aktiver Taskübernahme

ist kritisch und kann zu Inkonsistenzen zwischen dem KHS und der HTV führen, wenn der V-Zyklus zum Zeitpunkt der Übernahme den Schritt der Ausführung noch nicht erreicht hatte. Erfolgt die Neuzuordnung von T_i dennoch bevor dies geschieht, dann erfolgt die Taskverarbeitung solange inkonsistent zur Taskverwaltung bis der Zustandstransfer im konsistenten Folgezyklus beginnt. Der Eintritt eines solch kritischen Verlaufs bei einer Neuzuordnung einer Task ist unter Berücksichtigung der taskgebundenen V-Zykluszeit in Gleichung 4.6 gegenüber der taskunabhängigen des Hormonzyklus durchaus möglich und tritt daher in allen Phasen des KHS auf.

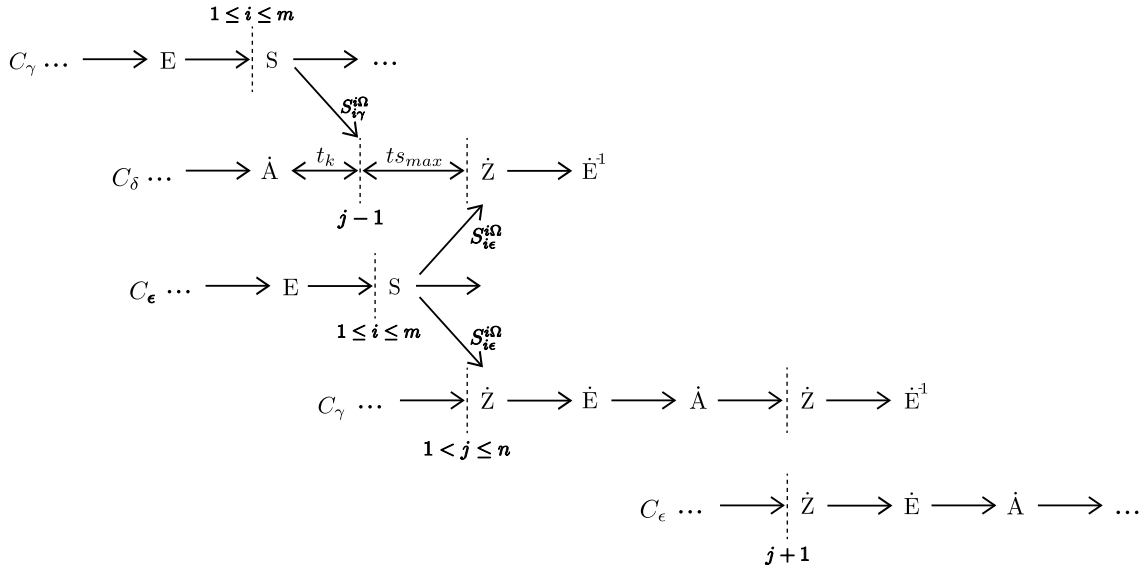


Abbildung 4.4.: Worst-Case Zeitverhalten des V-Zyklus bei aktiver Taskübernahme

Abbildung 4.4 zeigt den schlimmsten anzunehmenden Fall: C_γ überprüft auf Übernahme von T_i und gewinnt. Diese befand sich zum Zeitpunkt der Übernahme von C_γ im vergangenen V-Zyklus $(j-1)$ in einer anderen Instanz auf C_δ in der Ausführung (unkritisch). Nachdem der Zustandstransfer zu C_γ im aktuellen Zyklus (j) gestartet wurde, überprüft nun ein weiterer Kern C_ϵ $(\gamma \neq \delta \neq \epsilon)$ aufgrund eines neu empfangenen Akzelerators auf Übernahme und ist Sieger. Der aktuelle Zyklus befindet sich aber beim Empfangen des neuen Übernahme-Suppressor $S_{i\epsilon}^{i\Omega}$ noch im Zustandstransfer (kritisch). Aufgrund der vorgegebenen Sequenz wird der Zyklus auch nach dem Empfangen des neuen Suppressor weiter fortgesetzt bis die Ausführung abgebrochen und der Folgezyklus mit dem Zustandstransfer nach C_ϵ gestartet wird. Somit ist die Taskverarbeitung während des aktuellen Zyklus, ausgehend vom Zeitpunkt des Empfangs von $S_{i\epsilon}^{i\Omega}$ auf C_γ bis zum Start des Folgezyklus, inkonsistent gegenüber der Taskverwaltung von T_i.

⇒ Der V-Zyklus muss vorzeitig abgebrochen und der Folgezyklus neu gestartet werden, falls sich die Zuordnung von T_i ändert bevor dieser die Ausführung auf C_γ erreicht. Es kann daher die folgende Bedingung für den V-Zyklus bei aktiver Taskübernahme abgeleitet werden, damit die Verarbeitung konsistent gegenüber der Taskverwaltung agiert und die Reaktionszeit weitestgehend verkürzt wird.

1. Vorschrift: Tritt eine Änderung der Taskzuordnung auf bevor der V-Zyklus einer Task den Schritt der Ausführung erreicht hat, so muss dieser sofort beim Empfangen des neuen Übernahme-Suppressor abgebrochen und der Folgezyklus gestartet werden.

Für die Reaktionszeit $t_{S\dot{Z}}$ bei der aktiven Taskübernahme gilt dann, unter Berücksichtigung der oben genannten Vorschrift 1, die Zeitspanne ausgehend vom Senden des Suppressor $S_{i\gamma}^{i\Omega}$ hinweg über den Abbruch der Ausführung innerhalb der Zeitspanne $t_{\dot{S}}$ bis zum Beginn des Zustandstransfer im Folgezyklus $j+1$.

$$\text{Reaktionszeit bei aktiver Taskübernahme} \quad t_{S\dot{Z}} \leq t_{\dot{S}} + t_K \leq t_K + ts_{max} \tag{4.8}$$

Zusammenfassung des fallgebundenen Verhalten der Reaktionszeit Abschließend erfolgt eine zusammenfassende Analyse des zeitlichen Verhalten der Reaktionszeit. Wie bereits bei der Betrachtung der Fälle 1 und 2 gezeigt wurde, ist die Dauer der Reaktionszeit bei inaktiver und aktiver Taskübernahme unterschiedlich. Eine gemeinsame Limitierung in beiden Fällen stellt jedoch die Kommunikationszeit dar, die benötigt wird um die Informationen der Taskübernahme zwischen den betroffenen Kernen im Netz zu verteilen. Für die Ableitung einer allgemeingültigen Aussage über das Zeitverhalten der Reaktionszeit ist es deshalb notwendig, dass eine uniforme Annahme über die Dauer der besagten Kommunikationszeit getroffen wird, damit ein weiterer Vergleich zwischen den unterschiedlichen Reaktionszeiten der beiden Fälle möglich ist. Für die Dauer der Kommunikationszeit im Netz gilt deshalb ab sofort die folgende Annahme:

Annahme zur uniformen Kommunikationszeit: Aufgrund der möglichen, maximalen Entfernung zwischen den taskgebundenen Kernen im Netz von T_i $(C_\gamma, C_\delta, Cz_i, Cb_i)$, gilt für das Senden und Empfangen von Informationen zwischen den Kernen immer die größtmögliche Kommunikationszeit t_K.

Unter Verwendung der obigen Annahme folgt nun der Vergleich der Reaktionszeit aus Fall 1 gegenüber Fall 2 mit anschließender Ableitung der allgemeingültigen Reaktionszeit unter Angabe einer unteren und oberen Schranke. Mittels der uniformen Kommunikationszeit sowie der Vorschrift 1, kann die folgende Ordnung zwischen den unterschiedlichen Reaktionszeiten aus Fall 1 und 2 definiert werden:

$$\text{Fall } 2 \geq \text{Fall } 1$$

Gemäß der Ordnung ist die Reaktionszeit von Fall 2 mindestens genauso groß wie die von Fall 1. Fall 2 schließt also Fall 1 ein und dominiert das Zeitverhalten. Die Ableitung beider Schranken erfolgt deshalb unter Berücksichtigung des Verhalten das bei einer aktiven Taskübernahme in Fall 2 gilt.

Für die maximale Reaktionszeit gilt die in Gleichung 4.8 aufgeführte Abschätzung bezüglich der Worst-Case Laufzeit von Fall 2:

$$\text{Maximale Reaktionszeit} \quad t_{S\dot{Z}} \leq t_K + ts_{max} \tag{4.9}$$

Die Ableitung der unteren Schranke erfolgt unter Betrachtung des Best-Case Zeitverhalten von Fall 2. Im Idealfall gilt bei der aktiven Taskübernahme, dass die für den Abbruch von Schritt Ȧ benötigte Zeitspanne mit $t_{\dot{S}} = 0$ gegen Null konvergiert.

Für die minimale Reaktionszeit gilt deshalb:

$$\text{Minimale Reaktionszeit} \quad t_{S\dot{Z}} \geq t_K \tag{4.10}$$

4.1.3.2. Betrachtung und Analyse der Vorlaufzeit

Im Gegensatz zur Reaktionszeit umfasst die Vorlaufzeit die Durchführung der Schritte Zustandstransfer (Ż) und Instanzerzeugung (Ė) des V-Zyklus. Die Ausführung der beiden Schritte wird im wesentlichen von den Transfers der Taskkomponenten von T_i, zwischen den Kernen C_δ und C_γ geprägt. Erneut repräsentiert C_δ den Kern, welcher die Task verliert und C_γ den Kern, der die Task übernimmt. Zu den Komponenten von T_i zählt die Beschreibung und dessen optionaler Zustand (initial oder aktuell). Die zugehörigen Datenmengen werden mit Db_i und Dz_i bezeichnet. Die Dauer eines Transfers wird von der Größe der Datenmenge, der Kommunikationszeit und vom Speicherverhalten der Kerne bezüglich den Komponenten beeinflusst. Das reguläre Speicherverhalten des V-Zyklus definiert sich dadurch, dass C_δ nur solange eine Komponente von T_i speichert, bis diese vollständig an den übernehmenden Kern C_γ transferiert wurde. Andernfalls speichert C_δ zu keinem Zeitpunkt die Komponenten von T_i, mit Ausnahme einer möglichen initialen Speicherung bei Systemstart. Ein unabhängiges Speicherverhalten definiert sich dadurch, dass beide Kerne auch ungeachtet der gegenwärtigen Zuordnung und Verarbeitung von T_i dessen Komponenten im Netz speichern können. Das Speicherverhalten legt somit fest, ob Kern C_γ, welcher die Task T_i übernimmt, zum Zeitpunkt der Übernahme bereits lokal über entsprechende Komponenten verfügt. Andernfalls müssen über die transferbehafteten Schritte Ż und Ė des V-Zyklus die entsprechenden Komponenten zum Zielort nachgeladen werden. Dabei wird die Rolle der Transfer initiierenden Kerne Cb_i und Cz_i je nach Speicherverhalten unterschiedlich von C_γ, C_δ oder auch anderen Kernen im Netz übernommen. Es ist daher möglich, dass sich Cb_i und Cz_i in ihrer Rolle wechselseitig darstellen oder voneinander unterscheiden, falls Komponenten am gleichen oder an unterschiedlichen Orten im Netz gespeichert werden. Diesbezüglich lassen sich zwischen den betroffenen Kernen C_γ, Cz_i und Cb_i fünf verschiedene Relationen definieren, die den folgenden drei Fällen der Verfügbarkeit von Komponenten im Netz zuzuordnen sind.

1. Fall: Globale Verfügbarkeit Die globale Verfügbarkeit (g.V.) definiert die Speicherung von Komponenten am Ort der vorangegangenen Taskzuordnung von T_i. In diesem Fall verfügt Kern C_δ, dessen H-Zyklus die Zuordnung verliert, über beide

Komponenten zum Zeitpunkt der Übernahme von T_i durch C_γ. Es gilt daher die folgende Relation zwischen den betroffenen Kernen.

Relation 1. $C_\gamma \neq Cz_i = Cb_i$

Diese Variante der Verfügbarkeit entspricht dem regulären Speicherverhalten, dass sich bei der Verarbeitung von Tasks durch den V-Zyklus abbildet. Die Speicherung von Komponenten erfolgt zyklusorientiert und impliziert, dass T_i im Verlauf seiner Verarbeitung zuvor auf einem Kern C_δ ausgeführt wurde und dieser folglich über beide Komponenten verfügt. Bei einer globalen Verfügbarkeit von Komponenten gilt deshalb, dass die Verfügbarkeit einer Komponente wechselseitig die der anderen einschließt und beide im Falle einer Neuzuordnung solange lokal erhalten bleiben bis die Instanzerzeugung auf C_γ ausgeführt wurde. Die Ausnahme bilden Varianten der lokalen Verfügbarkeit in Fall 3, die infolge einer instabilen oder initialen Taskzuordnung auftreten können.

Nach Ablauf der Reaktionszeit startet der V-Zyklus und C_δ beginnt mit dem sequentiellen Transfer der zustandsbehafteten Datenhormone in $\dot{\mathrm{Z}}$ nach C_γ, gefolgt von denen der Taskbeschreibung in Schritt $\dot{E}$. Nach Abschluss der Instanzerzeugung in $\dot{\mathrm{E}}$ startet die Taskausführung auf C_γ nach Ablauf der Zeit $t_{\dot{E}\dot{A}}$. Der zeitliche Verlauf wurde bereits bei der Betrachtung der Reaktionszeit in Abbildung 4.3 dargestellt. Entscheidend für die korrekte Funktionsweise der Taskverarbeitung bei globaler Verfügbarkeit von Komponenten, ist der sequentielle Empfang aller Datenhormone $Dz_{i\delta}^{i\gamma}$ und $Db_{i\delta}^{i\gamma}$ auf C_γ, damit der Zyklus die Schritte $\dot{\mathrm{Z}}$ und $\dot{\mathrm{E}}$ durchläuft und die Ausführung erreicht. Die minimale Vorlaufzeit beschränkt sich somit bei globaler Verfügbarkeit der Komponenten gemäß Relation 1 auf die volle Komplexität der Vorverarbeitung und somit auf die in Gleichung 4.5 definierte Zeitdauer.

$$\begin{aligned} \text{Vorlaufzeit g.V.} \quad t_{\dot{Z}\dot{A}} &= t_{\dot{Z}} + t_{\dot{E}} + t_{\dot{E}\dot{A}} \\ &\leq t_K \cdot \left(\left\lceil \frac{Dz_{max}}{Dh} \right\rceil + \left\lceil \frac{Db_{max}}{Dh} \right\rceil \right) + ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.11}$$

2. Fall: Separierte Verfügbarkeit Die separierte Verfügbarkeit (s.V.) definiert die getrennte Speicherung von Komponenten auf voneinander unterschiedlichen Kernen im Netz. Kern Cz_i verfügt über den aktuellen Zustand von T_i und repräsentiert bei einer aktiven Taskübernahme Kern C_δ, welcher die Zuordnung von T_i verliert. Kern Cb_i verfügt unabhängig vom Verlauf der Verarbeitung von T_i über die Beschreibung. Es gilt also zwischen den Kernen die folgende Relation.

Relation 2. $C_\gamma \neq Cz_i \neq Cb_i$

Diese Variante der Verfügbarkeit entspricht einem unabhängigen Speicherverhalten, dass ungeachtet der Verarbeitung des V-Zyklus über die Beschreibung von Tasks verfügt und auch agiert. Der V-Zyklus beginnt regulär nachdem der Broadcast der

Taskübernahme von C_γ ausgeführt und die beiden Kerne Cz_i und Cb_i informiert wurden. Unter Annahme einer uniformen Kommunikationszeit starten die Transfers auf beiden Kernen annähernd zeitgleich, so dass eine parallele Durchführung der beiden Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ erfolgt. Bei einer individuellen Kommunikationszeit ist es hingegen möglich, dass der Empfang des Übernahme-Suppressor auf beiden Kernen zeitlich variiert, so dass bei der Ausführung der Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ gegebenenfalls ein zeitlicher Versatz ($\leq t_K$) entsteht. Da der V-Zyklus aber regulär nach Ablauf der einheitlich definierten Reaktionszeit $t_{S\dot{Z}}$ startet, gilt auch hier, dass der Ablauf von $\dot{\text{E}}$ spätestens zeitgleich mit dem in $\dot{\text{Z}}$ startet, was wiederum dem Fall einer uniformen Kommunikationszeit entspricht. Deshalb gilt bei einer separierten Verfügbarkeit unabhängig von der Art der Kommunikationszeit (uniform oder individuell) stets eine nebenläufige Ausführung von Zustandstransfer und Instanzerzeugung, deren zeitliche Abschätzung unter Annahme der in Abschnitt 4.1.3.1 getätigten uniformen Kommunikationszeit erfolgt[1]. Für den V-Zyklus gilt daher eine weitere Vorschrift:

2. Vorschrift: Erfolgt die Speicherung der Taskbeschreibung unabhängig gegenüber der Verarbeitung des V-Zyklus, dann erfolgt der zeitliche Ablauf der Instanzerzeugung nebenläufig zu der des Zustandstransfers.

Des Weiteren ist bei der Nebenläufigkeit zu berücksichtigen, dass die Instanzerzeugung nur dann einen zeitlichen Einfluss auf die Vorlaufzeit hat, wenn die Ausführung von Schritt $\dot{\text{E}}$ nach der von $\dot{\text{Z}}$ endet. Es gilt daher, dass sich der zeitliche Einfluss von Schritt $\dot{\text{E}}$ und $\dot{\text{Z}}$ auf die Vorlaufzeit wechselseitig durch den jeweils anderen einschließt und somit nur durch einen der beiden Schritte definiert. Wessen Einfluss das zeitliche Verhalten dominiert, kann aus dem Maximum der benötigten Zeitdauer für $\dot{\text{Z}}$ und $\dot{\text{E}}$ bestimmt werden. Die Ableitung der Vorlaufzeit bei separierter Verfügbarkeit gemäß Relation 2 kann daher aus dem Maximum der beiden Zeitspannen $t_{\dot{Z}}$ und $t_{\dot{E}}$ plus der Wartezeit $t_{\dot{E}\dot{A}}$ für den Start der Ausführung berechnet werden.

$$\begin{aligned} \text{Vorlaufzeit s.V.} \quad t_{\dot{Z}\dot{A}} &= max\{t_{\dot{Z}}, t_{\dot{E}}\} + t_{\dot{E}\dot{A}} \\ &\leq t_K \cdot \left\lceil \frac{max\{Dz_{max}, Db_{max}\}}{Dh} \right\rceil + ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.12}$$

3. Fall: Lokale Verfügbarkeit Die lokale Verfügbarkeit (l.V.) definiert das vorzeitige Vorhandensein von Komponenten am Zielort der Taskzuordnung von T_i. In diesem Fall verfügt Kern C_γ entweder über den aktuellen Zustand, die Beschreibung oder über beides. Lokale Verfügbarkeiten treten je nach Relation auch im regulären

[1] Bei einer individuellen Kommunikationszeit kann die tatsächliche Dauer der beiden Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ gegenüber der Abschätzung bei uniformer Kommunikationszeit um maximal $\leq t_K$ abweichen.

Verlauf der Taskverarbeitung auf und sind daher nicht an die in Fall 2 beschriebene separierte Speicherung gebunden. Es sind die folgenden Relationen einer lokalen Verfügbarkeit von Komponenten zu unterscheiden:

Relation 3. $C_\gamma = Cz_i = Cb_i$

Relation 4. $C_\gamma = Cz_i \neq Cb_i$

Relation 5. $C_\gamma = Cb_i \neq Cz_i$

Relation 3 definiert die bestmögliche Variante einer lokalen Verfügbarkeit bei der beide Komponenten zum Zeitpunkt der Übernahme auf C_γ vorhanden sind. Somit entfällt der Aufwand für den Transfer der Komponenten zum Zielort. Die Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ können daher in der Sequenz übersprungen werden, so dass deren Einfluss auf die Vorlaufzeit entfällt. Eine Verfügbarkeit beider Komponenten am Zielort der Zuordnung tritt bei einer aktiven Taskübernahme von T_i innerhalb des V-Zyklus auf. Dies ist genau dann der Fall, wenn ein Kern C_γ die Zuordnung von T_i gegenüber einem Kern C_δ ($\delta \neq \gamma$) verliert und diese innerhalb der Vorverarbeitung des V-Zyklus zurück erlangt, also bevor auf C_γ die Instanztermination ausgeführt wurde. Die Vorlaufzeit beschränkt sich bei einer lokalen Verfügbarkeit gemäß Relation 3 auf die Wartezeit $t_{\dot{E}\dot{A}}$ die zum Starten der Taskausführung benötigt wird. Bei einer Höchstanzahl von m_{max} Tasks auf die sich ein Kern bewirbt gilt somit:

$$\begin{aligned} \text{Vorlaufzeit Rel. 3} \quad t_{\dot{Z}\dot{A}} &= t_{\dot{E}\dot{A}} \quad | \, t_{\dot{Z}}, t_{\dot{E}} = 0 \\ &\leq ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.13}$$

Relation 4 definiert die einseitige, lokale Verfügbarkeit des aktuellen Zustands von T_i auf C_γ. Der Zustandstransfer entfällt und Schritt $\dot{\text{Z}}$ kann übersprungen werden, so dass sich die Vorverarbeitung des Zyklus gegenüber der von Relation 3 nur bedingt verkürzt. Diese Variante tritt ebenfalls bei einer aktiven Übernahme von Tasks auf und kann durch folgenden Fall erklärt werden. Kern C_γ übernimmt T_i. Während der Vorverarbeitung wird der aktuelle V-Zyklus aufgrund einer Neuzuordnung gegenüber C_δ abgebrochen. Der Zustandstransfer nach C_γ war diesem Zeitpunkt bereits vollständig ausgeführt, weshalb nun C_γ seinerseits den Transfer in $\dot{\text{Z}}$ nach C_δ einleitet. Erhält nun C_γ die Zuordnung erneut zurück, bevor der Transfer nach C_δ endet, dann verfügt C_γ lokal über den aktuellen Zustand gemäß Relation 2. Die Vorlaufzeit erstreckt sich dann über den Transfer der Beschreibung in $\dot{\text{E}}$ bis zum Start der Ausführung nach Ablauf von $t_{\dot{E}\dot{A}}$.

$$\begin{aligned} \text{Vorlaufzeit Rel. 4} \quad t_{\dot{Z}\dot{A}} &= t_{\dot{E}} + t_{\dot{E}\dot{A}} \quad | \, t_{\dot{Z}} = 0 \\ &\leq t_K \cdot \left\lceil \frac{Db_{max}}{Dh} \right\rceil + ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.14}$$

Relation 5 definiert die einseitige, lokale Verfügbarkeit der Beschreibung von T_i auf C_γ. Umgekehrt zu Relation 4 entfällt der Aufwand für den Transfer der Beschreibung und Schritt $\dot{\text{E}}$ kann in der Sequenz übersprungen werden. Diese Variante tritt bei einer regulären Verarbeitung durch den V-Zyklus nicht auf, ist aber in Kombination mit dem in Fall 2 beschriebenen Speicherverhalten möglich wenn die lokale Instanztermination unabhängig gegenüber der aktuellen Instanzerzeugung des V-Zyklus erfolgt und Komponenten auch nach dem Transfer für eine bestimmte Zeit weiterhin lokal erhalten bleiben. Im Falle einer Rückübernahme von T_i durch C_γ wären dann beide Komponenten noch lokal verfügbar, aber der ursprüngliche Zustand von C_γ wäre aufgrund der fortgeschrittenen Weiterverarbeitung inkonsistent und müsste gemäß Relation 5 nachgeladen werden. Die Vorlaufzeit beschränkt sich daher bei einer lokalen Verfügbarkeit gemäß Relation 3 auf den Zustandstransfer plus der Wartezeit die zum Starten der Ausführung benötigt wird.

$$\begin{aligned} \text{Vorlaufzeit Rel. 5} \quad t_{\dot{Z}\dot{A}} &= t_{\dot{Z}} + t_{\dot{E}\dot{A}} \quad | \, t_{\dot{E}} = 0 \\ &\leq t_K \cdot \left\lceil \frac{Dz_{max}}{Dh} \right\rceil + ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.15}$$

Unter Berücksichtigung der Relationen 3, 4 und 5 kann eine weitere Bedingung für den V-Zyklus abgeleitet werden, um die Vorlaufzeit bei lokaler Verfügbarkeit von Komponenten zu minimieren.

3. Vorschrift: Verfügt ein Kern zu Beginn des V-Zyklus bereits über zugehörige Taskkomponenten, dann können die Schritte Zustandstransfer und Instanzerzeugung je nach Verfügbarkeit in der Zyklussequenz übersprungen werden (Bypass).

Die Ableitung der regulären Vorlaufzeit bei lokaler Verfügbarkeit erfolgt über den Größenvergleich zwischen den Relationen. Damit ein Vergleich möglich ist, gilt die in Abschnitt 4.1.3.1 getätigte Annahme zur uniformen Kommunikationszeit. Die Vorlaufzeit definiert sich dann über die größte Laufzeit, die aufgrund des Größenverhältniss von $Db_{max} > 0$ Byte und $Dz_{max} \geq 0$ Byte, entweder von Relation 3 oder Relation 4 ausgeht. Für die Vorlaufzeit gilt deshalb das Maximum aus beiden.

$$\begin{aligned} \text{Vorlaufzeit bei l.V.} &= max\{Rel.\ 3, Rel.\ 4\} = max\{t_{\dot{Z}}, t_{\dot{E}}\} + t_{\dot{E}\dot{A}} \\ &\leq t_K \cdot \left\lceil \frac{max\{Dz_{max}, Db_{max}\}}{Dh} \right\rceil + ts_{max} \cdot (m_{max} - 1) \end{aligned} \tag{4.16}$$

Zusammenfassung des fallgebundenen Verhaltens der Vorlaufzeit Es folgt die zusammenfassende Betrachtung der lokalen Vorlaufzeiten aus Fall 1, 2 und 3 zur

Ableitung einer allgemeingültigen oberen und unteren Schranke für die Vorlaufzeit. Wie bei der einzelnen Analyse der Fälle ersichtlich wurde, ist die Vorlaufzeit weitestgehend vom zeitlichen Aufwand geprägt, der beim Transfer der Taskkomponenten zum Zielort der Taskzuordnung verstreicht. Je nach Art der Verfügbarkeit ist ein unterschiedlicher zeitlicher Ablauf der transferbehafteten Schritte während der Vorverarbeitung möglich, der seinerseits entweder sequentiell, nebenläufig oder mittels Bypass erfolgen kann. Die Vorlaufzeit kann daher in den einzelnen Fällen stark variieren. Ähnlich wie bei der Analyse der Reaktionszeit stellt aber auch hier die Kommunikationszeit, die für den Transfer von Informationen zwischen den beteiligten Kernen benötigt wird, den limitierenden Faktor dar. Deshalb kann unter Annahme der uniformen Kommunikationszeit aus Abschnitt 4.1.3.1 und der Bedingung für die kleinstmöglichen Datenmengen von Taskkomponenten mit $Db_{max} = 1$ Byte und $Dz_{max} = 0$ Byte, die folgende Größenordnung bezüglich der Vorlaufzeiten aus Fall 1, 2 und 3 abgeleitet werden:

$$\text{Fall } 1 \geq \text{Fall } 2 \geq \text{Fall } 3$$

Ausgehend von der größten Vorlaufzeit in Fall 1, folgt in der Ordnung Fall 2. Dies gilt, da nur im Best-Case (kleinstmögliche Datenmengen, $Db_{max} > Dz_{max} = 0$ Byte) die Laufzeit bei beiden gleich ist. Ansonsten ist die Laufzeit von Fall 1 immer größer als die von Fall 2. Fall 3 stellt in der Ordnung die kleinste Vorlaufzeit dar, weil dessen reguläre Laufzeit mit der von Fall 2 übereinstimmt, aber im Best-Case (lokale Verfügbarkeit beider Komponenten, Rel. 3) kleiner ist. Somit wird Fall 3 von den beiden anderen in der Ordnung dominiert. Folglich ist für die Definition der beiden Schranken das zeitliche Verhalten der globalen Verfügbarkeit in Fall 1 bindend.

Somit ergibt sich für die obere Schranke der Vorlaufzeit die in Fall 1 gültige Worst-Case Laufzeit.

$$\text{Maximale Vorlaufzeit} \quad t_{\dot{Z}\dot{A}} \leq t_K \cdot \left(\left\lceil \frac{Dz_{max}}{Dh} \right\rceil + \left\lceil \frac{Db_{max}}{Dh} \right\rceil \right) + ts_{max} \cdot (m_{max} - 1) \tag{4.17}$$

Die Abschätzung der minimalen Vorlaufzeit erfolgt unter Berücksichtigung des bestmöglichen Zeitverhalten von Fall 1. Dies ist genau dann der Fall, wenn einerseits die Wartezeit $t_{\dot{E}\dot{A}}$ zum Starten der Ausführung mit $m_{max} = 1$ gegen Null konvergiert und der Zustandstransfer aufgrund der kleinstmöglichen Datenmengen entfällt ($Db_{max} > Dz_{max} = 0$ Byte).

Für die minimale Vorlaufzeit gilt dann die folgende untere Schranke.

$$\text{Minimale Vorlaufzeit} \quad t_{\dot{Z}\dot{A}} \geq t_K \tag{4.18}$$

4.1.3.3. Abschließende Abschätzung der V-Zykluszeit

Abschließend erfolgt die Betrachtung des zeitlichen Gesamtverhaltens des V-Zyklus in Bezug auf die in Abschnitt 4.1.3.1 und 4.1.3.2 abgeleiteten Aussagen über das Verhalten der Reaktions- und Vorlaufzeit. Diese werden nun zusammengeführt, damit eine allgemeingültige Abschätzung über die Gesamtdauer des V-Zyklus abgeleitet werden kann. Dies erfolgt unter Angabe der kleinstmöglichen Zykluszeit in Anbetracht der Verbesserungen von Vorschrift 1, 2 und 3. Wie bereits zu Beginn von Kapitel 4.1 erwähnt wurde, ist die V-Zykluszeit weitestgehend Task gebunden, kann jedoch durch Optimierung der zur Taskverarbeitung notwendigen Abläufe des Zyklus verkürzt werden.

Die Reaktionszeit konnte soweit verbessert werden, dass sich der zeitliche Ablauf , der bis zum Start der Taskverarbeitung verstreicht, auf eine Zyklussequenz mit einer Minimalanzahl an Schritten beschränkt, die für die weitere Vorverarbeitung notwendig sind. So ist im Best-Case mit einer kleinstmöglichen Reaktionszeit zu rechnen, die sich lediglich auf die Kommunikationszeit beschränkt welche für das Senden der Taskübernahme im Netz verstreicht. Im Worst-Case Fall hingegen ist mit einer zusätzlichen Wartezeit zu rechnen die für den Abbruch der Taskausführung benötigt wird, sofern die Task nicht unmittelbar nach dem Empfang der Taskübernahme abgebrochen werden kann. Entscheidend für die Verkürzung der Reaktionszeit ist das in Vorschrift 1 formulierte Abbruchkriterium, welches dem V-Zyklus erlaubt jeden Schritt innerhalb der Vorverarbeitung zu jedem Zeitpunkt abzubrechen um direkt den Folgezyklus einzuleiten. Dies garantiert neben einer minimalen Reaktionszeit auch die Konsistenz der Taskverarbeitung gegenüber der Taskverwaltung, so dass letztlich die Reaktion auf eine Neuzuordnung sofort erfolgt.

Bei der Betrachtung der Vorlaufzeit wurde deutlich, dass deren Zeitdauer stark variiert und der zeitliche Aufwand taskabhängig ist. Die Optimierung der Vorlaufzeit beschränkt sich deshalb im Einzelnen auf den jeweiligen Fall, der sich in der Art der Verfügbarkeit von Taskkomponenten unterscheidet.

Bei lokaler Verfügbarkeit ist im bestmöglichen Fall mit einer Speicherung beider Komponenten zu rechnen, so dass die kleinstmögliche Zeitdauer gilt, die sich über die Wartezeit nach der Vorverarbeitung bis zum Start der Ausführung erstreckt und im Idealfall gegen Null konvergiert. Andernfalls beschränkt sich die Zeitdauer auf den Transfer von nur einer der beiden Komponenten plus der Wartezeit zum Start. Entscheidend für die Verkürzung der Vorlaufzeit bei lokaler Verfügbarkeit ist die in Vorschrift 3 ergänzte Bypass-Funktion des V-Zyklus. Somit ist es der Taskverarbeitung möglich, bereits verfügbare Komponenten bei erneuter Taskzuordnung wiederzuverwenden, was entweder bei einer instabilen Verarbeitung oder einem unabhängigen Speicherverhalten gegenüber dem V-Zyklus auftritt. Dadurch können die transferbehafteten Schritte des Zyklus je nach Verfügbarkeit übersprungen werden, so dass deren Einfluss auf die Zeitdauer entfällt.

Bei separierter Verfügbarkeit gilt aufgrund der getrennten Speicherung ein un-

abhängiges Speicherverhalten gegenüber der Beschreibung von Tasks. Ein Optimierung stellt hier die in Vorschrift 2 definierte Eigenschaft des V-Zyklus dar, die eine nebenläufige Ausführung des Zustandstransfer und der Instanzerzeugung erlaubt. Dadurch entfällt der zeitliche Einfluss von einer der beiden Schritte, weshalb sich die Vorlaufzeit über die größere der beiden Ausführungszeiten plus der Wartezeit zum Start erstreckt.

Anders verhält es sich bei globaler Verfügbarkeit der schlechtesten Variante, die aber zugleich dem regulären Verhalten der Taskverarbeitung durch den V-Zyklus entspricht. Der zeitliche Aufwand definiert sich im Regelfall über die sequentielle Ausführung aller transferbehafteten Schritte der Vorverarbeitung, bedingt durch die globale Speicherung von Komponenten. Eine Verkürzung der Vorlaufzeit ist hier nur im Best-Case bei kleinstmöglicher Datenmenge von Komponenten möglich.

Zusammenfassend wurden bei der Betrachtung der Reaktions- und Vorlaufzeit in den Abschnitten 4.1.3.1 und 4.1.3.2 die folgenden Vorschriften zur Optimierung des Zeitverhalten des V-Zyklus ergänzt:

1. Vorschrift Tritt eine Änderung der Taskzuordnung auf, bevor der V-Zyklus einer Task den Schritt der Ausführung erreicht hat, so muss dieser sofort beim Empfangen des neuen Übernahme-Suppressor abgebrochen und der Folgezyklus gestartet werden.

2. Vorschrift: Erfolgt die Speicherung der Taskbeschreibung unabhängig gegenüber der Verarbeitung des V-Zyklus, dann erfolgt der zeitliche Ablauf der Instanzerzeugung nebenläufig zu der des Zustandstransfer.

3. Vorschrift: Verfügt ein Kern zu Beginn des V-Zyklus bereits über zugehörige Taskkomponenten, dann können die Schritte Zustandstransfer und Instanzerzeugung je nach Verfügbarkeit in der Zyklussequenz übersprungen werden (Bypass).

Anhand der obigen Vorschriften kann eine modifizierte Funktionsweise des V-Zyklus abgeleitet werden, dessen Schema in Abbildung 4.5 dargestellt ist. Dementsprechend wurden die in den Vorschriften beschriebenen Eigenschaften und Funktionen in das ursprüngliche Zyklusverhalten integriert, weshalb die in Abschnitt 4.1.3 beschriebene Grundsequenz des V-Zyklus auch weiterhin gültig ist. Die vorzeitige Abbruchkontrolle bezüglich Vorschrift 1 ist aus Sichtweise der transferbehafteten Schritte so zu deuten, dass der Neustart des Zyklus ohne eine weitere zeitliche Verzögerung erfolgt. Bei einem Bypass gemäß Vorschrift 3 ist der Sequenzverlauf so zu interpretieren, dass beim Überspringen des jeweiligen Schrittes dessen Zeitdauer entfällt, keine weitere Verzögerung durch den Bypass selbst entsteht und dieser direkt in die nachfolgende Sequenz überführt. Angrenzende

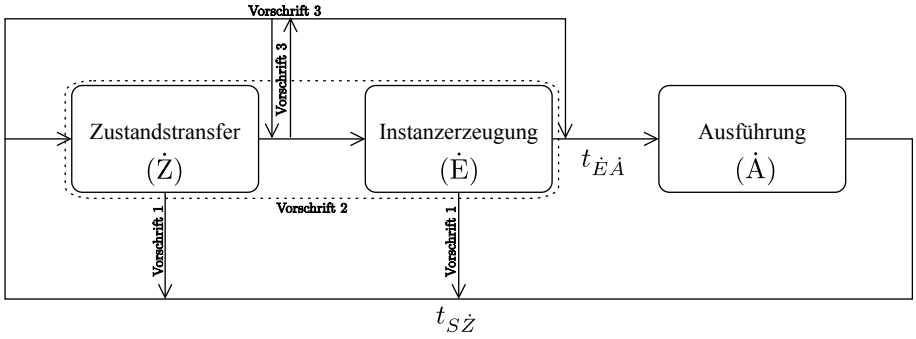


Abbildung 4.5.: Modifizierter V-Zyklus

Wartezeiten bleiben auch weiterhin in voller Länge bestehen. Das nebenläufige Zeitverhalten von Schritt Ė aus Vorschrift 2 ist so zu deuten, dass sich der zeitliche Einfluss von Schritt Ż und Ė auf die Gesamtdauer nur aus dem zeitlich längeren der beiden definiert.

Des Weiteren wurde die folgende Annahme und Bedingung bezüglich der uniformen Kommunikationszeit und dem Größenverhältnis der kleinstmöglichen Datenmenge von Taskkomponenten getätigt:

- Maximale Kommunikationszeit t_K zwischen den beteiligten Kernen
- Größenverhältnis der minimalen Datenmenge mit $Db_{max} > Dz_{max} = 0$ Byte

Daraus ergibt sich für die minimale V-Zykluszeit:

$$\text{Minimale V-Zykluszeit} \quad t_{VC} = t_{S\dot{Z}} + t_{\dot{Z}\dot{A}} \geq 2 \cdot t_K \tag{4.19}$$

4.2. Funktionsweise und Zeitverhalten bei Selbst-Konfiguration

Die Selbst-Konfiguration repräsentiert die zentrale Kerneigenschaft der Architektur, bei der alle Tasks nach Systemstart durch das KHS verteilt und durch die HTV gestartet werden. Diesbezüglich wird nun die Funktionsweise, unter Einschluss des ergänzten Taskverarbeitungsmodell detailliert erörtert und das Zeitverhalten untersucht.

Die Funktionsweise setzt sich aus der Verknüpfung des in Kapitel 3.1.6 beschriebenen H-Zyklus mit dem im vorangegangenen Abschnitt definierten V-Zyklus zusammen. Das gemeinsame Funktionsprinzip aus beiden Kontrollzyklen ist dann wie folgt: Nach Systemstart überprüft jeder Kern in jedem H-Zyklus für sich, ob er aufgrund seiner Eignung eine Task übernehmen darf, auf die er sich bewirbt. Hierfür tauschen alle Kerne ihre Hormone in jedem H-Zyklus untereinander aus. Entscheidet sich ein

Kern zur Übernahme einer Task, so werden ab dem folgenden H-Zyklus zugehörige Suppressoren ausgeschüttet, um die Übernahme im Netz zu kommunizieren und fehlerhafte Mehrfachübernahmen zu verhindern. Das Aussenden der Suppressoren initiiert den V-Zyklus. Dieser leitet die notwendigen Schritte der Taskverarbeitung ein und erreicht nach deren Durchführung, wie auch die Task selbst, die Ausführung. Für jede Taskübernahme wird immer ein zugehöriger V-Zyklus initiiert, der solange aktiv bleibt, bis die Task eigenständig oder unvorhergesehen terminiert. Dieses Prinzip wird solange auf allen Kernen wiederholt, bis alle Tasks verteilt und alle V-Zyklen gestartet wurden. Die Selbst-Konfiguration ist beendet, sobald sich alle Tasks in der Ausführung befinden.

Nun wird die obere Schranke für die Gesamtzeitdauer der Selbst-Konfiguration bestimmt. Durch das Zusammenspiel beider Kontrollzyklen, lässt sich das Zeitverhalten in zwei Bereiche aufteilen. In 1. wird die obere Schranke für die Verteilung aller Tasks bemessen. Gleichermaßen erfolgt dies für das angrenzende Starten aller Tasks in 2.

1. Worst-Case Zeitdauer bis zur Übernahme aller Tasks (H-Zyklus)

2. Worst-Case Zeitdauer bis zur Ausführung aller Tasks (V-Zyklus)

Die Dauer die für die Taskübernahme in 1. benötigt wird, hängt von der verwendeten Strategie des KHS ab, welche die Auswahl der Task übernimmt, die von einem Kern im aktuellen H-Zyklus auf Übernahme geprüft wird. In [BPvR08] wurde die klassische Variante vorgestellt. Diese führt eine periodische Überprüfung von Tasks gemäß der aufsteigenden Indexmenge IM $= \{1..m\}$ durch. Wird ein erhöhter Eignungswert für eine Task gesendet, der höher ist als alle bisherigen empfangenen Werte, so wird die Reihenfolge unterbrochen und die betroffene Task direkt geprüft. Anschließend wird die ursprüngliche Reihenfolge fortgesetzt. Aufgrund der maximalen Anzahl an Tasks, auf die sich ein Kern bewirbt, sowie der maximalen Anzahl an Taskverwandtschaften gilt, dass im schlimmsten Fall die Task T_i erst nach m_{max} Zyklen überprüft wird. Die Entscheidung kann dann aufgrund von Akzelerator Einfluss noch v_{max}-mal verschoben werden, bevor diese gefällt und T_i übernommen wird. Für eine Task ist die Übernahme daher spätestens nach $m_{max} + v_{max} \leq 2m - 1$ H-Zyklen beendet. Dies entspricht zugleich der oberen Schranke für die Verteilung aller Tasks. In [BP12] wurde eine weitere Variante vorgestellt, die eine aggressive Übernahme von Tasks forciert. Im Gegensatz zur Klassischen werden Tasks nicht nur periodisch oder infolge von erhöhten Eignungswerten geprüft, sondern auf Basis der größten Eignung gegenüber allen Tasks, auf die sich ein Kern bewirbt. Ein Kern überprüft daher gezielt die Task, für die er Gewinner ist, sofern vorhanden. Da es für jede Task immer einen bestgeeigneten Kern gibt, wird in jedem H-Zyklus mindestens eine Task übernommen. Im schlimmsten Fall ist die Übernahme von T_i sowie der restlichen Tasks nach m_{max} Zyklen beendet. Die bei beiden Strategien angegebene Worst-Case Anzahl an H-Zyklen berücksichtigt, dass während der Selbst-Konfiguration keine

Optimierung oder Heilung von Tasks erfolgt. Für die Zeitdauer in 1. ergibt sich für beide Strategien eine unterschiedliche Laufzeit für die Verteilung aller Tasks. Diese wird als vielfaches der statischen H-Zykluszeit berechnet und ist in 4.20 aufgeführt.

$$\text{Worst-Case Taskübernahme in 1.} \; = t_{HC} \cdot h \tag{4.20}$$

$$h := \begin{cases} m_{max} + v_{max} & : \text{klassisch} \\ m_{max} & : \text{aggressiv} \end{cases}$$

Erfolgt die Betriebsweise des KHS asynchron, so kann sich die Dauer aufgrund des größtmöglichen, zeitlichen Versatzes zwischen Kernen (1 H-Zyklus) um maximal eine Zykluszeit bei beiden Strategien verschlechtern.

Im Gegensatz zur Taskverteilung ist die Zeitdauer bis zum Start der Taskausführung in 2. für jede Task unterschiedlich, jedoch unabhängig gegenüber der Anzahl von Tasks und deren Verwandtschaften. Für jede Übernahme einer Task T_i wird der zugehörige V-Zyklus gestartet. Dieser erreicht nach Ablauf der dynamischen Zykluszeit t^i_{VC} die Ausführung und verweilt solange in diesem Zustand, bis sich die Zuordnung durch eine erneute Übernahme von T_i ändert. Da aber während der Selbst-Konfiguration jede Task nur einmal übernommen wird, bleibt die Zuordnung erhalten und T_i erreicht spätestens nach einmaligem Durchlauf des V-Zyklus die Ausführung. Es gilt daher, dass während der Selbst-Konfiguration der zugehörige V-Zyklus einer jeden Task nur einmal durchlaufen wird. Da aber die Übernahme von Tasks zu verschiedenen H-Zyklen erfolgt, werden auch die V-Zyklen unterschiedlich gestartet. Die Durchführung des V-Zyklus von T_i überdeckt sich mit denen der anderen Tasks und es entsteht ein paralleler Zeitverlauf. Deshalb wird die Zeitdauer bis sich alle Tasks in der Ausführung befinden von der Task mit der größten Zykluszeit in 4.21 beschränkt.

$$\text{Worst-Case Taskausführung in 2.} \; = {}_{max}t_{VC} \tag{4.21}$$

$${}_{max}t_{VC} := \max_{T_i \in M}\{t^i_{VC}\}, \quad \text{die maximale V-Zykluszeit einer Task}$$

Das Zeitverhalten der Selbst-Konfiguration folgt nun aus der Verknüpfung beider Worst-Case Laufzeiten für 1. und 2. Unabhängig von der verwendeten Strategie in 1. wird im schlimmsten Fall die Task mit der maximalen Zykluszeit als letztes übernommen. Die Gesamtzeitdauer hierfür errechnet sich aus der Summe der zur Taskübernahme benötigten vielfachen H-Zykluszeit in 4.20 und der größtmöglichen V-Zykluszeit der Task in 4.21.

Für die Selbst-Konfiguration gilt deshalb:

$$\text{Worst-Case Zeitverhalten} \; = t_{HC} \cdot h + {}_{max}t_{VC} \tag{4.22}$$

4.3. Funktionsweise und Zeitverhalten bei Selbst-Optimierung

In Kapitel 4.1.3 wurde das zeitliche Verhalten der Taskverarbeitung durch den V-Zyklus analysiert. Nun soll untersucht werden, welche Auswirkungen dies auf die Selbst-Optimierung hat. Unter anderem soll geklärt werden, unter welchen Bedingungen bei einer unterbrechungsbehafteten Optimierung (Ausfallzeit der Taskausführung während der Optimierungssequenz) eine qualitative Verbesserung der Taskverarbeitung unter Einhaltung der vorgegebenen Zeitschranken zu erwarten ist. Hierfür wird nun der zeitliche Ablauf der Optimierung genauer betrachtet. Wie bereits in Kapitel 2 beschrieben, erfolgt die Optimierung seitens der Taskverwaltung mittels Taskmigration. Ihre Sequenz unterteilt sich in zwei Phasen.

Innerhalb der ersten Phase erfolgt die erneute Ausschreibung von T_i durch den H-Zyklus des aktuell zugeordneten Kern C_δ, wodurch die Task im System neu angeboten wird. Hierfür wird der gegenwärtig gesendete Übernahme-Suppressor $S_{i\delta}$ unterdrückt, indem der aktuelle Wert auf Null gesetzt wird. Durch den Wegfall des Übernahme-Suppressor, erhöht sich der Eignungswert Em_i auf allen Kernen im Netz. Durch die periodisch wiederkehrende Regelschleife erfolgt die erneute Zuordnung von Task T_i unter der Berücksichtigung aller verfügbaren Kerne. Da eine Migration von T_i einen zusätzlichen Aufwand hinsichtlich des Transfers von Daten aufwirft, sendet Kern C_δ den lokalen Angebots-Akzelerator $A_{i\delta}^{i\delta}$ aus, um seine gegenwärtige Zuordnung zu stärken. Die Stärke dieser Akzeleratoren steht in Relation zum Aufwand, verbunden durch eine Migration der Task auf einen anderen Kern C_γ ($\delta \neq \gamma$). Die erste Phase erfolgt entweder periodisch oder ereignisgesteuert und endet sobald die Übernahme der Task durch C_γ erfolgt ist. Bei einer erneuten Rückübernahme von T_i durch C_δ verweilt die Taskverarbeitung auch nach der Ausschreibung auf dem gleichen Kern und es erfolgt keine Selbst-Optimierung.

Die zweite Phase umfasst die eigentliche Migration der Taskverarbeitung von T_i durch den V-Zyklus. Ausgehend vom Zeitpunkt der Übernahme definiert sich die Dauer der zweiten Phase ausschließlich über die V-Zykluszeit. Nach Ablauf der Zykluszeit befindet sich T_i erneut in der Ausführung auf C_γ. Die Migration der Taskverarbeitung in Phase 2 erfolgt gegenüber der Taskausführung (Ȧ) des V-Zyklus unterbrechungsbehaftet oder unterbrechungsfrei. Es werden daher zwei Arten der Optimierung unterschieden:

unterbrechungsfreie Optimierung: Behandelt den Fall das die Ausführung aus zeitlicher Sicht in Phase 2 fortlaufend ununterbrochen erfolgt. Es herrscht ein nahtloser Übergang zwischen dem Abbruch der aktuellen Taskinstanz auf C_δ und dem Start der migrierten Instanz auf C_γ. Die Migration in Phase 2 erfolgt zeitlich betrachtet passiv.

unterbrechungsbehaftete Optimierung: Behandelt den Fall das die Ausführung nach Übernahme in Phase 1 abgebrochen wird, so dass während Phase 2 eine zeitliche Lücke bis zum erneuten erreichen der Ausführung auf C_γ entsteht. Die Migration erfolgt zeitlich aktiv in Phase 2.

Erfolgt die Optimierung aufgrund der hohen Beschäftigung der Task nicht außerhalb der Bedienzeit von C_δ (aktiver Taskprozess), so entsteht bei einer unterbrechungsbehafteten Optimierung ein Ausfall in der Ausführungssequenz, da der aktive Taskprozess in Phase 2 in den blockierten Zustand wechselt. Erfolgt die Optimierung von T_i hingegen außerhalb der Bedienzeit, so hat Phase 2 unabhängig von beiden Arten keine Auswirkung auf die Ausführungssequenz, weil sich der Taskprozess bereits im blockierten Zustand befindet (schlafender Taskprozess). Dieser Fall ist gegenüber der Analyse des Ausfallverhaltens unerheblich und wird deshalb nicht weiter berücksichtigt. Abbildung 4.6 verdeutlicht den zeitlichen Verlauf der Taskausführung von T_i während einer unterbrechungsfreien und unterbrechungsbehafteten Selbst-Optimierung. Während der gesamten ersten Phase, ausgehend vom

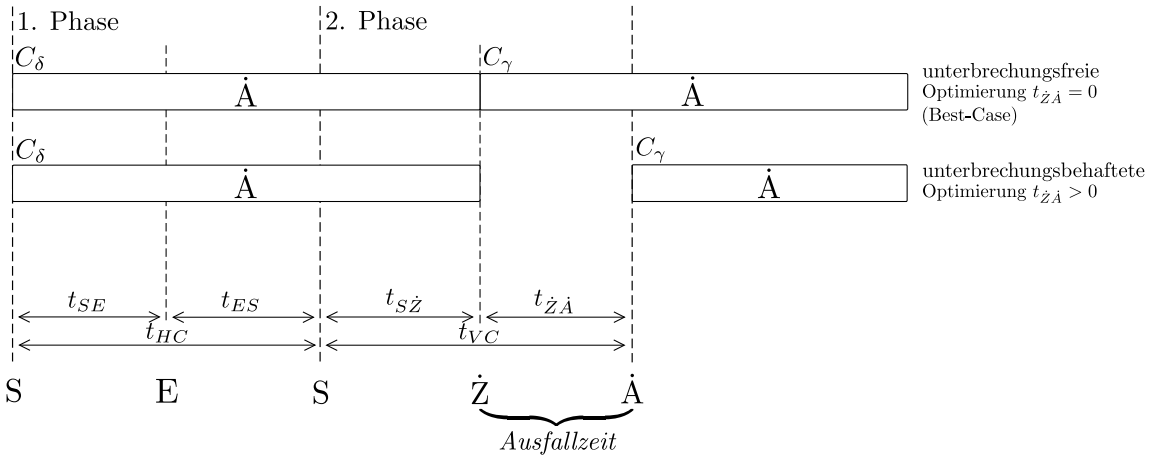


Abbildung 4.6.: Zeitlicher Verlauf der Taskausführung während beider Phasen der Selbst-Optimierung

Zeitpunkt der erneuten Ausschreibung in Schritt S durch den H-Zyklus von C_δ bis zum Beginn des Zustandstransfer am Anfang der zweiten Phase, befindet sich T_i auch weiterhin in der Ausführung. Ab diesem Zeitpunkt erfolgt aufgrund der unterschiedlichen Verfügbarkeit von Komponenten (optionaler Zustand, Beschreibung) ein unterschiedlicher Verlauf bezüglich der weiteren Taskausführung. Im besten anzunehmenden Fall gilt, dass die fortlaufende Ausführung von T_i auch während der Migration unterbrechungsfrei erfolgt, da beide Komponenten zum Zeitpunkt der

Übernahme bereits auf C_γ verfügbar sind und die Wartezeit für den Start der migrierten Taskinstanz mit $t_{\dot{E}\dot{A}} = 0$ entfällt. Dies entspricht einer optimalen lokalen Verfügbarkeit aus Abschnitt 4.1.3.2. Es ist dann mit keinem Ausfall infolge der Migration zu rechnen. Für die Ausfallzeit die sich durch die Vorlaufzeit des V-Zyklus definiert, gilt daher $t_{\dot{Z}\dot{A}} = 0$. In allen anderen Fällen erfolgt eine Unterbrechung der Ausführung, da mindestens eine Komponente zum Zielort transferiert werden muss oder die Wartezeit für den Startvorgang nicht gegen Null konvergiert. Während der Unterbrechung befindet sich die aktuelle Instanz von C_δ im blockierten Zustand, so dass die Ausfallzeit daher mit $t_{\dot{Z}\dot{A}} > t_K$ anzunehmen ist.

In Bezug auf die minimale Vorlaufzeit in Gleichung 4.18 erfolgt die weitere Analyse des zeitlichen Ausfalls unter der Annahme, dass keine unterbrechungsfreie Optimierung aufgrund einer lokalen Verfügbarkeit beider Komponenten gemäß Relation 1 auftritt und die Optimierung wegen der hohen Beschäftigung der Task nicht außerhalb der Bedienzeit (schlafender Taskprozess von T_i) erfolgen kann. Des Weiteren gilt auch weiterhin die Annahme zur maximalen Kommunikationszeit sowie dem minimalen Größenverhältnis der Datenmengen von Komponenten.

4.3.0.4. Analyse und Bewertung des zeitlichen Ausfall

Der zeitliche Ausfall der während einer nicht unterbrechungsfreien Optimierung seitens der Taskausführung entsteht, lässt sich im direkten Vergleich gegenüber der Variante mit keiner Optimierung analysieren, deren Sequenz vorzeitig nach Ablauf der ersten Phase endet. Hierfür wird das aus dem vorigen Abschnitt bekannte Optimierungsszenario von Task T_i mit den beteiligten Kernen C_δ und C_γ erneut aufgegriffen und für beide Varianten anhand der folgenden beiden Sequenzen gegenübergestellt.

Sequenz 1: Behandelt den Fall, dass keine Optimierung von T_i erfolgt, weil die Task im Verlauf der ersten Phase erneut von C_δ übernommen wurde.

Sequenz 2: Behandelt die nicht unterbrechungsfreie Optimierung von T_i, die genau dann eintritt, wenn die Task in der ersten Phase neu von C_γ übernommen wird und in der zweiten Phase eine unterbrechungsbehaftete Migration der Instanz von C_δ nach C_γ erfolgt.

Für die Gegenüberstellung beider Sequenzen wird die Verarbeitung von T_i im System betrachtet, deren zeitlicher Verlauf sich über den lokalen Fortschritt der Taskinstanz ableiten lässt. Der aktuelle Fortschritt einer Taskinstanz definiert die Teilmenge der Tasksequenz, die seit Beginn der Erstausführung im Zeitpunkt t_s bis zum aktuellen Zeitpunkt t_n verarbeitet wurde. Eine Quantifizierung dieser Teilmenge ist durch die Anzahl an Instruktionen möglich, die in dem Zeitraum $[t_s, t_n]$ ausgewertet wurden. Diese wird synchron für jede Taskinstanz gegenüber einem

einheitlichen Quantifizierungszyklus mit Zyklusdauer Δt_q ermittelt[2]. Für den Zeitraum $[t_s, t_n]$ ergibt sich somit eine Referenzfolge aus Quantifizierungszyklen $s, ..., n$, die in Abbildung 4.7 dargestellt ist. Für einen bestimmten Zyklus k in der Folge $[s, n]$ definiert Q_k die Anzahl an Instruktionen, die während dessen Zyklusdauer Δt_q ausgeführt wurden. Dies entspricht der mittleren Verarbeitungsgeschwindigkeit im Zeitraum $[k-1 \cdot \Delta t_q, k \cdot \Delta t_q]$, weshalb Q_k nachfolgend als lokaler Verarbeitungstrend von T_i in Zyklus k bezeichnet wird. Der Gesamtfortschritt der Taskinstanz zum Zeitpunkt t_n kann nach Ablauf des entsprechenden Zyklus n durch Akkumulation der zugehörigen Instruktionssummen $Q_s, ..., Q_n$ über die Funktion F berechnet werden.

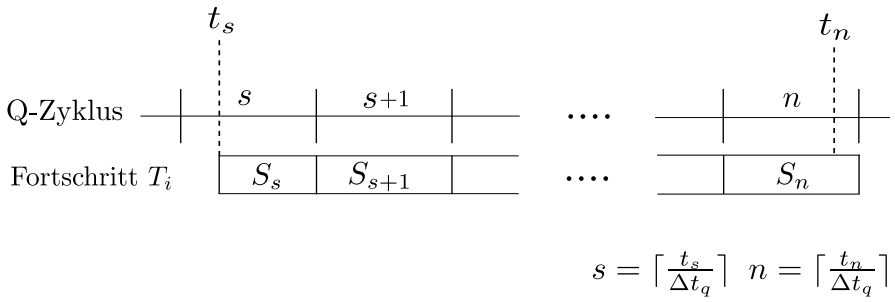


Abbildung 4.7.: Quantifizierung der Taskverarbeitung

$$F(n) = \begin{cases} Q_s + Q_{s+1} + ... + Q_n = \sum_{k=s}^{n} Q_k & : n \geq s \\ 0 & : n < s \end{cases} \tag{4.23}$$

Unter Berücksichtigung der Funktion, die den Fortschritt der Taskinstanz von T_i bestimmt, erfolgt die Analyse des zeitlichen Ausfalls in Sequenz 2 nunmehr grafisch anhand des Schaubildes der Funktion, die sich sich für beide Sequenzen wie folgt differenzieren lässt. Die Funktion F^1 stellt den Verlauf der Taskverarbeitung in Sequenz 1 dar. Die Funktion F^2 den Verlauf in Sequenz 2. Die Graphen beider Funktionen sind in Abbildung 4.8 dargestellt, so dass sich die Taskverarbeitung von T_i in beiden Sequenzen nun wie folgt interpretieren lässt. Unter der Annahme einer uniformen Startkonfiguration des KHS verläuft der Taskfortschritt, ausgehend vom Zeitpunkt der Erstausführung in Zyklus s, bei beiden Sequenzen zunächst annähernd identisch. In Zyklus $\dot{z}$ tritt in Sequenz 2 die Migration von T_i ein, die über den V-Zyklus erfolgt und mit dem Zustandstransfer nach C_γ beginnt. Aufgrund dessen kommt es ab diesem Zeitpunkt in Sequenz 2 zu keinem weiteren Fortschritt der Instanz von T_i, was sich im Schaubild durch eine Stagnation des Graphen von F^2 darstellt und

[2]Für Instanzen, deren Fortschritt sich aufgrund einer formalen Beschreibung nicht anhand von Instruktion bemessen lässt, kann alternativ eine Vergleichsmodellierung herangezogen werden, die beispielsweise die Anzahl verarbeiteter Zustände für jedes Intervall ermittelt.

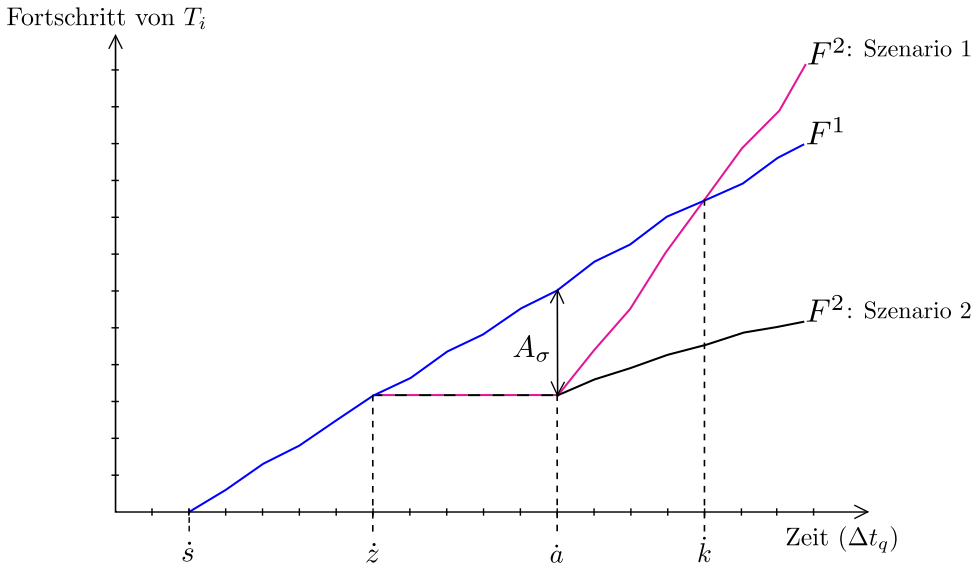


Abbildung 4.8.: Vergleich vs unterbrechungsbehafteter Selbst-Optimierung

den zeitlichen Ausfall repräsentiert. Hingegen läuft die Taskausführung in Sequenz 1 auch nach Ablauf des Zyklus $\dot{z}$ weiter, so dass hier ein fortlaufender Anstieg des Graphen F^1 zu beobachten ist. In Zyklus $\dot{a}$ erreicht der V-Zyklus in Sequenz 2 erneut die Ausführung und der Graph von F^2 beginnt erneut zu steigen.

Der Ausfall der durch die Unterbrechung von T_i während der Migrationsphase in Sequenz 2 entsteht, wird im Schaubild durch den räumlichen Abstand beider Funktionsgraphen in Zyklus $\dot{a}$ ersichtlich. Dieser lässt sich über die Abstandsfunktion A erfassen, die sich aus der Differenz beider Funktionen definiert.

$$A(n) = F^1(n) - F^2(n) \quad , n \geq \dot{z} \tag{4.24}$$

Mittels der Funktion kann der Ausfall nach Ablauf von Zyklus $\dot{a}$ über den zugehörigen Funktionswert $A_{\dot{a}}$ quantifiziert werden, der die Differenz ausgeführter Instruktionen in Sequenz 2 definiert, die als Ausfallsumme A_σ bezeichnet wird.

$$\text{Ausfallsumme} \quad A_\sigma = A_{\dot{a}} = F^1_{\dot{a}} - F^2_{\dot{a}} \tag{4.25}$$

Des Weiteren kann über die Funktion der dynamische Abstandsverlauf zwischen beiden Funktionen verfolgt werden, der den Vorsprung der Taskinstanz in Sequenz 1 misst, der während der Migrationsphase gegenüber Sequenz 2 entsteht. Diesbezüglich lassen sich zwei mögliche Szenarien analysieren, die nach nach Ablauf der Optimierung in Zyklus $\dot{a}$ bei Sequenz 2 auftreten können. Beide Szenarien berücksichtigen einen gegensätzlichen aber kontinuierlichen Verarbeitungstrend von T_i, der sich durch eine höhere oder geringere Anzahl an Instruktionen pro Zyklus gegenüber Sequenz 1 abzeichnet.

Szenario 1: Kompensation der Ausfallsumme
Bei der Kompensation stellt sich nach Ablauf der Optimierung in Zyklus $\dot{a}$ ein gesteigerter Verarbeitungstrend von T_i in Sequenz 2 ein. Dies bedeutet, dass die Anzahl ausgeführter Instruktionen pro Zyklus in Sequenz 2 über der von Sequenz 1 liegt. Es gilt daher das folgende Größenverhältnis zwischen den Trends beider Sequenzen.

$$\text{Szenario 1:} \quad Q_n^2 > Q_n^1 \quad , n > \dot{a} \tag{4.26}$$

Die entstandene Ausfallsumme A_σ wird stetig minimiert und letztlich in Zyklus $\dot{k}$ vollständig kompensiert. Dies ist der Moment an dem Sequenz 2 den aktuellen Taskfortschritt von Sequenz 1 einholt, was sich im Schaubild 4.8 durch den Schnittpunkt beider Funktionsgraphen in Zyklus $\dot{k}$ darstellt. Der Zeitraum der für die Minimierung der Ausfallmenge A_σ verstreicht, wird als Kompensationsphase in Szenario 1 bezeichnet. Nach Erreichen der Kompensation in Zyklus $\dot{k}$ gilt weiterhin ein gesteigerter Verarbeitungstrend, weshalb sich Sequenz 2 in jedem weiteren Zyklus zunehmend gegenüber Sequenz 1 amortisiert. Formal kann dieses Szenario durch die bestimmte Divergenz der Abstandsfunktion gegen $-\infty$ im Intervall $]\dot{a}, \infty[$ ausgedrückt werden.

$$\lim_{n \to \infty} A(n) = -\infty \quad , n > \dot{a}$$

Szenario 2: Stetigkeit der Ausfallsumme
Dieses Szenario umfasst eine gleichbleibende oder verringerte Taskverarbeitung von T_i nach Ablauf der Optimierung in Sequenz 2. Die Anzahl ausgeführter Instruktionen pro Intervall ist entweder gleich oder niedriger gegenüber der in Sequenz 1.

$$\text{Szenario 2:} \quad Q_n^2 \leq Q_n^1 \quad , n > \dot{a} \tag{4.27}$$

Die Ausfallsumme bleibt im weiteren Taskverlauf von T_i dauerhaft erhalten. Eine Annäherung des Taskfortschritt in Sequenz 1 tritt nicht ein. Für die Abstandsfunktion gilt daher ein positiver Wertebereich im Intervall $]\dot{a}, \infty[$.

$$A(n) \leq A_\sigma \quad , n > \dot{a}$$

Das in Szenario 1 beschriebene Kompensationsverhalten in Sequenz 2 zeigt Einfluss auf die Festlegung des Intervalls einer periodischen Selbst-Optimierung von T_i. Wie das Szenario zeigt, entspricht die Kompensationsphase der maximal benötigten Wirkungsdauer, bis sich Sequenz 2 gegenüber Sequenz 1 amortisiert. Demnach sollte eine erneute Optimierung der Task in Szenario 1 frühestens nach Ablauf der Kompensation erfolgen. Dies garantiert, dass der Taskverarbeitung in Sequenz 2 ausreichend Zeit zur Verfügung steht, um den vollen Wirkungsgrad der Optimierung auszuschöpfen. Deshalb wird die Dauer der Kompensationsphase nun genauer untersucht. Hingegen ist bei Erhalt der Ausfallsumme mit keiner Wirkungsdauer

zu rechnen, weshalb Szenario 2 gegenüber dem Zeitverhalten der Optimierung unerheblich ist und nicht weiter betrachtet wird. Die Dauer der Kompensationsphase ist dynamisch und verläuft proportional zum aktuellen Trend von T_i in Sequenz 2. Unter Berücksichtigung des Größenverhältnis 4.26, kann für Szenario 1 die in 4.28 aufgeführte Äquivalenz formuliert werden.

$$\begin{aligned} Q_n^2 > Q_n^1 \quad &\equiv \quad Q_n^2 = Q_n^1 + \Delta Q_n \quad , \Delta Q_n > 0 \\ &\Rightarrow \Delta Q_n = Q_n^2 - Q_n^1 \end{aligned} \tag{4.28}$$

Mittels der obigen Äquivalenz lässt sich der Prozess der Kompensation als eine schrittweise Minimierung der Ausfallmenge darstellen, die in jedem Zyklus durch die Differenz ΔQ_n reduziert wird. Diese stellt den lokalen Zuwachs im Trend von T_i dar, der nach Ablauf der Optimierung in Sequenz 2 erfolgt und die Dauer der Kompensation reguliert. Dies lässt sich durch den Graphen der Abstandsfunktion in Abbildung 4.9 veranschaulichen.

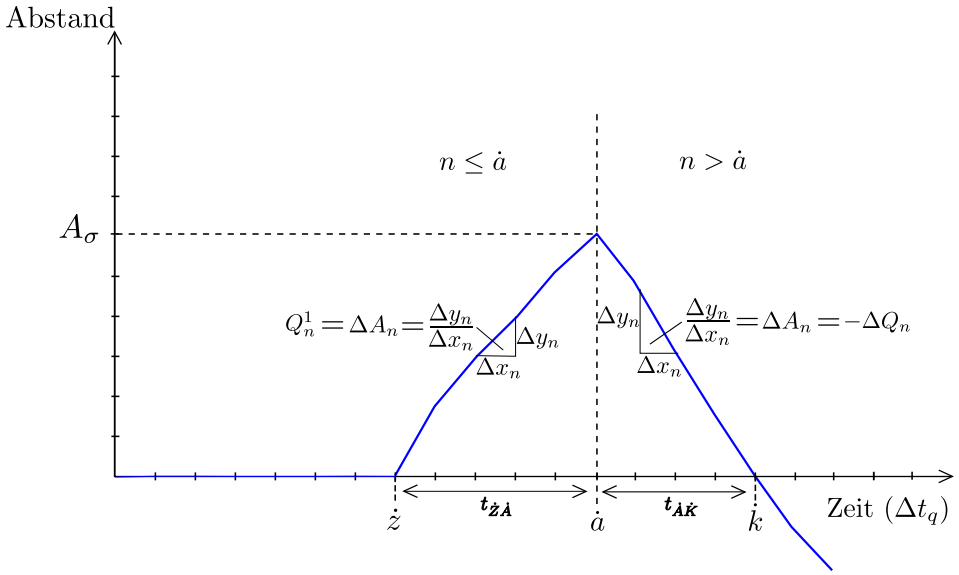


Abbildung 4.9.: Kompensationsverhalten in Szenario 1

Hier entspricht ΔQ_n dem Betrag der mittleren Änderungsrate des Graphen in Zyklus $n > \dot{a}$. Dies kann durch umformen des Differenzenquotient in 4.29 gezeigt werden. Der Zuwachs von Sequenz 2 bestimmt daher das negative Steigungsverhalten der Abstandsfunktion im Intervall $[\dot{a}, \infty[$ und somit den Eintritt des Kompensationszeitpunkt in Zyklus $\dot{k}$ (Schnittpunkt X-Achse).

$$\begin{aligned}\Delta A_n = \frac{\Delta y_n}{\Delta x_n} &= \frac{A_n - A_{n-1}}{n-(n-1)} = (F_n^1 - F_n^2) - (F_{n-1}^1 - F_{n-1}^2) \\ &= (F_n^1 - F_{n-1}^1) - (F_n^2 - F_{n-1}^2) \\ &= \Delta F_n^1 - \Delta F_n^2 \\ &= Q_n^1 - Q_n^2 \\ \\ &\Rightarrow \Delta A_n = -\Delta Q_n\end{aligned} \tag{4.29}$$

Deshalb wird nun untersucht, für welchen durchschnittlichen Zuwachs die Dauer der Kompensationsphase der Ausfallzeit $t_{\dot{Z}\dot{A}}$ entspricht. Ziel ist es, aus diesem Referenzfall eine Gesetzmäßigkeit abzuleiten, die für eine beliebige Optimierung gemäß Szenario 1 die zugehörige Kompensationszeit bestimmt. Der durchschnittliche Zuwachs $\Delta\overline{Q_K}$ von Sequenz 2 während der Kompensationsphase lässt sich über das arithmetische Mittel von ΔQ_n im Intervall $[\dot{a}+1, \dot{k}]$ berechnen.

$$\text{Durchschnittliche Zuwachs} \quad \Delta\overline{Q_K} = \frac{1}{\dot{k}-\dot{a}+1} \sum_{k=\dot{a}+1}^{\dot{k}} \Delta Q_k \tag{4.30}$$

Der durchschnittliche Ausfall $\Delta\overline{Q_A}$ von Sequenz 2 berechnet sich analog über die mittlere Änderungsrate der Abstandsfunktion im Intervall $[\dot{z}, \dot{a}]$, die mittels Umformung aus Gleichung 4.29 dem mittleren Trend von Sequenz 1 entspricht.

$$\begin{aligned}\text{Durchschnittliche Ausfall} \quad \Delta\overline{Q_A} = \frac{A_\sigma}{\dot{a}-\dot{z}} &= \frac{1}{\dot{a}-\dot{z}} \sum_{k=\dot{z}}^{\dot{a}} \Delta A_k \\ &= \frac{1}{\dot{a}-\dot{z}} \sum_{k=\dot{z}}^{\dot{a}} -\Delta Q_k = \frac{1}{\dot{a}-\dot{z}} \sum_{k=\dot{z}}^{\dot{a}} \Delta Q_k^1\end{aligned} \tag{4.31}$$

Unter Berücksichtigung der beiden Mittelwerte aus Gleichung 4.30 und 4.31, kann nun für jede beliebige Optimierung gemäß Szenario 1 (Kompensation der Ausfallsumme) das folgende Kompensationsverhältnis gebildet werden, mit welchem sich die Dauer der Kompensationsphase als vielfaches der Ausfallzeit ausdrücken lässt.

$$\text{Kompensationsverhältnis} \quad c_\sigma = \frac{\Delta\overline{Q_K}}{\Delta\overline{Q_A}}$$

$$\Rightarrow \text{Kompensationszeit} \quad t_{\dot{A}\dot{K}} = c_\sigma \cdot t_{\dot{Z}\dot{A}}$$

Abschließend kann unter Annahme eines geeigneten Kompensationsverhältnises, eine untere Schranke für die Dauer der Kompensationsphase abgeleitet werden. Deshalb wird nun die folgende Annahme erhoben, die den Zuwachs von Sequenz 2 begrenzt und unter Berücksichtigung früherer Qualitätsanalysen der Taskverteilung in [BvRP08] sowie der in Kapitel 6 aufgeführten Messungen erfolgt.

Annahme zum Kompensationsverhältnis: Der Zuwachs im Trend von T_i entspricht maximal dem durchschnittlichen Ausfall in 4.31, so dass die Kompensationszeit mindestens der Ausfallzeit entspricht. Für das Kompensationsverhältnis gilt daher:

$$c_\sigma \geq 1$$

Daraus ergibt sich folgende untere Schranke für die Kompensationszeit, bei einer unterbrechungsbehafteten Optimierung:

$$\text{Minimale Kompensationszeit} \quad t_{\dot{A}\dot{K}} \geq t_K \quad , c_\sigma = 1 \tag{4.32}$$

4.3.0.5. Zusammenfassung des Zeitverhalten bei Selbst-Optimierung

Zusammenfassend kann für das erweiterte Taskverarbeitungsmodell des V-Zyklus das Zeitverhalten bei Selbst-Optimierung angegeben werden. Wie die Untersuchungen zeigen, ist der Aufwand der Sequenz dynamisch und definiert sich über die Migration der Taskverarbeitung in Phase 2. Diese erfolgt mittels des V-Zyklus entweder unterbrechungsfrei oder unterbrechungsbehaftet gegenüber der Taskausführung. Bei einer unterbrechungsfreien Optimierung erfolgt die Migration aufgrund einer lokalen Verfügbarkeit beider Komponenten passiv, so dass der zeitliche Aufwand von Phase 2 vollständig entfällt ($t_{\dot{Z}\dot{A}} = 0$). Folglich beschränkt sich der Aufwand ausschließlich auf das erneute Anbieten der Task im System in Phase 1. Mit der Einschränkung, dass unter allen Kernen in jedem H-Zyklus nur eine Task im System angeboten wird, gilt für die Optimierung die in [vR12] aufgeführte konstante Laufzeit von genau einer H-Zykluszeit (t_{HC}). Bei einer unterbrechungsbehafteten Optimierung erfolgt die Migration aktiv in Phase 2, so dass im schlimmsten Fall eine globale Verfügbarkeit von Komponenten vorliegt, welche die volle Komplexität des V-Zyklus und dessen größtmögliche Vorlaufzeit in Abschnitt 4.17 beansprucht.

Für die Bestimmung der oberen Zeitschranke gilt der Fall, dass das Anbieten von T_i in Phase 1 zeitgleich für alle Tasks im System erfolgt. Dies entspricht dem gleichen Aufwand, der bei der Selbst-Konfiguration entsteht. Deshalb ist das Zeitverhalten im Worst-Case Fall äquivalent zu dem in Gleichung 4.22.

Für die Selbst-Optimierung aller Tasks gilt daher:

$$\text{Worst-Case Zeitverhalten} \; = t_{HC} \cdot h + {}_{max}t_{VC} \tag{4.33}$$

Um das Zeitverhalten zu verbessern, erfolgt im Normalbetrieb keine zeitgleiche Optimierung aller Tasks. Deshalb bietet ein Kern nur dann eine Task zur Übernahme an, wenn sich im aktuellen H-Zyklus keine weiteren Tasks in der Optimierung befinden. Daraus entsteht eine Folge von Optimierungssequenzen, die sich durch das Aussenden des leeren Übernahme-Suppressors in Phase 1 synchronisiert. Wird ein solcher Suppressor von einem Kern empfangen, so wartet dieser so lange mit dem Anbieten seiner Task ab, bis die aktuelle Sequenz beendet ist. Somit beschränkt sich das Zeitverhalten auf die Optimierung einer einzelnen Task pro H-Zyklus. Die Zeitdauer in Phase 1 ist dann konstant ($h = 1$). Hingegen ist der Aufwand in Phase 2 weiterhin taskabhängig. Für die Laufzeit gilt dann die Summe, die sich aus beiden Zykluszeiten akkumuliert.

Für die Selbst-Optimierung einer einzelnen Task gilt daher:

$$\text{Worst-Case Zeitverhalten } = t_{HC} + {}_{max}t_{VC} \geq 4 \cdot t_K \qquad |t_{HC} \geq 2t_K, t_{VC} \geq 2t_K \tag{4.34}$$

Für die Wahl des kleinstmöglichen Optimierungsintervall einer Task, ist die größtmögliche Dauer der Sequenz in Phase 2 bindend. Andernfalls bestünde bei einer zyklischen Optimierung die Möglichkeit der gegenseitigen Überschneidung mit der darauffolgenden Sequenz. Deshalb sollte der zeitliche Abstand zwischen zwei Sequenzen mindestens so groß sein, wie die maximale V-Zykluszeit der Task. Des Weiteren gilt, dass im Falle einer unterbrechungsbehafteten Optimierung eine zusätzliche Wartezeit für die Kompensation der entstandenen Ausfallmenge anfällt, bevor die Optimierung ihre vollen Wirkungsgrad entfaltet. Für die minimale Intervallänge einer Optimierung gilt daher die Zeitspanne, die sich aus maximaler V-Zykluszeit und Kompensationszeit akkumuliert.

Für das Minimale Optimierungsintervall einer Task gilt:

$$\text{Min. Optimierungsintervall } \quad \Delta t_{Opt} \geq t_{VC} + t_{\dot{AK}} \geq 3 \cdot t_K \qquad |t_{VC} \geq 2t_K, t_{\dot{AK}} \geq t_K \tag{4.35}$$

4.4. Funktionsweise und Zeitverhalten bei Selbst-Heilung

Die Selbst-Heilung stellt eine weitere Kerneigenschaft des Systems betreffend der Zuverlässigkeit gegenüber den in Kapitel 2 klassifizierten Ausfällen und Fehlern von Systemkomponenten dar. Aufgrund des unterschiedlichen Verhaltens ist eine spezifische Selbst-Heilung notwendig, deren Funktionsweise und Zeitverhalten nun genauer erörtert werden soll. Für die Behandlung von Ausfällen und Fehlern werden daher zwei verschiedene Varianten der Selbst-Heilung verwendet. Diese sind:

1. Reaktive Selbst-Heilung
2. Proaktive Selbst-Heilung

Die reaktive Selbst-Heilung in 1. wird bei Ausfällen von Systemkomponenten verwendet, die aufgrund der wechselseitigen Relation zwischen Komponenten eine erneute Verteilung der durch den Ausfall betroffenen Taskmenge impliziert. Im schlimmsten Fall bedeutet dies, dass bei einem Kernausfall dessen m_{max} Tasks ebenfalls ausfallen. Folglich müssen diese zeitnah umverteilt werden, damit das System erneut funktionsfähig ist.

Die Funktionsweise der reaktiven Heilung ist dabei wie folgt: Durch den Ausfall der Tasks entfällt auch das Aussenden derer Hormone. Der Wegfall dieser Hormone führt dazu, dass die negative Rückkopplung die auf den H-Zyklus durch Suppressor-Einfluss einwirkt, zum erliegen kommt. Die H-Zyklen der noch funktionierenden Kerne reagieren auf diese Veränderung indem Sie Ihre erhöhten Eignungswerte für die betroffenen Tasks erneut austauschen und auf Basis dieser übernehmen. Durch die Umverteilung der Tasks, werden die V-Zyklen neu gestartet. Im schlimmsten Fall beinhaltet ein Neustart des V-Zyklus den Transfer beider Taskkomponenten, die ausfallbedingt von unterschiedlichen Kernen aus gesendet werden. Erlaubt der funktionale Zustand des ausgefallenen Kerns den Zugriff und das Senden von Daten der ausgefallenen Taskinstanz, so werden die Komponenten direkt von diesem aus zum Zielort transferiert. Andernfalls erfolgt der Transfer durch den Kern, der die Rolle der Sicherung von Taskkomponenten im Netz übernimmt und auf zuletzt gesicherte Daten der ausgefallenen Taskinstanzen zurückgreift. Nachdem alle V-Zyklen durch dieses redundante Sicherungskonzept gestartet und durchlaufen wurden, befinden sich alle ausgefallenen Tasks erneut in der Ausführung und die Selbst-Heilung ist beendet. Durch das Umverteilen und Neustarten ausgefallener Tasks, erfolgt stets eine Heilung des Systems solange genügend funktionsfähige Kerne vorhanden sind. Obwohl die Funktionsweise dem Prinzip der Selbst-Konfiguration in 4.3 entspricht, ist das Zeitverhalten gegenüber dieser unterschiedlich. Dies lässt sich dadurch erklären, dass die Kontrollzyklen nach dem Ausfall der Tasks eine gewisse Zeit benötigen um dies zu detektieren. Es ist daher bei beiden Abläufen mit zusätzlichen Verzögerungszeiten zu rechnen. Diese werden nun betrachtet.

Hormon-Verfallszeit: Die Taskübernahme verzögert sich, da die H-Zyklen der noch funktionsfähigen Kerne zunächst auf die zuletzt empfangenen Hormonwerte zurückgreifen, sofern der weitere Empfang besagter Hormone ausbleibt. Der Wegfall der Hormone wirkt sich daher erst nach einer bestimmten Zeit aus. Diese wird als Verfallszeit eines Hormon bezeichnet und definiert, wie viele H-Zyklen ein Hormon von einer bestimmten Quelle ohne erneutes Senden im Speicher eines Kern erhalten bleibt. Die Zeitdauer verschlechtert sich daher um eine konstante Anzahl von a H-Zyklen die sich nach Art der Betriebsweise des KHS richtet.

Task-Antwortzeit: Die Taskausführung verzögert sich ebenfalls, weil die V-Zyklen nicht zwingend innerhalb der maximalen Reaktionszeit aus Abschnitt 4.9 neu gestartet werden. Nach Empfang der Übernahme-Suppressoren wartet der Kern, der die Sicherung von Taskkomponenten im Netz übernimmt, noch eine bestimmte Zeit bis dieser mit dem Transfer gesicherter Komponenten beginnt. Diese Wartezeit wird als Task-Antwortzeit bezeichnet, die der ausgefallen Taskinstanz eingeräumt wird, um selbst auf die Neuzuordnung zu reagieren und Komponenten selbst zum Zielort zu transferieren oder veränderte Zustandsdaten zu sichern. Die Zeitdauer die bis zur Ausführung aller Tasks verstreicht, verschlechtert sich daher im schlimmsten Fall um die statische Antwortzeit, die durch ein vielfaches b der maximalen Reaktionszeit des V-Zyklus passend für das jeweilige System reguliert wird.

Für die reaktive Selbst-Heilung aller Tasks gilt deshalb:

$$\text{Worst-Case Zeitverhalten } = t_{HC} \cdot (a + h) + {}_{max}t_{VC} + b \cdot {}_{max}t_{S\dot{Z}} \tag{4.36}$$

a := Anzahl der H-Zyklen bis Ausfall detektiert wird (Hormon-Verfallszeit)
b := Vielfache Reaktionszeit bis V-Zyklus gestartet wird (Task-Antwortzeit)

Die proaktive Selbst-Heilung in 2. wird bei permanenten oder transienten Fehlern von Systemkomponenten verwendet. Im Gegensatz zur reaktiven Variante ist, aufgrund des untergeordneten Verhaltens gegenüber Ausfällen, mit keiner unmittelbaren Umverteilung der durch den Fehler betroffenen Taskmenge zu rechnen. Die resultierenden Taskfehler werden optional behoben und durch lokale Monitoring-Einheiten detektiert und überwacht. Im schlimmsten Fall werden die m_{max} Tasks des jeweiligen Kerns mit zunehmender Fehleranzahl an die umliegenden Kerne abgegeben. Die Heilung erfolgt daher präventiv, um den weiteren Ausfall der Tasks zu verhindern.

Die detaillierte Funktionsweise ist wie folgt: Nach Auftreten des Taskfehler, wird die Verarbeitung der betroffenen Taskinstanz durch den V-Zyklus weiter fortgesetzt. Je nach Fehlerart werden Gegenmaßnahmen mittels Bypass oder Neustart der Taskausführung durchgeführt, um die negativen Auswirkungen zu beheben. Diese Maßnahmen umfassen aber keinen Transfer von Taskkomponenten, weshalb sich der Aufwand im V-Zyklus auf das Starten der Taskausführung beschränkt. Durch die lokale Monitoring-Einheit des zugehörigen Kern, erfolgt die Berechnung und Ausschüttung eines lokalen Fehler-Suppressor, der die negative Rückkopplung auf den H-Zyklus verstärkt. Die Wirkung des Suppressor kann unter Berücksichtigung der Selbst-Optimierung von Tasks nun folgend erklärt werden. Spätestens nach Ablauf des Optimierungsintervalls wird die Task durch den Kern erneut zur Übernahme angeboten. Dessen Eignung ist nun aufgrund des zusätzlichen Fehler-Suppressor herabgesetzt, so dass diese von einem anderen Kern übernommen werden kann. Ob oder

wie schnell eine Migration der fehlerbehafteten Taskinstanz erfolgt, wird über die Stärke des Fehler-Suppressors reguliert. Für transiente Fehler erfolgt eine stufenweise Erhöhung des Suppressorwerts. Beim Auftreten eines permanenten Fehlers wird direkt der maximale Wert des Suppressors gesendet. erfolgt die Zunahme bei permanentem Fehler abrupt, weshalb die Migration der Task bereits zur nächsten Optimierung erfolgen kann. Des Weiteren können neben den Fehler-Suppressoren auch weitere Suppressoren für vitale Parameter, wie beispielsweise dem Gesundheitszustand von Kernen, umgesetzt werden.

Die Zeitdauer die für die proaktive Heilung des Systems verstreicht, wird schlimmsten Falls durch die zeitgleiche Migration aller fehlerbehafteten Taskinstanzen dominiert. Anders als bei einer ausfallbedingten Umverteilung, treten keine weiteren Verzögerungszeiten gegenüber den Kontrollzyklen auf, so dass das Laufzeitverhalten der Selbst-Optimierung entspricht.

Für die proaktive Selbst-Heilung aller Tasks gilt deshalb:

$$\text{Worst-Case Zeitverhalten } = t_{HC} \cdot h + {}_{max}t_{VC} \tag{4.37}$$

5. System Implementierungen

Für die Evaluation des Taskverarbeitungssystem auf realer SoC-Zielplattform, wurden verschiedenen Implementierungen erstellt, die in den folgenden Abschnitten vorgestellt und erläutert werden.

5.1. KHS-Hardwareentwurf zum Einsatz auf Multi-Core Systemen

Zur Evaluation in realer Multi-Core Architektur, wurde ein Hardwareentwurf des KHS erstellt, der erstmals in Veröffentlichung [BPB12] vorgestellt wurde und im Folgenden auch als AHS*LiHaDe* (Artificial Hormone System - Lightweight Hardware Described) bezeichnet wird. Es handelt sich hierbei um ein eigenständiges Schaltwerk, welches die Funktionalität einer lokalen KHS-Instanz durch den Einsatz eigener Funktionseinheiten zur Verfügung stellt, die peripher zu den Schaltkreisen von Kernen integriert wird. Im Gegensatz zur ursprünglichen Umsetzung als Bibliothek, die in Veröffentlichung [vRB10] vorgestellt wurde, beansprucht das AHS*LiHaDe* zur Auswertung der hormongeregelten Taskverwaltung somit nicht die Rechen-Ressourcen des verwalteten Kern, sondern nur die eigenen. So soll sichergestellt werden, dass die Taskverwaltung auch von Kernen genutzt werden kann, deren Instruktionssatz-Architektur eine Ausführung des KHS als Bibliothek nicht unterstützt. Damit das AHS*LiHaDe* auch als Referenz für einen realen Schaltungsentwurf genutzt werden kann, ist der Entwurf synthesefähig in VHDL modelliert. Dies soll gewährleisten, dass einerseits die Portierung auf rekonfigurierbare Plattformen zu Evaluationszwecken wie in Kapitel 6 und andererseits eine erste Flächenabschätzung bezüglich einer Fertigung als IC möglich ist.

5.1.1. Zeitverhalten des KHS-Hardwareentwurf

Bevor detailliert auf die Struktur des AHS*LiHaDe* eingegangenen wird, erfolgt die Betrachtung des realen Zeitmodell. Im Gegensatz zur AHS*lib* verwendet die Hardwarebeschreibung ausschließlich eine zeitlich selbst-synchronisierende Durchführung von H-Zyklen auf allen Kernen. Durch dieses Verhalten kann auf die Verwendung eines komplexen Hormonspeichers verzichtet werden. Des Weiteren ermöglicht es die Betrachtung eines gemeinsamen Zeitverlaufs einer bestimmten Iteration des H-Zyklus auf allen Kernen. Der Mechanismus der zur Synchronisation von H-Zyklen

im System verwendet wird, wurde ebenfalls in Veröffentlichung [BPB12] gezeigt und wird nachfolgend in adaptierter Form für das AHS*LiHaDe* erläutert.

Für die Betrachtung des synchronen Zeitmodells gilt die folgende Präzisierung gegenüber den H-Zyklus behafteten Wartezeiten t_{SE} und t_{ES}, sowie der größtmöglichen Kommunikationszeit im Netz.

Präzisierung der Kommunikationszeit: Aufgrund der Vielzahl an Hormonen, die zu Beginn des H-Zyklus von jedem Kern ausgesendet werden, umfasst die größtmögliche Kommunikationszeit t_K den Zeitraum der für das Senden und Empfangen der größtmöglichen Hormonmenge eines Kerns bei maximaler Netz-Distanz innerhalb der Sendephase des H-Zyklus benötigt wird.

Präzisierung der Wartezeit $\mathbf{t_{SE}}$: Berücksichtigt den Zeitraum der für das Senden der eigenen Hormone, sowie für den Empfang der Fremd-Hormone von allen anderen Kernen innerhalb des H-Zyklus benötigt wird. Das Senden der eigenen Hormone berücksichtigt neben dem eigentlichen Transfer auch deren Berechnungsaufwand (Hormonwerte).

Präzisierung der Wartezeit $\mathbf{t_{ES}}$: Umfasst den Zeitraum der für die Entscheidungsfindung im H-Zyklus benötigt wird, plus der verbleibenden Ruhezeit die bis zum Beginn der erneuten Sendephase im Folgezyklus verstreicht. Der Berechnungsaufwand für die Entscheidung sowie lokale Schwankungen innerhalb der vorhergehenden Zeit t_{SE} (Varianz von Zeitgebereinheiten etc.) werden in der Wartezeit durch einen Jitter-Anteil Δt_{SE} berücksichtigt.

Ausgehend vom initialen Startzeitpunkt ($t = 0$) des Systems, befinden sich die H-Zyklen der lokalen KHS-Instanzen auf allen Kernen zunächst im Ruhemodus. Der Erststart eines H-Zyklus erfolgt dann entweder nach Ablauf einer initialen Wartezeit oder durch den Empfang von Hormonen, die von früher gestarteten KHS-Instanzen ausgesendet wurden. Am leichtesten lässt sich dieses Verhalten zwischen dem zuerst gestarteten H-Zyklus des frühesten Kern C_γ und dem zuletzt gestarteten H-Zyklus des zeit-chronologisch spätesten Kern C_δ im Netz betrachten. Abbildung 5.1 zeigt das zugehörige Zeitdiagramm, welches die folgenden nummerierten Schritte berücksichtigt.

0. Der früheste Kern C_γ im Netz, startet seinen H-Zyklus nach Ablauf der initialen Wartezeit $t_{SE} + t_{ES}$, die der vollen H-Zykluszeit t_{HC} entspricht. Dieses Verhalten wird durch den Synchronisationsmechanismus definiert, der den Erststart eines H-Zyklus entweder nach Ablauf der initialen Wartezeit oder beim Empfang von Hormonen vorsieht, die von H-Zyklen früher gestarteter Kerne ausgesendet wurden. Letzteres ist in diesem Fall nicht möglich, da C_γ der früheste Kern im Netz ist und dieser nach Ablauf der initialen Wartezeit sei-

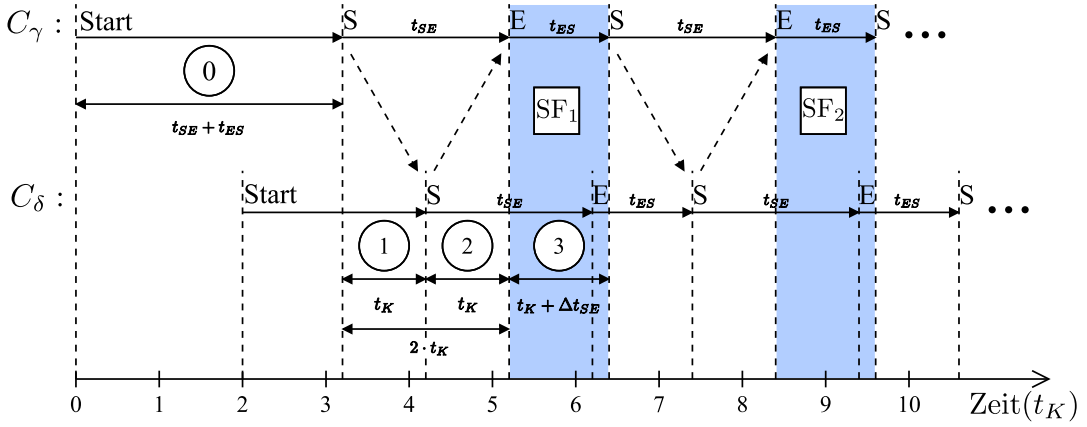


Abbildung 5.1.: Realer Zeitverlauf von H-Zyklen bei synchroner Betriebsweise

nen H-Zyklus vor allen anderen startet. Da es sich bei diesem Schritt um einen Vorgang handelt, der nur bei dem Erststart eines H-Zyklus durchgeführt wird, ist die zugehörige Zeitspanne nicht Bestandteil der regulären H-Zykluszeit t_{HC}.

1. Der H-Zyklus des zeit-chronologisch spätesten Kern C_δ im Netz, wird durch den Empfang von Hormonen aus dem Ruhemodus aufgeweckt und gestartet. Diese wurden durch den früheren H-Zyklus von C_γ ausgesendet, der nach Ablauf seiner initialen Wartezeit in Schritt (0.) gestartet ist. Der Empfang dieser Hormone erfolgt bei größtmöglicher Netz-Distanz zwischen beiden Kernen spätestens nach Ablauf der maximalen Kommunikationszeit t_K. Alle anderen Kerne starten ihre H-Zyklen ebenfalls innerhalb dieser Zeitspanne, da entweder deren Wartezeit aus Schritt (0.) ausläuft oder diese ebenfalls durch die eintreffenden Hormone von C_δ aufgeweckt werden.

2. Aufgrund des in Schritt 1. aufgeführten, größtmöglichen Zeitversatzes (t_K) der zwischen dem Start des frühesten und spätesten H-Zyklus gilt, muss der früheste Kern C_γ nach dem Senden seiner Hormone mindestens das doppelte der maximalen Kommunikationszeit t_K abwarten, um die später ausgesendeten Hormone des zeitlich letzten Kern C_δ noch rechtzeitig zu empfangen, bevor C_γ seine Entscheidung trifft. Für die minimale Wartezeit zwischen Senden und Entscheiden des H-Zyklus gilt deshalb auch bei synchroner Betriebsweise die untere Schranke der doppelten maximalen Kommunikationszeit mit $t_{SE} > 2t_K$. Dies garantiert, dass jeder Kern im Netz stets die Hormone aller anderen empfängt, bevor dieser mit der Entscheidung beginnt.

3. Nach der Entscheidung wartet C_γ noch solange mit dem Start des Folgezyklus ab, bis auch C_δ sicher seine Entscheidung getroffen hat. Dieses Verhalten gehört ebenfalls zum Synchronisationsmechanismus und garantiert, dass alle Kerne

die nach C_γ gestartet wurden, sich korrekt in den Folgezyklus synchronisieren, nachdem diese ihre Entscheidungen getroffen haben. Der Folgestart eines H-Zyklus erfolgt deshalb entweder erneut beim Empfang von Hormonen, die aus Folgezyklen früher gestarteter Kerne ausgesendet wurden, oder spätestens nach Ablauf der Wartezeit t_{ES}. Für diese gilt analog der größtmögliche Zeitversatz t_K zwischen zwei Kernen in Schritt 2. Dieser wird um einen zusätzlichen Jitter-Anteil Δt_{SE} ergänzt, der einerseits den Zeitaufwand für die Entscheidungsfindung berücksichtigt und andererseits lokale Schwankungen innerhalb der Zeitspanne t_{SE} kompensiert. Für die minimale Wartezeit zwischen Entscheiden (im aktuellen Zyklus) und Senden (im Folgezyklus) gilt deshalb die maximale Kommunikationszeit plus Jitter mit $t_{ES} \geq t_K + \Delta t_{SE}$.

Die minimale H-Zykluszeit bei synchroner Betriebsweise des KHS, setzt sich aus den aufgeführten Zeitspannen der Schritte 1., 2. und 3. zusammen.

Für die minimale H-Zykluszeit mit Synchronisation gilt deshalb:

$$\text{Minimale H-Zykluszeit } t_{HC} \geq 3t_K + \Delta t_{SE} \tag{5.1}$$

Neben Einhaltung der minimalen H-Zykluszeit, ist für die korrekte Funktionsweise der Selbstsynchronisation ein verzögertes Entscheidungsverhalten bei Erststart von H-Zyklen zu berücksichtigen. Um diesen Sachverhalt genauer zu erläutern, folgt die erneute Betrachtung des Falls, dass ein beliebiger Kern C_ϵ im Netz seinen H-Zyklus nach dem Empfang von Hormonen startet. Die Annahme, dass es sich dabei um die ersten Hormone des zeitlich frühesten Kern C_δ handelt, gilt hier nur, sofern C_ϵ seine initiale Wartezeit $t_{SE} + t_{ES}$ startet, bevor die frühere von C_δ in Schritt 0. ausläuft. Dieses Verhalten ist aber nur bei Systemstart zum Zeitpunkt $t = 0$ gegeben. Für den Fall das C_ϵ zu einem späteren Zeitpunkt in das bereits eingeschwungene KHS integriert werden muss, gilt die Annahme nur unter der Voraussetzung, dass C_ϵ genau dann eintritt, wenn sich C_γ in der Wartezeit t_{ES} befindet. In diesem Fall kann sich C_ϵ sicher sein, dass es sich bei den eintreffenden Hormonen tatsächlich um die von C_γ handelt. Deshalb wird die Wartezeit t_{ES} von C_γ als Synchronisationsfenster SF_i des aktuellen H-Zyklus i bezeichnet und ist in Abbildung 5.1 durch den wiederkehrenden, blaugefärbten Bereich dargestellt (SF_1, SF_2 etc.). Innerhalb dieses Zeitfensters werden von allen Kernen im Netz keine Hormone gesendet, was C_ϵ die Synchronisation in den Folgezyklus ermöglicht. Erfolgt der Eintritt von C_ϵ jedoch zum Zeitpunkt während sich C_γ in seiner Wartezeit t_{SE} befindet, so kann sich C_ϵ nicht sicher sein dessen erste Hormone verpasst zu haben. Die obige Annahme ist in diesem Fall nicht gültig, weshalb C_ϵ bei Erststart eine gewisse Anzahl an H-Zyklen abwarten muss, bevor dieser seine erste Entscheidung treffen darf. Andernfalls besteht die Gefahr, dass C_ϵ seine Entscheidung auf Basis unvollständig empfangener Hormone vollzieht. Durch das Synchronisationsfenster mit $t_{ES} > t_K + \Delta t_{SE}$ wird sichergestellt, das sich die Wartezeit t_{ES} von C_ϵ nach mehrmaliger Iteration des H-Zyklus schrittweise in den Zeitbereich des Synchronisationsfensters verschiebt. Die

genaue Anzahl an H-Zyklen bis zum Eintritt der Synchronisation von C_ϵ, lässt sich über das folgende Verhältnis bestimmen:

$$\text{Worst-Case Synchronisations-Wartezyklen } S_{syn} = \left\lceil \frac{t_{SE}}{t_{ES}} \right\rceil \tag{5.2}$$

Das obige Wort-Case Verhältnis gibt an, wie viele H-Zyklen Kern C_ϵ mit seiner ersten Entscheidung im schlimmsten Fall abwarten muss, um sicher zu gehen das seine Entscheidung auf Basis konsistenter Hormone erfolgt. Da C_ϵ nicht wissen kann, ob dessen Eintritt während des Systemstart oder nachträglich erfolgt, gilt die Anzahl von S_{syn} Wartezyklen verbindlich als untere Schranke für alle Kerne, deren Start durch den Empfang von Hormonen ausgelöst wird. Für die minimale H-Zykluszeit ergibt sich unter Berücksichtigung der Zeitbedingung für t_{ES} und t_{SE} aus Gleichung 5.1 ein Verhältnis von $2:1$ ($S_{syn} = 2$). Dies bedeutet, dass C_ϵ bei Erststart zwei H-Zyklen mit abwartet d.h. erfolgt der Eintritt in H-Zyklus i, so beginnt C_ϵ ab dem übernächsten H-Zyklus $i + 2$ mit der Entscheidung.

Das adaptierte Zeitmodell zeigt keine nennenswerten Auswirkungen auf die in Kapitel 4.1 vorgestellten Worst-Case Laufzeiten von Selbst-X Eigenschaften. Durch das verzögerte Entscheidungsverhalten gilt gegenüber der Selbst-Konfiguration schlimmsten Falls eine zeitliche Abweichung um S_{syn} H-Zyklen. Des Weiteren gilt durch den Verzicht auf einen Hormonspeicher bei Selbst-Heilung eine Hormon-Verfallszeit mit $a = 1$ H-Zyklen.

5.1.2. KHS-Hardwareentwurf als VHDL-Modell

Um den Hardwareentwurf in einem realen SoC verwenden zu können, wurde das AHS*LiHaDe* als VHDL-Modell erstellt, dessen Beschreibung entsprechend den funktionalen Anforderungen des KHS auf Algorithmischer- und Register-Transfer-Ebene vollzogen wurde. Hierbei wurde auf die Verwendung synthesefähiger Strukturen und Konstrukte des VHDL-Standards zurückgegriffenen, um eine spätere Transformation des Modells auf tieferliegende Entwurfsebenen mittels Syntheseverfahren zu gewährleisten. Um das KHS entsprechend dem funktionalen Umfang der ursprünglichen AHS*lib* auf Hardwareebene umzusetzen, wurde das gesamte Verhalten in verschiedene Hardware-Funktionseinheiten abstrahiert, die im Modell durch sogenannte Entwurfseinheiten abgebildet werden. Die Entwurfseinheiten modellieren die eigentlichen Schaltungskomponenten (Schaltnetze, Schaltwerke) des AHS*LiHaDe*, die analog zu Klassen der Bibliothek, instantiiert und mit anderen Einheiten vernetzt werden. Jede Funktionseinheit wird durch mindestens eine Einheit modelliert, die sich wiederum aus weiteren Sub-Einheiten zusammensetzen kann. Durch die Vernetzung aller Einheiten des Modells, entsteht der gesamte Hardwareentwurf einer lokalen KHS-Instanz, deren Funktionsumfang gleichwertig gegenüber einer Instanz der Bibliothek ist. Durch die Modellierung unterhalb der Systemebene, ist die Umsetzung jedoch in vielen Bereichen gegenüber der Bibliothek stark unterschiedlich.

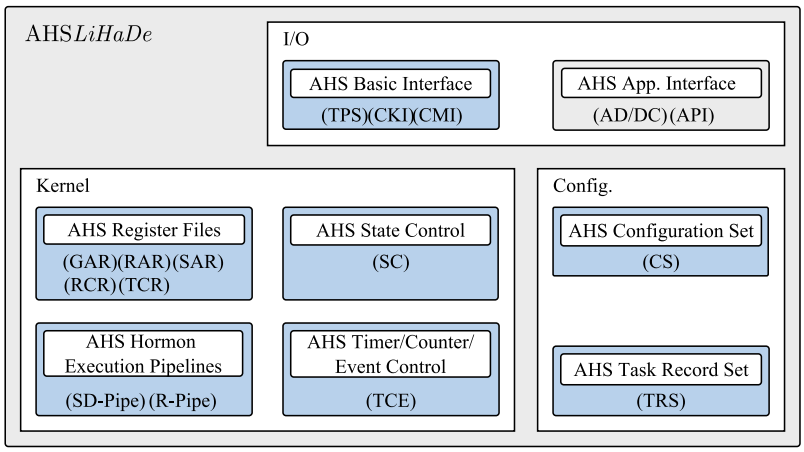


Abbildung 5.2.: Struktur des KHS-Hardwareentwurf

Dies gilt beispielsweise bei der Kommunikation von Hormonen. Hier konnten im Modell weitestgehend viele Prozesse, die auf Systemebene benötigt werden, entweder zusammengefasst, oder gar ganz vernachlässigt werden. Abbildung 5.2 zeigt die Struktur des VHDL-Modells. Neben den primären Entwurfseinheiten (blau gefärbt), die das Verhalten von Funktionseinheit abbilden, umfasst das Modell auch sekundäre Einheiten (gelb gefärbt). Diese dienen den Primären als Container (Pakete) für globale Strukturen, wie beispielsweise die Definition und Deklarationen von Konstanten (Parametern), Typen (Records) oder mehrfach genutzten Funktionen. Des Weiteren umfasst die Struktur applikationsspezifische Einheiten (grün gefärbt), die entsprechend den Anforderungen des Systems (OS-API, Mixed-Signal) optional eingebunden werden. Im wesentlichen lassen sich alle Einheiten des Modells, den drei unterschiedlichen Bereichen Kernel, Konfiguration und IO zuordnen. Ausgehend von der funktionalen Ebene, stellen die primären Einheiten, die den sequentiellen Ablauf des H-Zyklus modellieren, den Kernel dar. Die sekundären Einheiten werden der Konfiguration des Modells zugeordnet, da diese lediglich einzelne Strukturen und keine konkrete Funktionseinheit darstellen. Zur Ein-/Ausgabe (IO) gehören die Einheiten, die Schnittstellenanbindungen (Monitoring Taskverarbeitung etc.) gegenüber dem System oder anderen Anwendungen ermöglichen. Dies ist der einzige Teil des Modells, der gegebenenfalls an die Systemumgebung angepasst werden muss, sofern die vordefinierten Schnittstellen unzureichend sind. Im Nachfolgenden werden nur die wichtigsten zentralen Entwurfseinheiten der Modell-Struktur vorgestellt und in ihrer Funktion erläutert.

AHS State Control (SC) Die SC-Einheit beschreibt die zentrale Kontrollogik, die für die Steuerung und Überwachung von KHS-Vorgängen innerhalb der primären Einheiten verantwortlich ist. Die Funktion umfasst daher im Wesentlichen die Ausgabe von Mikrobefehlen, die für die eigentliche Verarbeitung der sequentiellen Abläufe

des H-Zyklus in den primären Einheiten benötigt werden. Dies wird im Inneren durch eine Finite State Machine (FSM) mit 3 Zuständen verwaltet, die nachfolgend erläutert werden.

0. **Sleep&Standby** Definiert den initialen Ruhezustand der Instanz, der nach Systemstart eingenommen wird und solange aktiv bleibt, bis entweder die initiale Wartezeit t_{HC} verstrichen ist oder ein fremdes Hormon empfangen wurde. Anschließend wechselt die Instanz in den regulären Folgezustand 1. (Send&Receive). In Ausnahmefällen kann die Instanz auch nach Ablauf der initialen Wartezeit in diesem Zustand verweilen oder in diesen zurückversetzt werden. Hierzu zählen Ereignisse, die durch den Ausfall von Systemkomponenten (Kern, Betriebssystem etc.) ausgelöst werden. Des Weiteren ist ein Rücksetzen durch die Eingabe expliziter Steuerbefehle über die vordefinierten Schnittstellen (API, CKI etc.) möglich. Erfolgt das Rücksetzen aus den Zuständen 1. oder 2, so werden alle aktuell gespeicherten Hormonwerte (Hormonsummen, Eignungswerte) des Zyklus in den Sub-Einheiten der *AHS Register Files* auf den initialen Wert zurückgesetzt.

1. **Send&Receive** Dieser Zustand repräsentiert den ersten Teil des H-Zyklus. Hier erfolgt das Senden und Empfangen von Hormonen. Da die SC-Einheit lediglich die Steuerlogik der Instanz umfasst, erfolgt der eigentliche Sende- und Empfangsvorgang in den Sub-Einheiten der *AHS Execution Pipelines*. Nach Ablauf der vordefinierten Zeitspanne t_{SE} sind alle Hormone verarbeitet und die Instanz wechselt in den regulären Folgezustand 2. (Decide&Sync). In Ausnahmefällen erfolgt nach Ablauf der Zeit t_{SE} ein Rücksetzen in den initialen Zustand 0.

2. **Decide&Sync** In diesem Zustand erfolgt der zweite Teil des H-Zyklus. Nach Ablauf von maximal S_{syn} Wartezyklen erfolgt bei Eintritt des Zustands der Entscheidungsprozess durch die AHS Execution Pipelines. Anschließend geht die Instanz in den vorangegangenen Zustand 1. über, wenn die vollständige Zeitspanne t_{ES} verstrichen ist oder ein Hormon eines früher gestarteten H-Zyklus empfangen wurde. In Ausnahmefällen erfolgt ein direktes Rücksetzen in den Zustand 0.

Die Zustände 1. und 2. bilden den eigentlichen Ablauf des H-Zyklus und werden daher als die aktiven Zustände bezeichnet, in welchem sich die Instanz abwechselnd nach Systemstart aufhält. Hingegen wird Zustand 0. nur bei Systemstart oder in Ausnahmefällen eingenommen.

AHS Register Files (RF) Diese Einheit umfasst den lokalen Registersatz, der innerhalb des H-Zyklus zur Speicherung von Hormonwerten, deren Summen und taskbezogenen Kontrolldaten verwendet wird. Aufgrund der zeitlich unterschiedlichen Verfügbarkeit zwischen den eigenen und empfangenen Hormonen sowie deren

un-/spezifische Wirkungsweise gegenüber Tasks, werden die Werte und Summen in verschiedenen Akkumulations- und Zyklus-Registern gespeichert. Diese werden nun im Einzelnen gemäß ihrer Funktion und dem Speicheraufwand erläutert. Der Speicheraufwand eines Registers ist entweder konstant oder richtet sich nach der Anzahl der m Tasks auf die sich ein Kern C_γ im Netz bewirbt. Ferner bezeichnet $D_{hs} = 2 \cdot D_{hv}$ die Datenmenge (in Bytes), die zur Speicherung einer einzelnen Summe von Hormonwerten benötigt wird.

- **GAR - Global Accumulate Register** Speichert die Summe, die sich aus den Werten aller taskunspezifischen Hormone $H^{M\gamma}$ akkumuliert. Hierzu gehören Suppressoren und Akzeleratoren (Last, Fehler, Temperatur, etc.) die sich auf die Eignungswerte aller Tasks von C_γ auswirken. Diese werden von den Monitoringeinheiten des Kerns oder von den Tasks selbst ausgeschüttet. Der Speicheraufwand des GAR ist daher konstant und es gilt:

$$\text{Speicheraufwand } S_{GAR} = D_{hs} \geq 2 \text{ Byte}$$

- **SAR - Send Accumulate Register** Speichert für jede Task $T_i \in M_\gamma$ die Summe, die sich aus den Werten der eigenen Hormone $H_{i\gamma}$ akkumuliert, sowie den eigenen modifizierten Eignungswert $Em_{i\gamma}$. Die Summen und Eignungswerte werden nacheinander in der Sendephase des aktuellen H-Zyklus i berechnet und gespeichert. Für den taskabhängigen Speicheraufwand des SAR gilt somit:

$$\text{Speicheraufwand } S_{SAR} = m \cdot (D_{hs} + D_{hv}) \geq 3 \text{ Byte}$$

- **RAR - Received Accumulate Register** Speichert für jede Task $T_i \in M_\gamma$ die Summe, die sich aus den Werten der empfangenen Hormone $H^{i\gamma}$ (Suppressoren, Akzeleratoren) akkumuliert, sowie den höchsten empfangenen Eignungswert $Em^{i\gamma}$. Die Summen und Eignungswerte werden fortlaufend innerhalb der Zeitspanne t_{SE} des aktuellen H-Zyklus berechnet, weshalb das Register erst bei Eintritt der Entscheidungsphase die endgültigen stabilen Werte speichert. Bei den Eignungswerten, wird zusätzlich der Absender (Kern-ID) gespeichert, der als zweites Vergleichskriterium gilt, und der mit einer zusätzlichen Datenmenge D_{hv} berücksichtigt wird. Für den taskabhängigen Speicheraufwand des RAR gilt somit:

$$\text{Speicheraufwand } S_{RAR} = m \cdot (D_{hs} + 2 \cdot D_{hv}) \geq 4 \text{ Byte}$$

- **RCR - Received Cycle Register** Speichert aus der Menge der empfangenen Hormone $H^{i\gamma}$ für jede Task $T_i \in M_\gamma$ die Summe sowie den höchsten Eignungswert. Im Gegensatz zum Register RAR, werden hier die stabilen Werte aus dem vorangegangenen H-Zyklus $i - 1$ gespeichert. Diese werden in der Sendephase des aktuellen H-Zyklus i dazu verwendet, die eigenen modifizierten

Eignungswerte $Em_{i\gamma}$ zu berechnen, da sich die Werte des RAR durch die nebenläufig eintreffenden Hormone noch ändern können. Der Speicheraufwand entspricht daher dem Register RAR und es gilt:

$$\text{Speicheraufwand } S_{RCR} = S_{RAR} = m \cdot (D_{hs} + 2 \cdot D_{hv}) \geq 4 \text{ Byte}$$

- **TCR - Task Control Register** Speichert für jede Task $T_i \in M_\gamma$ die Kontrolldaten, die den Status der Taskverwaltung von T_i auf C_γ repräsentieren. Zu den Kontrolldaten gehört unter anderem der aktuelle Stand der Taskübernahme von T_i, der Fortschritt der Optimierungssequenz sowie weitere zyklusbehaftete und optionale Steuerinformationen (Wartezyklus, gesendete Nullhormone etc.). Die Datenmenge $D_{tc} \geq 1$ Byte definiert die benötige Speichermenge pro Task und richtet sich nach der individuellen Funktionalität einzelner Instanzen (Mehrfachübernahmen, Mehrfachoptimierung pro Zyklus etc.).

$$\text{Speicheraufwand } S_{TCR} = m \cdot D_{tc} \geq 1 \text{ Byte}$$

Unter Berücksichtigung der maximalen Anzahl von m_{max} Tasks sowie den Datenmengen $D_{hs} \geq 2$ Byte, $D_{hv} \geq 1$ Byte und $D_{tc} \geq 1$ Byte, gilt für den Gesamtspeicheraufwand einer lokalen Instanz die Summe aus allen oben genannten Registern.

$$\begin{aligned} \text{Gesamtspeicheraufwand } S_{RF} &= S_{GAR} + S_{SAR} + S_{RAR} + S_{RCR} + S_{TCR} \\ &= D_{hs} + m_{max} \cdot (3 \cdot D_{hs} + 5 \cdot D_{hv} + D_{tc}) \\ &= 2 \text{ Byte} + m_{max} \cdot 12 \text{ Byte} \end{aligned} \tag{5.3}$$

AHS Execution Pipelines Diese Einheit umfasst die Funktionseinheiten, welche die eigentliche Verarbeitung von Hormonen innerhalb des H-Zyklus übernehmen. Aufgrund des nebenläufigen Senden und Empfangen von Hormonen, erfolgt die Verarbeitung in getrennten Einheiten, die nachfolgend im Detail erläutert werden.

- **HSP - Hormone Send Pipeline** Diese Einheit modelliert den Verarbeitungszyklus von Hormonen, der während der Sende- und der Entscheidungsphase des H-Zyklus aktiv ist. Dessen Umsetzung erfolgt als skalare Pipeline-Architektur, die eine Durchführung der Zyklussequenz in fünf kaskadierten Verarbeitungsstufen implementiert. Abbildung 5.3 zeigt den Datenpfad der Pipeline für eine maximale Operandengröße (Summen, Werte) mit $D_{hs} \leq 2$ Byte. In der Pipeline werden ausschließlich die eigenen und nicht die empfangenen Hormone verarbeitet. Unabhängig von der Prozessphase (Senden, Entscheiden), erfolgt die Verarbeitung eines oder mehrere Hormone $H_{i\gamma}$ unter Berücksichtigung der zugehörigen Taskdaten (Kontrolldaten, Parameter) von T_i.
  Daher erfolgt in Stufe 1. zunächst das Lesen der Kontrolldaten (TCR) von T_i, Bei mehreren Hormonen der gleichen Task bleiben die Kontrolldaten solange

in dieser Stufe erhalten, bis auch das letzte Hormon in die darauf folgende Stufe 2. eintritt. Bei taskunspezifischen Hormonen (Monitoring-Suppressoren und -Akzeleratoren), wird dieser Schritt vollständig übersprungen (Bypass).
In Stufe 2. erfolgt das Dekodieren der Kontrolldaten, die Erzeugung der Mikro-Steuerbefehle für die nachfolgenden Stufen (EX, PX, WB in Abbildung 5.3) und das Auslesen der hormonbezogenen Operanden von T_i. Diese befinden sich entweder im Registersatz (SAR, RCR, GAR) oder in der Konfiguration (TRS).
Anschließend erfolgt in Stufe 3. die Berechnung des Hormonwerts mittels Multi-Operanden Arithmetik, die eine simultane Verknüpfung von bis zu 6 Operanden in einem Pipeline-Zyklus ermöglicht. In beiden Phasen wird hier der gegenwärtige, modifizierte Eignungswert von T_i berechnet. Zusätzlich werden beim Senden vorab weitere Suppressoren und Akzeleratoren berechnet, die durch eine Übernahme von T_i ausgesendet werden.
Nach der Berechnung erfolgt in Stufe 4. die (Nach-)Auswertung des Hormonwerts. Hier erfolgt in der Entscheidungsphase[1] der Vergleich mit dem höchsten, empfangenen Eignungswert für T_i aus dem Register RCR (Entscheidungsfindung). Beim Senden erfolgt die Erzeugung der restlichen Hormondaten sowie gegebenenfalls eine Akkumulation des Hormonwerts mit der zugehörigen Summe des entsprechenden Registers (SAR, GAR).
In der letzten Stufe 5. erfolgt das Rückschreiben von Ergebnissen der Auswertung. Hierzu zählt in der Entscheidungsphase das speichern der geänderten Kontrolldaten bei Task-Übernahme/-Abgabe sowie weiteren zyklusbehafteten Steuerinformationen von T_i. Beim Senden erfolgt in dieser Stufe zuletzt der Transfer (Broadcast, Multicast) der Hormondaten an die entsprechenden Kerne im Netz.

- **HRP - Hormone Receive Pipeline** Diese Einheit modelliert den Verarbeitungszyklus von empfangenen Hormonen, der während der gesamten Dauer des H-Zyklus aktiv ist. Die Umsetzung erfolgt als skalare Pipeline-Architektur mit vier kaskadierten Verarbeitungsstufen. Im Gegensatz zur HSP erfolgt die Verarbeitung eines empfangenen Hormons $H^{i\gamma}$ ohne die Taskdaten (Kontrolldaten, Parameter) der Empfängertask T_i. Des Weiteren werden ausschließlich fremde Hormone verarbeitet, da die Auswertung der eigenen, gesendeten Hormone $H^{i\gamma}_{j\gamma}$ für T_i bereits beim Senden durch die HSP erfolgt. In Stufe 1. erfolgt der Empfang von Hormonen, wobei die Daten des aktuell eintreffenden Hormon $H^{i\gamma}$ auf die Empfängertask T_i überprüft und gegebenenfalls verworfen werden, falls $i \notin M_\gamma$ gilt. Anschließend erfolgt in Stufe 2. die weitere Dekodierung des Hormons (Typ, Wert) sowie das Lesen des zugehörigen Operanden

[1] Bei der in Kapitel 4.2 vorgestellten klassischen Variante erfolgt der Vergleich von Eignungswerten auch beim Senden, da die Task T_i in der darauffolgenden Entscheidungsphase des H-Zyklus auf Übernahme geprüft wird, sofern der berechnete Eignungswert größer ist als der höchste, empfangene Wert.

(Hormonsumme, Eignungswert) von T_i aus dem Register RAR. In Stufe 3. erfolgt die Auswertung, bei dem der Wert von $H^{i\gamma}$ gemäß dessen Typs entweder mit dem Operanden aus dem RAR akkumuliert oder verglichen wird (negative Summen werden im Zweierkomplement gespeichert). Zuletzt erfolgt in Stufe 4. das Rückschreiben der manipulierten Summe bzw. des empfangenen Werts als höchster Eignungswert in das Register RAR.

Aufgrund der zeitlich parallelen Verarbeitung von Hormonen, können in beiden Pipelines Datenkonflikte (RAW) auftreten, die durch Abhängigkeiten zwischen den Operanden der Hormone entstehen. Diese werden mittels gezielter Weiterleitung (Forwarding) der betroffenen Operanden gelöst, so dass in der Sequenz keine zeitliche Verzögerung infolge zusätzlicher Straf- oder Wartezyklen entsteht. Des Weiteren werden in der Sendephase zuerst alle Suppressoren und Akzeleratoren verarbeitet, so dass deren Auswirkung auf die eigenen Eignungswerte bei der Berechnung berücksichtigt wird.

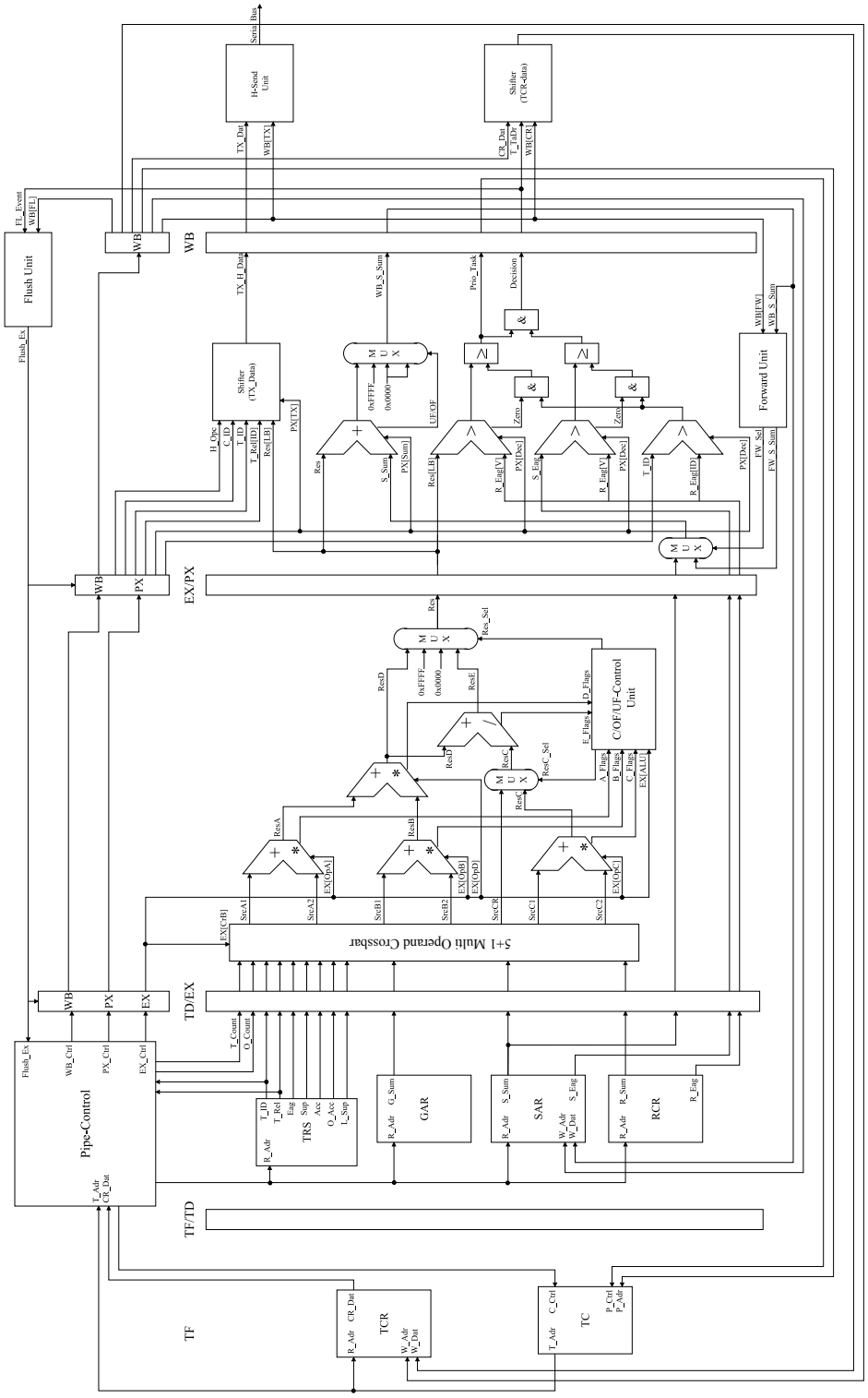


Abbildung 5.3.: Daten- und Kontrollpfad der Hormone Send Pipeline (HSP)

5.1.3. Quantifizierung der H-Zykluszeit des Hardwareentwurf

Abschließend erfolgt unter Berücksichtigung des im vorigen Abschnitt beschriebenen Funktionsverhaltens einzelner Entwurfseinheiten, eine Quantifizierung der H-Zykluszeit des Hardwareentwurfs. Entsprechend dem realen Zeitverhalten in Abschnitt 5.1.1, definiert sich die H-Zykluszeit aus den beiden Zeitspannen t_{SE} und t_{ES}, in denen jeweils nebenläufig die drei Vorgänge Senden, Empfangen und Entscheiden erfolgen. Nun wird die maximale Zeitdauer dieser Vorgänge in Abhängigkeit des zu verarbeitenden Hormonaufkommens im Anfangs- wie auch im Endzustand des KHS untersucht, um daraus eine Berechnungsvorschrift für die zyklusbehafteten Zeiten t_{SE}, t_{ES} und der resultierenden minimalen H-Zykluszeit abzuleiten.

$$Hs_\gamma = lm + s + a + e \tag{5.4a}$$
$$Hr_\gamma = s + a + e \tag{5.4b}$$
$$He_\gamma = e \tag{5.4c}$$

$$\begin{aligned} lm &:= \text{ Anzahl der Monitoring-Hormone} \\ s &:= \text{ Anzahl der Suppressoren} \\ a &:= \text{ Anzahl der Akzeleratoren} \\ e &:= \text{ Anzahl der Eignungswerte} \end{aligned}$$

- Die Summe Hs_γ in Gleichung 5.4a definiert die Gesamtanzahl an Hormonen, die während der Sendephase von C_γ ausgeschüttet wird. Dies erfolgt innerhalb der HS-Pipeline und umfasst auch die Verarbeitung der nicht gesendeten Nullhormone. Die Gesamtanzahl berücksichtigt somit die Hormone des lokalen Monitoring (lm), Übernahme-und Last-Suppressoren (s), Organ- und Angebots-Akzeleratoren (a) sowie die modifizierten Eignungswerte (e) von C_γ.

- Die Summe Hr_γ in Gleichung 5.4b definiert die Gesamtanzahl an Hormonen, die während der Zeitspanne t_{SE} auf C_γ empfangen und durch die HR-Pipeline verarbeitet wird. Zu den empfangenen Hormonen zählen Übernahme-Suppressoren (s), Organ-Akzeleratoren (a) und die modifizierten Eignungswerte (e) der anderen Kerne $C_\delta \in \Omega \setminus C_\gamma$.

- Die Summe He_γ in Gleichung 5.4c definiert die Gesamtanzahl an Hormonen, die für die Entscheidungsfindung in der HS-Pipeline von C_γ verarbeitet wird. Hierzu zählen ausschließlich die gesendeten Eignungswerte (e) von C_γ.

Nachfolgend werden die aufgeführten Summen für den Start- und den eingeschwungenen Endzustand des KHS quantifiziert, um daraus das maximale Hormonaufkommen für den jeweiligen Vorgang (Senden, Empfangen, Entscheiden) abzuleiten.

Im Startzustand befindet sich das KHS in der Selbst-Konfiguration, bei der noch keine Tasks im System verteilt sind. Die Summe ${}_{Start}Hs_\gamma$ akkumuliert sich daher aus

der Anzahl der lokal ausgeschütteten Monitoring-Hormone lm_γ plus den gesendeten Eignungswerten bezüglich der m_γ Tasks auf die sich C_γ bewirbt. Für die Anzahl der gesendeten Suppressoren und Akzeleratoren gilt in diesem Zustand $s, a = 0$. Folglich definiert sich ${}_{Start}Hr_\gamma$ ausschließlich aus der Summe der empfangenen Eignungswerte. Falls sich C_γ auf alle m Tasks bewirbt, müssen schlimmsten Falls alle empfangenen Eignungswerte, außer den eigenen, verarbeitet werden. Hingegen richtet sich ${}_{Start}He_\gamma$ nach dem verwendeten Entscheidungsverfahren aus Kapitel 4.2, weshalb die Anzahl der zu prüfenden Tasks (Eignungswerte) entweder konstant ist oder durch m_γ begrenzt wird.

Das Hormonaufkommen auf C_γ im Startzustand beträgt somit:

$$\left.\begin{aligned} {}_{Start}Hs_\gamma &= lm + e = lm_\gamma + m_\gamma \\ {}_{Start}Hr_\gamma &= e \quad \leq \sum_{C_\delta \in \Omega \setminus C_\gamma} m_\delta \\ {}_{Start}He_\gamma &= e \quad \leq \begin{cases} 1 & : \text{klassisch} \\ m_\gamma & : \text{aggressiv} \end{cases} \end{aligned}\right\} \; s, a = 0 \text{ im Startzustand}$$

Im eingeschwungenen Endzustand ist die Selbst-Konfiguration des KHS abgeschlossen und alle Tasks verteilt. Es werden deshalb keine Eignungswerte mehr gesendet. Dennoch müssen diese zu Null gewordenen Hormone von C_γ in jeder Sendephase neu berechnet werden, um auf mögliche Veränderung der Hormonpegel bei Selbst-Optimierung oder -Heilung zu reagieren. Die Summe ${}_{Ende}Hs_\gamma$ setzt sich daher nach wie vor aus der Anzahl der Monitoring-Hormone plus den berechneten aber nicht gesendeten Eignungswerten von C_γ zusammen. Hinzu kommen nun noch die Suppressoren und Akzeleratoren, die durch die Übernahme der e_γ Tasks ausgesendet werden. Für jeden übernommenen Task $T_i \in E_\gamma$, wird jeweils ein Übernahme- und ein Last-Suppressor gesendet. Die Anzahl der gesendeten Organ-Akzeleratoren definiert sich über die Summe der v_i Verwandschaften aller e_γ übernommenen Tasks T_i. Des Weiteren werden zur Selbst-Optimierung von o_γ Tasks entsprechend viele Angebots-Akzeleratoren gesendet, welche die aktuelle Zuordnung der betroffenen Tasks festigen. Folglich gilt für ${}_{Ende}Hr_\gamma$ ausschließlich die Anzahl der empfangenen Übernahme-Suppressoren und Organ-Akzeleratoren[2]. Gleichermaßen gilt daher wie im Startzustand, dass schlimmsten Falls alle empfangenen Übernahme-Suppressoren sowie alle Organ-Akzeleratoren der φ_γ benachbarten Kerne verarbeitet werden müssen (außer den eigenen). Das Hormonaufkommen ${}_{Ende}He_\gamma$ beträgt im Endzustand Null, kann aber schlimmsten Falls bei Selbst-Optimierung um eine Anzahl

[2]Bei Selbst-Optimierung, werden die zusätzlich empfangenen Eignungswerte durch die nicht verarbeiteten, zu Null gewordenen Übernahme-Suppressoren kompensiert, weshalb die Hormonanzahl gleichbleibend ist.

von o_γ zu prüfender Tasks ansteigen.

Das Hormonaufkommen auf C_γ im eingeschwungenen Endzustand beträgt daher:

$$\left.\begin{aligned}
{}_{Ende}Hs_\gamma &= lm + s + a + e \\
&= lm_\gamma + 2 \cdot e_\gamma + \sum_{T_i \in E_\gamma} v_i + o_\gamma + m_\gamma \\
{}_{Ende}Hr_\gamma &= s + a \\
&\leq \sum_{C_\delta \in \Omega \setminus C_\gamma} e_\delta + \sum_{C_\delta \in \Phi_\gamma \setminus C_\gamma} \sum_{T_i \in E_\delta} v_i \\
{}_{Ende}He_\gamma &= e \\
&= \begin{cases} 1 & : \text{klassisch} \\ o_\gamma & : \text{aggressiv} \end{cases}
\end{aligned}\right\} \text{ im Endzustand}$$

Unter Berücksichtigung des einzelnen Hormonaufkommens für den Start- als auch für den Endzustand, erfolgt nun die Abschätzung der maximal zu verarbeitenden Hormonanzahl in allen drei Vorgängen.
Für das Senden gilt, dass auf jedem Kern die Anzahl der ausgeschütteten Hormone im Endzustand stets größer ist, als die im Startzustand (${}_{Ende}Hs_\gamma > {}_{Start}Hs_\gamma, \forall C_\gamma \in \Omega$). Folglich wird das maximale Hormonaufkommen beim Senden durch den Kern im Netz definiert, der im Endzustand die größte Anzahl an Hormonen ausschüttet.

Das größtmögliche, zu verarbeitende Hormonaufkommen eines Kerns beim Senden beträgt daher:

$$\begin{aligned}
Hs_{max} &= \max_{C_\gamma \in \Omega} \{{}_{Ende}Hs_\gamma\} \\
&= lm_{max} + e_{max} \cdot (2 + v_{max}) + o_{max} + m_{max}
\end{aligned} \tag{5.5}$$

$$\begin{aligned}
lm_{max} &:= \max_{C_\gamma \in \Omega} \{lm_\gamma\}, && \text{Maximum lokaler Monitoring-Hormone} \\
e_{max} &:= \max_{C_\gamma \in \Omega} \{e_\gamma\}, && \text{Maximum übernommener Tasks} \\
v_{max} &:= \max_{T_i \in M} \{v_i\}, && \text{Maximum an Task-Verwandtschaften} \\
o_{max} &:= \max_{C_\gamma \in \Omega} \{o_\gamma\}, && \text{Maximum optimierter Tasks} \\
m_{max} &:= \max_{C_\gamma \in \Omega} \{m_\gamma\}, && \text{Maximum an Tasks}
\end{aligned}$$

Die Abschätzung des maximalen Hormonaufkommen beim Empfangen erfolgt unter der Annahme, dass in beiden Zuständen des KHS alle Kerne genau so viele Hormone senden, wie der Kern mit der höchsten gesendeten Hormonanzahl. Für den Startzustand bedeutet dies, dass ein Kern schlimmsten Falls von jedem der $\omega - 1$ anderen Kern m_{max} Eignungswerte empfängt und verarbeitet. Hingegen

empfängt ein Kern im Endzustand maximal von jedem der $\omega - 1$ anderen Kerne e_{max} Übernahme-Suppressoren plus die maximale Anzahl an Organ-Akzeleratoren, die von benachbarten Kernen gesendet werden. Hierbei wird angenommen, dass jeder Kern die höchste Anzahl von φ_{max} benachbarten Kernen hat und jeder Nachbar die höchste Anzahl an Organ-Akzeleratoren sendet. Damit kann nun das maximale Hormonaufkommen beim Empfangen über das Maximum aus beiden Phasen berechnet werden.

Das größtmögliche, zu verarbeitende Hormonaufkommen eines Kerns beim Empfangen beträgt daher:

$$Hr_{max} = max\{{}_{Start}Hr_{max}, {}_{Ende}Hr_{max}\} \tag{5.6}$$

$$\begin{aligned} {}_{Start}Hr_{max} &= \max_{C_\gamma \in \Omega}\{{}_{Start}Hr_\gamma\} \\ &= (\omega - 1) \cdot m_{max} \\ {}_{Ende}Hr_{max} &= \max_{C_\gamma \in \Omega}\{{}_{Ende}Hr_\gamma\} \\ &= ((\omega - 1) \cdot e_{\max}) + ((\varphi_{max} - 1) \cdot e_{max} \cdot v_{max}) \end{aligned}$$

$$\begin{aligned} e_{max} &:= \max_{C_\gamma \in \Omega}\{e_\gamma\}, && \text{Maximum übernommener Tasks} \\ v_{max} &:= \max_{T_i \in M}\{v_i\}, && \text{Maximum an Task-Verwandtschaften} \\ m_{max} &:= \max_{C_\gamma \in \Omega}\{m_\gamma\}, && \text{Maximum an Tasks} \\ \varphi_{max} &:= \max_{C_\gamma \in \Omega}\{\varphi_\gamma\}, && \text{Maximum an benachbarten Kernen} \end{aligned}$$

Für das Entscheiden gilt, dass bei aggressiver Übernahmevariante auf jedem Kern die Anzahl der zu prüfenden Tasks im Startzustand stets am größten ist (${}_{Start}He_\gamma \geq {}_{Ende}He_\gamma, \forall C_\gamma \in \Omega$, aggressive Strategie). Folglich wird das maximale Hormonaufkommen beim Entscheiden durch den Kern begrenzt, der sich im Startzustand auf die größte Anzahl von m_{max} Tasks bewirbt und schlimmsten Falls gleich viele, gesendete Eignungswerte überprüft.

Das größtmögliche, zu verarbeitende Hormonaufkommen eines Kerns beim Entscheiden beträgt daher:

$$\begin{aligned} He_{max} &= \max_{C_\gamma \in \Omega}\{{}_{Start}He_\gamma\} \\ &= m_{max} \end{aligned} \tag{5.7}$$

Zusammenfassend kann nun für jeden der drei Vorgänge das Worst-Case Zeitverhalten berechnet werden. Da die Verarbeitung von Hormonen mittels skalarer Pipeline-Ausführung erfolgt, die zeitlich keine Verzögerungen aufweist, kann die maximal benötigte Zeitdauer für einen Vorgang über dessen Ausführungszeit in der zugehörigen Pipeline bei maximalen Hormonaufkommen berechnet werden.

Dies erfolgt unter Berücksichtigung der insgesamt K Verarbeitungsstufen sowie der Zykluszeit der Pipeline[3]. Beim Senden und Entscheiden erfolgt die Verarbeitung der entsprechenden Hormone innerhalb der HS-Pipeline mit $K = 5$ Verarbeitungsstufen und der Zykluszeit t_{CS}. Die Verarbeitung der empfangenen Hormonen erfolgt innerhalb der HR-Pipeline mit $K = 4$ Verarbeitungsstufen und der Zykluszeit t_{CE}.

Für die Vorgänge Senden, Empfangen und Entscheiden ergeben sich somit unter Berücksichtigung der genannten Pipeline-Parameter und des maximal zu verarbeitenden Hormonaufkommen aus Gleichung 5.5, 5.6 und 5.7 die folgenden Worst-Case Ausführungszeiten:

Worst-Case Sendezeit $$t_{Hs} = (5 + Hs_{max} - 1) \cdot t_{CS} \tag{5.8}$$

Worst-Case Empfangszeit $$t_{Hr} = (4 + Hr_{max} - 1) \cdot t_{CE} \tag{5.9}$$

Worst-Case Entscheidezeit $$t_{He} = (5 + He_{max} - 1) \cdot t_{CS} \tag{5.10}$$

5.1.3.1. Abschließende Quantifizierung der H-Zykluszeit

Abschließend kann die Quantifizierung der minimalen H-Zykluszeit für den Entwurf auf Basis der Ergebnisse des vorangegangenen Abschnitts erfolgen. Da sich die Zykluszeit aus der Summe der Zeitspannen t_{SE} und t_{ES} definiert, erfolgt die Quantifizierung über die Kernzeiten t_K und Δt_{SE} des H-Zyklus gemäß Gleichung 5.1. Die Kommunikationszeit t_K umfasst die Zeitspanne die für das Senden und Empfangen der größtmöglichen Hormonmenge Hs_{max} im Netz benötigt wird. Da innerhalb des H-Zyklus das Senden und Empfangen von Hormonen nebenläufig erfolgt, umfasst t_K daher auch den Zeitraum der nach dem Senden verstreicht, bis auch der letzte Kern im Netz die von C_γ gesendete Hormonmenge Hs_{max} innerhalb seiner HR-Pipeline verarbeitet hat (vollständig empfangen). Im besten anzunehmenden Fall gilt, dass die Menge Hs_{max} bereits nach Ablauf der Sendezeit $t_{Hs_{max}}$ vollständig auf allen Kernen empfangen wurde, plus dem zeitlichen Versatz der zwischen beiden Pipelines herrscht ($= 4 \cdot t_{CE}$), der jedoch aufgrund des geringen Zeitanteils gegenüber der Zeitspanne $t_{Hs_{max}}$ vernachlässigt werden kann. In allen anderen Fällen gilt, dass nach Ablauf von $t_{Hs_{max}}$ die Menge Hs_{max} gesendet, aber noch nicht vollständig auf allen Kernen empfangen wurde, was auf die sequentielle Pipeline-Auswertung der betroffenen Hormone innerhalb der gesamt möglichen Empfangsmenge Hr_{max} zurückzuführen ist. Im schlimmsten anzunehmenden Fall gilt deshalb, dass die von C_γ gesendeten Hormone erst nach Ablauf der Zeit t_{Hr} vollständig empfangen wurden. Folglich definiert sich die Kommunikationszeit durch die Nebenläufigkeit zwischen Senden und

[3]Die Pipeline-Zykluszeit misst die maximale Berechnungsdauer eines Hormons in einer der K Pipeline-Stufen.

Empfangen aus dem Maximum beider Ausführungszeiten t_{Hs} und t_{Hr}.

$$\textbf{Worst-Case Kommunikationszeit} \quad t_K = max\{t_{Hs}, t_{Hr}\} \tag{5.11}$$

Für den Jitter Δt_{SE} innerhalb der Zeitspanne t_{SE} gilt ebenfalls das Maximum, welches sich aus der Zeit für die Entscheidungsfindung t_{He} und dem tatsächlichen Jitter t_{JT} definiert, welcher durch die zeitlichen Varianzen lokaler Funktionseinheiten (Zeitgebervarianzen, Verzögerungszeiten etc.) auf der jeweiligen Zielplattform verursacht wird.

$$\textbf{Worst-Case Jitter} \quad \Delta t_{SE} = max\{t_{He}, t_{JT}\} \tag{5.12}$$

Durch Einsetzen der beiden Kernzeiten t_K und Δt_{SE} in Gleichung 5.1 erhalten wir die Berechnungsvorschrift für die minimale H-Zykluszeit des Hardwareentwurfs.

$$\textbf{Minimale H-Zykluszeit} \quad \begin{aligned} t_{HC} &= t_{SE} + t_{ES} \\ &= 3 \cdot t_K + \Delta t_{SE} \\ &= 3 \cdot max\{t_{Hs}, t_{Hr}\} + max\{t_{He}, t_{JT}\} \end{aligned} \tag{5.13}$$

$$t_{SE} \geq 2 \cdot t_K = 2 \cdot max\{t_{Hs}, t_{Hr}\}$$

$$t_{ES} \geq t_K + \Delta t_{SE} = max\{t_{Hs}, t_{Hr}\} + max\{t_{He}, t_{JT}\}$$

Beispiel für die Berechnung einer minimalen H-Zykluszeit Zur Veranschaulichung der hergeleiteten Quantifizierung, erfolgt die Berechnung der minimalen H-Zykluszeit für ein Beispiel-Szenario mit konkreten Werten. Hierfür wird für das KHS ein typisches System gewählt, in welchem 64 Tasks dynamisch auf 64 Kerne verteilt werden müssen. Für die in der Quantifizierung spezifizierten KHS-Parameter wurde eine System-Konfiguration mit den folgenden Werten gewählt.

$lm_{max} := 2,$	Maximum lokaler Monitoring-Hormone
$e_{max} := 2,$	Maximum an Taskübernahmen pro Kern
$v_{max} := 8,$	Maximum an Task-Verwandtschaften
$o_{max} := 1,$	Maximum zeitgleich, optimierter Tasks pro Kern
$m_{max} := 32,$	Maximum an Tasks, auf die sich ein Kern bewirbt
$\varphi_{max} := 9,$	Maximum an benachbarten Kernen
$\omega := 64,$	Anzahl der Kerne

Des Weiteren wurden für die Zielplattform abhängigen Pipeline-Zykluszeiten t_{CS} und t_{CE} ebenfalls konkrete Werte angenommen, die aus der Portierung des VHDL-Hardwareentwurfs auf die in Kapitel 6 verwendete Zielplattform ermittelt wurden.

Unter Verwendung der aufgeführten System-Konfiguration, erfolgt zuerst die Berechnung des maximalen Hormonaufkommens für die Vorgänge Senden, Empfangen und Entscheiden des H-Zyklus. Durch Einsetzen der KHS-Parameter in Gleichung 5.5, 5.6 und 5.7 erhalten wir das maximale Hormonaufkommen für einen Kern im Systembetrieb.

$$\boldsymbol{Hs_{max}} = 2 + 2 \cdot (2 + 8) + 1 + 32 = 55 \text{ Hormone}$$

$$\begin{aligned}\boldsymbol{Hr_{max}} &= max\{(64 - 1) \cdot 32,\ ((64 - 1) \cdot 2) + ((9 - 1) \cdot 2 \cdot 8)\} = \\ &= 2016 \text{ Hormone}\end{aligned}$$

$$\boldsymbol{He_{max}} = 32 \text{ Hormone}$$

Durch weiteres Einsetzen der errechneten Werte für das Hormonaufkommen in Gleichung 5.8, 5.9 und 5.10 erhalten wir für jeden Vorgang die zugehörige Ausführungszeit, wobei für die beiden Pipeline-Zykluszeiten, die Werte des Prototypen (siehe Abschnitt 6) mit $t_{CS} = 0.13\,\mu s$ und $t_{CE} = 0.05\,\mu s$ eingesetzt wurden[4].

$$\boldsymbol{t_{Hs}} = (5 + 55 - 1) \cdot 0.13\ \mu s = 7.67\,\mu s$$

$$\boldsymbol{t_{Hr}} = (4 + 2016 - 1) \cdot 0.05\ \mu s = 100.95\,\mu s$$

$$\boldsymbol{t_{He}} = (5 + 32 - 1) \cdot 0.13\ \mu s = 4.68\,\mu s$$

Mit Hilfe der Ausführungszeiten lassen sich nun die Werte für die maximale Kommunikationszeit in Gleichung 5.11 sowie den maximalen Jitter in Gleichung 5.12 berechnen. Für die zeitliche Varianz t_{JT} des Jitters wurde analog zu den Pipeline-Zykluszeiten der Wert mit $t_{JT} = 0.04\,\mu s$ für die Zielplattform aus Kapitel 6 verwendet.

$$\boldsymbol{t_K} = max\{7.67, 100.95\}\ \mu s = 100.95\,\mu s$$

$$\boldsymbol{\Delta t_{SE}} = max\{4,68, 0.04\}\ \mu s = 4.68\,\mu s$$

Abschließend erhalten wir unter Verwendung von Gleichung 5.13 die minimale H-Zykluszeit für das aufgeführte Beispiel-Szenario (gültig für jede KHS-Instanz im Netz unter Berücksichtigung der aufgeführten System-Konfiguration).

$$\boldsymbol{t_{HC}} = 3 \cdot 100.95 + 4,68\ \mu s = 307.53\,\mu s = 0.31\,ms$$

[4]Die für die beiden Pipeline-Zykluszeiten t_{CS} und t_{CE} aufgeführten Werte sind unabhängig von der verwendeten System-Konfiguration (m_{max}, ϕ_{max} etc.) und wurden durch Portierung des AHS*LiHaDe* für eine XILINX Virtex-6 Zielplattform mit einer System-Taktfrequenz von 250 MHz ermittelt.

Die berechneten Zahlenwerte sind für den Betrieb des aufgeführten Beispiel-Systems auf der in Kapitel 6 verwendeten Zielplattform gültig. Bei einer Portierung auf anderweitige Zielarchitekturen (anderer FPGA-Chip, ASIC), ergeben sich aufgrund der plattformabhängigen Pipeline-Zykluszeiten (t_{CS},t_{CE}) und der zeitlichen Jitter-Varianz (t_{JT}) gegebenenfalls abweichende Werte. Anhand des Beispiels wird verdeutlicht, dass das Hormonaufkommen und die resultierende H-Zykluszeit im Wesentlichen von der Anzahl der Kerne (ω), den Task-Verwandtschaften (v_{max}) und den Taskübernahmen pro Kern (e_{max}) dominiert wird. Für dasselbe System mit einer maximalen Anzahl an Taskübernahmen pro Kern ($e_{max} = m_{max}$) errechnet sich beispielhaft bereits die doppelte H-Zykluszeit. Des Weiteren ist zu berücksichtigen, dass ausschließlich das durch die Taskverwaltung verursachte Hormonaufkommen berechnet wurde. Für das totale Hormonaufkommen im System ist zusätzlich die Hormonmenge seitens der Taskverarbeitung (V-Zyklus) zu berücksichtigen, deren Quantifizierung in Abschnitt 5.2.3 erfolgt.

5.2. HTV zum Einsatz auf Multi-Core Systemen

Für die Evaluation der hormongeregelten Taskverarbeitung (HTV), wurde das in Kapitel 4 vorgestellte Taskverarbeitungsmodell des V-Zyklus erstmals als eigenständige Implementierung umgesetzt, die zusammen mit der Umsetzung für das KHS den gesamten Entwurf des Taskverarbeitungssystems auf realer SoC-Zielplattform bilden. Hierfür wurde die HTV aus Sichtweise eines Drei-Ebenenmodells realisiert, das die

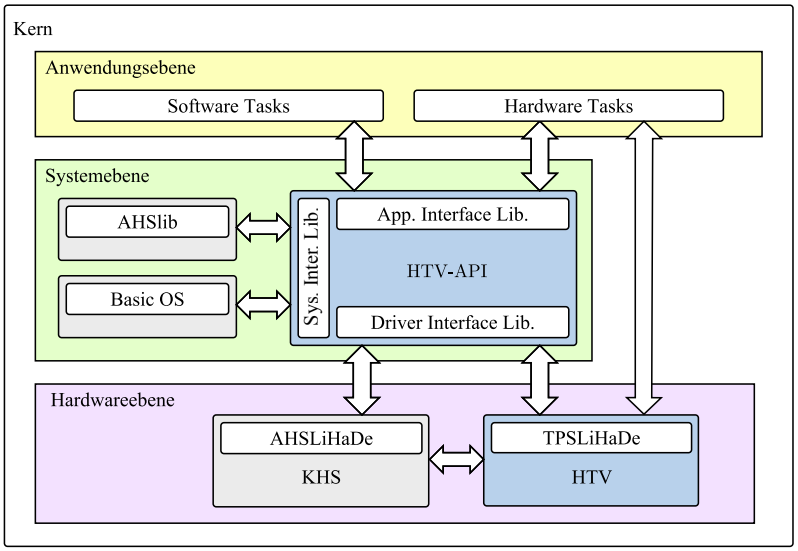


Abbildung 5.4.: Aufbau und Struktur der HTV-Implementierung

Umsetzung in zwei voneinander getrennten Teilen implementiert, die auf Hardware- und Systemebene eingesetzt werden und in Abbildung 5.4 farblich blau dargestellt

sind. Wie anhand des Ebenenmodells zu erkennen ist, setzt sich die Implementierung aus einem statischen Hardwareentwurf und einer adaptiven Programmierschnittstelle zusammen. Der Hardwareentwurf (TPS*LiHaDe*) realisiert die lokale Funktionalität des V-Zyklus durch den Einsatz eigener Rechen-Ressourcen. Die Programmierschnittstelle (HTV-API) übernimmt die Funktion einer dynamischen Bibliothek, die zur Anbindung des HTV-Hardwareteils sowie des KHS an die Systemebene genutzt wird und somit die Kommunikation mit anderen Systemkomponenten (Tasks, Betriebssystem etc.) auf höherliegenden Ebenen ermöglicht. Der Vorteil in dieser eindeutigen Trennung der Implementierung liegt in der individuellen Nutzbarkeit und Integration beider Teile. Da die verwendeten Kerne je nach Funktionalität entweder über keine, eine oder mehrere Systemebenen verfügen, können beide Teile der Taskverarbeitung nach Bedarf passend für jeden Kern eingebunden werden, sodass der Mehraufwand der durch die zusätzliche Nutzung und Integration der HTV entsteht lokal minimiert wird. In den nachfolgenden Abschnitten werden die beiden Implementierungsteile nun genauer im Detail betrachtet.

5.2.1. HTV-Entwurf zum Einsatz auf Hardwaresystemebene

Um den V-Zyklus unabhängig von den funktionalen Eigenschaften der verwalteten Kerne auf Hardwareebene umzusetzen, wurde ein weiteres VHDL-Modell erstellt, dass die Taskverarbeitung als ergänzendes Schaltwerk implementiert und als TPS*LiHaDe* (Task Processing System - Lightweight Hardware Described) bezeichnet wird. Der Einsatz dieses Schaltwerks erfolgt aber auch unter Berücksichtigung der lokalen Anforderungen der System- und Anwenderprogramme. Der Betrieb kann daher mit oder ohne Programmierschnittstelle (HTV-API) erfolgen. Analog zur Modellierung des AHS*LiHaDe* wurden ausschließlich synthesefähige VHDL-Konstrukte verwendet um eine Portierbarkeit auf tiefere Entwurfsebenen und auf die in Kapitel 6 verwendete Zielplattform zu garantieren. Die Funktionalität des Schaltwerks umfasst die Ablaufkontrolle des V-Zyklus sowie die Durchführung der Verarbeitungsschritte $\dot{Z}$ und $\dot{E}$, die sich durch den Transfer und die Verarbeitung von Taskkomponenten definieren. Durch die Bereitstellung dieser Funktionalität als zusätzliches Schaltwerk soll sichergestellt werden, dass das Taskverarbeitungsmodell auch auf Kernen eingesetzt werden kann, die eine Ausführung des V-Zyklus über die API nicht unterstützen. Im Gegensatz zu den Softwareinstanzen der API werden dann nicht die Funktionseinheiten des verwalteten Kerns, sondern nur die eigenen genutzt. Zusätzlich bietet das TPS*LiHaDe* dem AHS*LiHaDe* die Möglichkeit über Kommunikationskanäle (Interruptvektoradressen etc.) direkt mit den Tasks auf Anwendungsebene zu kommunizieren. Die Implementierung des VHDL-Modells beinhaltet drei Entwurfseinheiten die nachfolgend vorgestellt werden.

TPC - Cycle Control Unit (CCU) Diese Einheit umfasst die Kontrollogik die den Sequenzablauf des V-Zyklus steuert und überwacht. Die Umsetzung erfolgt mittels

Zustandsautomaten, der die gesendeten und empfangenen Übernahme-Suppressoren des aktuellen H-Zyklus lokal analysiert und daraus das erforderliche Sequenzverhalten einleitet. Dazu werden unter anderem die Funktionseinheiten des AHS*LiHaDe* genutzt. Die Analyse von Suppressoren erfolgt zeitlich parallel zu deren Verarbeitung innerhalb des KHS. Die Durchführung der Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ der Sequenz erfolgt dann in den Funktionseinheiten der *Direct Data Transfer*.

TPC - Direct Data Transfer Unit (DDT) Die DDT-Einheit modelliert die Umsetzung der Schritte $\dot{\text{Z}}$ und $\dot{\text{E}}$ des V-Zyklus. Die Funktionalität beschränkt sich auf den Transfer von Daten zur Bereitstellung der Komponenten und der Kommunikation von Tasks im System. Ein Transfer umfasst neben der eigentlichen Übertragung auch die lokale Vor- Nachverarbeitung der Daten, die das Auslesen und Rückschreiben der Daten in die Speicherbereiche der Tasks übernimmt. Die Durchführung von Transfers erfolgt durch Sequenzierung der Gesamtdatenmenge in einzelne Datenhormone. Die Verarbeitung der Datenhormone erfolgt zeitlich parallel, so dass eine nebenläufige Durchführung unterschiedlicher Transfersequenzen entsteht. Die Umsetzung erfolgt in zwei getrennten Pipelines.

- **DSP - Data Send Pipeline** Diese Einheit modelliert den Sendezyklus eines einzelnen Datenhormons mittels skalarer Pipeline-Verarbeitung in $K = 3$ kaskadierten Stufen. In Stufe 1 werden die Kontroll- und Kontextdaten der zugehörigen Transfersequenz geladen und dekodiert. Stufe 2 umfasst das Auslesen der Transferdaten aus dem Speicher und die Erzeugung der Hormondaten. Im letzten Schritt erfolgt das Senden des vollständigen Datenhormons sowie die Aktualisierung der Kontrolldaten im Kontrollregister.

- **DRP - Data Receive Pipeline** Diese Einheit modelliert den zur DSP gegenläufigen Pipeline-Verarbeitungszyklus eines empfangenen Datenhormons ($K = 3$). Ausgehend vom Zeitpunkt der Zustellung erfolgt in Stufe 1 zunächst das Auslesen des Datenhormons aus der Empfangseinheit. Anschließend erfolgt in Stufe 2 die Dekodierung und die Zuordnung des Datenhormons in die zugehörige Transfersequenz. Im letzten Schritt erfolgt das Rückschreiben der Transferdaten in den Speicherbereich der zugehörigen Task.

Der interne Zugriff auf Taskkomponenten erfolgt mittels Direktzugriff (Direct Memory Access Unit) auf den lokalen Speicher des verwalteten Kern. Dies ermöglicht eine nebenläufige Durchführung von Speicherzugriffen ohne dabei die Recheneinheiten des verwalteten Kerns zu nutzen.

TCS - Configuration Set (CS) Diese Sekundäre Einheit beinhaltet die Konfiguration, die für die Instantiierung des TPS*LiHaDe* im System benötigt wird. Sie umfasst Parameter und Kontextdaten die für den Datentransfer benötigt werden.

Hierzu zählen unter anderem die statischen Speicher- und Interruptvektoradressen der verarbeitenden Tasks.

5.2.2. HTV-Programmierschnittstelle zur Anbindung an die Systemebene

Um die Taskverarbeitung (HTV) als auch die Taskverwaltung (KHS) aus einer Softwareumgebung heraus verwenden zu können, wurde eine Programmier-Schnittstelle erstellt, welche die Funktionalität und den Zugriff auf die zugehörigen Hardwarekomponenten (TPSLiHaDe, AHSLiHaDe) gegenüber System-, Dienst- und Anwenderprogrammen (Betriebssystem, Anwendungstasks etc.) zur Verfügung stellt. Es handelt sich dabei um eine Ansammlung von Softwarebibliotheken, welche die Anbindung und die Nutzung der Hardwarekomponenten über eigens vordefinierte Routinen, Schnittstellen und Gerätetreiber implementiert. Hierfür muss die API als lokale Instanz auf dem jeweiligen Kern ausgeführt werden. Der Einsatz der API ist aber optional und beschränkt sich im wesentlichen auf Kerne, deren Ressourcenverwaltung auf Systemebene mittels Betriebssystem erfolgt, oder deren funktionale Programmanbindung gegenüber den Anwendungstasks unzureichend ist. Um eine weitreichende Portierbarkeit der API auf reale SoC-Umgebungen zu gewährleisten, wurden die plattformunabhängigen Komponenten komplett in ANSI-C implementiert und umfassen insgesamt 12.387 Programmzeilen (Source Lines of Code, SLC). Die kompilierte Objektdatei umfasst eine Größe von 35 KBytes. Die API selbst lässt sich in drei Bibliotheken unterteilen die nachfolgend erläutert werden.

System Interface Library Diese Bibliothek implementiert den eigentlichen Systemkernel der API und beinhaltet neben Schnittstellen-Funktionen auch weitere Basisklassen, die eine alternative Ausführung des V-Zyklus als Softwareinstanz auf Systemebene ermöglichen. Eine weitere Funktion der Bibliothek liegt in der Interaktion zwischen HTV, dem Hardware-KHS und dem Betriebssystem. Zu den elementaren Routinen, die beispielsweise das Konfigurieren, Starten und Abbrechen von Tasks, sowie deren Administrierung innerhalb des Prozess-Scheduler übernehmen, werden auch plattformabhängige Schnittstellen- Deklarationen bereitgestellt, die für die Erzeugung von Kommunikationskanälen (Sockets) oder Thread-Instanzen benötigt werden und für das jeweilige System entsprechend angepasst werden müssen.

Application Interface Library Diese Bibliothek realisiert die Programmanbindung der Applikations-Tasks mit dem Taskverarbeitungssystem (KHS, HTV) auf Softwareebene. Zu den wesentlichen Aufgaben gehört die Deklaration von Taskparametern (Optimierungsintervall etc.). Auf der funktionalen Ebene umfasst die Bibliothek Service-Routinen, die für die Ablaufsteuerung von Tasks und zum Nachrichtenaustausch mit anderen Systemkomponenten (Tasks, HTV, KHS etc.)

verwendet werden. Des Weiteren beinhaltet die Bibliothek auch taskspezifische Schnittstellen-Deklarationen, die beispielsweise für die Implementierung eines individuellen Transfer- oder Sicherungsverhalten von Taskkomponenten genutzt werden können.

Driver Interface Library Diese API-Komponente übernimmt die Geräteanbindung der Hardwarekomponenten TPS*LiHaDe* und AHS*LiHaDe* auf Systemebene. Neben einem plattformabhängigen Gerätetreiber (verfügbar für Debian-, Gentoo-Linux/μLinux ab Kernel 2.6) verfügt die Bibliothek über ein funktionsorientiertes Schnittstellenkonzept, dass die Kommunikation mit den Hardwarekomponenten mittels Handles und Funktionsaufrufen (mit und ohne Rückgabewert) implementiert. Diese beinhalten im wesentlichen Routinen die für die Übertragung von Daten (Hormonen) und Befehlen (Starten) sowie für die Konfiguration von Parametern (H-Zykluszeit etc.) der Hardwarekomponenten verwendet werden.

5.2.3. Quantifizierung der V-Zykluszeit der HTV

Ergänzend erfolgt eine Quantifizierung der in Kapitel 4.1 analysierten Zeitschranken des V-Zyklus für den Hardwareteil TPS*LiHaDe* der vorgestellten HTV-Implementierung. Da das Zeitverhalten des V-Zyklus primär von der Vorlaufzeit $t_{\dot{Z}\dot{A}}$ beeinflusst wird, die sich durch den Transfer von Taskkomponenten definiert, ist die Bestimmung der Kommunikationszeit von Datenhormon maßgebend für die Quantifizierung. Des Weiteren wird für den sekundären Zeitanteil der Reaktionszeit $t_{S\dot{Z}}$ die H-Zykluszeit des KHS-Hardwareentwurfs aus Abschnitt 5.1.3 verwendet. Im Folgenden wird daher zunächst die Kommunikationszeit von Datenhormonen bei maximalen Transferverhalten innerhalb des TPS*LiHaDe* bestimmt. Daraus abgeleitet erfolgt die Betrachtung der maximalen V-Zykluszeit, sowie die Veranschaulichung der Ergebnisse anhand eines Beispiel-Szenarios mit konkreten Werten.

Für die Betrachtung des größtmöglichen Transferverhaltens der HTV, muss die Kommunikationszeit t_K gegenüber Datenhormonen genauer präzisiert werden. Diese unterscheidet sich von der ursprünglichen Definition der Steuerhormone des KHS:

Präzisierung der maximalen Kommunikationszeit von Datenhormonen: Aufgrund der Vielzahl und Größe benötigter Datenhormone innerhalb eines Transfers von Taskkomponenten, umfasst die größtmögliche Kommunikationszeit t_K bei Datenhormonen den Zeitraum für das Senden und Empfangen eines einzelnen Datenhormons bei globaler Verfügbarkeit der Komponenten und maximaler Netz-Distanz.

Die Zeitspanne t_K beschränkt sich also auf den Transfer eines einzelnen Datenhormons und beinhaltet dessen vollständige Verarbeitung durch die beiden Pipelines der DDT-Einheit des TPS*LiHaDe*. Aus dem sequentiellen Transfer

aller Datenhormone ergibt sich dann die größtmögliche Gesamtdauer für den Schritt Zustandstransfer und Instanzerzeugung bei globaler Verfügbarkeit von Komponenten.

Ausgehend von dieser Definition folgt die Analyse des Transferverhaltens des V-Zyklus. Zunächst einmal lässt sich die Gesamtanzahl der auf C_γ anfallenden Datenhormone in zwei Summen unterteilen:

$$\left.\begin{matrix} Ds_\gamma \\ De_\gamma \end{matrix}\right\} := b + z + k \tag{5.14}$$

$b :=$ Anzahl der Datenhormone für Taskbeschreibungen
$z :=$ Anzahl der Datenhormone für Taskzustände
$k :=$ Anzahl der Datenhormone für Taskkommunikation

- Die Summe Ds_γ in Gleichung 5.14 definiert die Gesamtanzahl an Datenhormonen, die während der Taskverarbeitung von C_γ gesendet werden.

- Die Summe De_γ in Gleichung 5.14 definiert die Gesamtanzahl an Datenhormonen, die während der Taskverarbeitung auf C_γ empfangen werden.

Beide Summen akkumulieren sich jeweils aus der Menge an Datenhormonen, die für den Transfer von Taskkomponenten (b,z), sowie zur Kommunikation (k) zwischen den Tasks im Netz gesendet und empfangen werden. Erneut wie in Abschnitt 5.1.3 werden die beiden Summen nun für den initialen Start- und den eingeschwungenen Endzustand betrachtet, um daraus die maximale Kommunikationszeit für ein Datenhormon abzuleiten.
Im Startzustand werden alle Tasks fortlaufend durch die Selbst-Konfiguration verteilt. Währenddessen befinden sich die Tasks noch nicht in der Ausführung. Es herrscht deshalb in der Anfangsphase keine Kommunikation zwischen den Tasks. Des Weiteren beschränkt sich der Austausch an Komponenten ausschließlich auf den Transfer von Taskbeschreibungen, da bei Systemstart noch keine lokalen Zustände für Tasks im Netz vorhanden sind (z,k=0). Aufgrund dessen sendet Kern C_γ für jede seiner initial gespeicherten Tasks $T_i \in L_\gamma$ die Taskbeschreibung an den Zielort der Taskübernahme. Im Gegenzug empfängt C_γ für jede übernommene Task $T_i \in E_\gamma \backslash L_\gamma$ die Beschreibung.

Anzahl der Datenhormone im Startzustand auf C_γ:

$$\left.\begin{aligned} {}_{Start}Ds_\gamma = b = \sum_{T_i \in L_\gamma} b_i \\ {}_{Start}De_\gamma = b = \sum_{T_i \in E_\gamma \setminus L_\gamma} b_i \end{aligned}\right\} \text{ z,k = 0 im Startzustand}$$

Im eingeschwungenen Endzustand ist die Selbst-Konfiguration beendet und es befinden sich alle Tasks in der Ausführung. Im Gegensatz zur Anfangsphase sendet Kern C_γ daher im Normalfall lediglich die Datenhormone die für Taskkommunikation benötigt werden. Da sich im Endzustand aber auch Tasks in der Selbst-Optimierung oder -Heilung befinden können, sendet C_γ für jede optimierte Task $T_i \in O_\gamma$ und für jede Task $T_i \in L_\gamma$ die sich in der Heilung befindet die Taskkomponenten. Dieser Transfer beinhaltet dann anders als im Startzustand nicht nur die Beschreibung von T_i, sondern nun auch deren aktuellen oder zuletzt gespeicherten Zustand. Zusätzlich kommuniziert jede übernommene Task T_i eine Menge von k_i Datenhormonen an alle v_i verwandten Tasks im Netz. Im Gegenzug empfängt T_i die Kommunikation seiner v_i verwandten Tasks. Des Weiteren empfängt C_γ für jede migrierte oder reallokierte Task $T_i \in E_\gamma$ die zugehörigen Komponenten.

Anzahl der Datenhormone im Endzustand auf C_γ:

$$\left.\begin{aligned} {}_{Ende}Ds_\gamma &= b + z + k \\ &= \sum_{T_i \in L_\gamma} b_i + z_i \quad + \sum_{T_i \in O_\gamma} b_i + z_i \quad + \sum_{T_i \in E_\gamma} v_i \cdot k_i \\ {}_{Ende}De_\gamma &= b + z + k \\ &= \sum_{T_i \in E_\gamma} b_i + z_i \quad + \sum_{T_i \in E_\gamma} v_i \cdot k_i \end{aligned}\right\} \text{ im Endzustand}$$

Unter Annahme einer ausschließlich globalen Verfügbarkeit von Taskkomponenten und der größtmöglichen Datenmenge von Taskkomponenten im Netz, kann für den Start- wie auch den Endzustand das maximale Aufkommen an Datenhormonen bestimmt werden.
Für den Startzustand gilt dann, dass Kern C_γ während der Selbst-Konfiguration für alle l_{max} initial gespeicherten Tasks die zugehörigen Beschreibungen mit je b_{max} Datenhormonen an den Zielort der Taskübernahme sendet. Gegenläufig empfängt C_γ für jeden seiner e_{max} übernommenen Tasks b_{max} Datenhormone.

Für das maximale Datenhormonaufkommen im Startzustand gilt deshalb:

$$
\begin{aligned}
{}_{Start}Ds_{max} &= l_{max} \cdot b_{max} \\
{}_{Start}De_{max} &= e_{max} \cdot b_{max}
\end{aligned}
\tag{5.15}
$$

$l_{max} := \max_{C_\gamma \in \Omega}\{l_\gamma\}$, Maximum initial/lokal gespeicherter Tasks
$e_{max} := \max_{C_\gamma \in \Omega}\{e_\gamma\}$, Maximum übernommener Tasks
$b_{max} := \max_{T_i \in M}\{b_i\}$, Maximum an Datenhormonen pro Taskbeschreibung

Für den Endzustand gilt, dass der größtmögliche Transferaufwand für alle o_{max} optimierten und l_{max} geheilten Tasks auf Kern C_γ mit $b_{max} + z_{max}$ Datenhormonen pro Task begrenzt wird. Zusätzlich verursacht jede Task $T_i \in E_\gamma$ bei v_{max} Taskverwandtschaften den größtmöglichen Kommunikationsaufwand. Des Weiteren empfängt C_γ im Endzustand schlimmsten Falls für jede Task $T_i \in E_\gamma$ die Komponenten plus dem zugehörigen Kommunikationsaufwand der durch die verwandten Tasks ausgeschüttet wird.

Das maximale Aufkommen an Datenhormonen im Endzustand beträgt somit:

$$
\begin{aligned}
{}_{Ende}Ds_{max} &= l_{max} \cdot (b_{max} + z_{max}) + o_{max} \cdot (b_{max} + z_{max}) + e_{max} \cdot v_{max} \cdot k_{max} \\
{}_{Ende}De_{max} &= e_{max} \cdot (b_{max} + z_{max}) + e_{max} \cdot v_{max} \cdot k_{max}
\end{aligned}
\tag{5.16}
$$

$l_{max} := \max_{C_\gamma \in \Omega}\{l_\gamma\}$, Maximum initial/lokal gespeicherter Tasks
$e_{max} := \max_{C_\gamma \in \Omega}\{e_\gamma\}$, Maximum übernommener Tasks
$v_{max} := \max_{T_i \in M}\{v_i\}$, Maximum an Task-Verwandtschaften
$o_{max} := \max_{C_\gamma \in \Omega}\{o_\gamma\}$, Maximum optimierter Tasks
$k_{max} := \max_{T_i \in M}\{k_i\}$, Maximum an Datenhormonen pro Taskkommunikation
$b_{max} := \max_{T_i \in M}\{b_i\}$, Maximum an Datenhormonen pro Taskbeschreibung
$z_{max} := \max_{T_i \in M}\{z_i\}$, Maximum an Datenhormonen pro Taskzustand

Nachfolgend kann nun die Berechnungsvorschrift für die maximale Kommunikationszeit eines einzelnen Datenhormon abgeleitet werden. Hierfür wird zunächst die Gesamtdauer der beiden Vorgänge Senden und Empfangen bei maximalem Hormonaufkommen in Start- und Endzustand betrachtet. Diese definiert sich aus der Summe der beiden Ausführungszeiten t_{Ds} und t_{De}, welche für die Pipeline basierte Verarbeitung der Datenhormone innerhalb der DDT-Einheit benötigt wird. Für die beiden Pipelines gilt eine kaskadierte Verarbeitung mit $K = 3$ Stufen

sowie die zugehörigen Zykluszeiten t_{CDR} und t_{CDS}[5]. Zusammenfassend kann die Kommunikationszeit dann für jeden Zustand mittels des Quotienten berechnet werden, der sich aus der Gesamtdauer der beiden Ausführungszeiten und dem Transfervolumen definiert. Für den Startzustand gilt, dass das Transfervolumen bei globaler Verfügbarkeit von Taskkomponenten mit b_{max} Datenhormonen pro Task begrenzt ist. Hingegen gilt im Endzustand ein höheres Transfervolumen mit maximal $b_{max} + z_{max}$ Datenhormonen pro Task.

Die maximale Kommunikationszeit für ein Datenhormon im Startzustand beträgt deshalb:

$$ {}_{Start}t_K = \frac{{}_{Start}t_{Ds} + {}_{Start}t_{De}}{b_{max}} \tag{5.17} $$

$$ {}_{Start}t_{Ds} = (3 + {}_{Start}Ds_{max} - 1) \cdot t_{CDR} $$
$$ {}_{Start}t_{De} = (3 + {}_{Start}De_{max} - 1) \cdot t_{CDS} $$

Die maximale Kommunikationszeit für ein Datenhormon im eingeschwungenen Zustand beträgt somit:

$$ {}_{Ende}t_K = \frac{{}_{Ende}t_{Ds} + {}_{Ende}t_{De}}{b_{max} + z_{max}} \tag{5.18} $$

$$ {}_{Ende}t_{Ds} = (3 + {}_{Ende}Ds_{max} - 1) \cdot t_{CDR} $$
$$ {}_{Ende}t_{De} = (3 + {}_{Ende}De_{max} - 1) \cdot t_{CDS} $$

Folglich gilt für die obere Schranke der Kommunikationszeit eines einzelnen Datenhormon im Netz das Maximum aus den beiden Zeiten ${}_{Start}t_K$ und ${}_{End}t_K$.

Worst Case Daten-Kommunikationszeit $t_K = max\{{}_{Start}t_K, {}_{Ende}t_K\}$ (5.19)

5.2.3.1. Abschließende Quantifizierung der V-Zykluszeit

Abschließend folgt die Quantifizierung der maximalen V-Zykluszeit des HTV-Hardwareentwurfs. Hierfür wird die Reaktionszeit $t_{S\dot{Z}}$ sowie die Vorlaufzeit $t_{\dot{Z}\dot{A}}$ aus Kapitel 4.1 gemäß den Ergebnissen aus den vorangegangenen Abschnitten ergänzt. Gegenüber der maximalen Reaktionszeit aus Gleichung 4.9 (Abschnitt 4.1.3.1), gilt für den Broadcast der Taskübernahme die minimale Kommunikationszeit des KHS-Hardwareentwurfs aus Gleichung 5.11. Für die maximale Vorlaufzeit aus Gleichung

[5]Die für die beiden Pipeline-Zykluszeiten t_{CDS} und t_{CDR} aufgeführten Werte sind unabhängig von der verwendeten System-Konfiguration (l_{max}, ϕ_{max} etc.) und wurden durch Portierung des TPS*LiHaDe* für eine XILINX Virtex-6 Zielplattform mit einer System-Taktfrequenz von 250 MHz ermittelt.

4.17 (Abschnitt 4.1.3.2) gilt für den sequentiellen Transfer der Datenmengen Db_{max} und Dz_{max} die Kommunikationszeit aus Gleichung 5.19. Zusätzlich kann die Zeitdauer $t_{\dot{E}\dot{A}}$ für den Startvorgang der Taskausführung von ursprünglich m_{max} Tasks auf e_{max} Tasks präzisiert werden. Für den Hardwareentwurf ergibt sich dann aus Gleichung 4.6 die folgende totale Worst-Case Zykluszeit.

Max. V-Zykluszeit
$$\begin{aligned} t_{VC} &= t_{S\dot{Z}} + t_{\dot{Z}} + t_{\dot{E}} + t_{\dot{E}\dot{A}} \\ &= max\{t_{Hs}, t_{Hr}\} + \\ &\quad + max\{{}_{Start}t_K, {}_{Ende}t_K\} \cdot (b_{max} + z_{max}) + e_{max} \cdot ts_{max} \end{aligned} \tag{5.20}$$

$$t_{S\dot{Z}} = max\{t_{Hs}, t_{Hr}\} + ts_{max} \qquad t_{\dot{Z}} = max\{{}_{Start}t_K, {}_{Ende}t_K\} \cdot z_{max}$$
$$t_{\dot{E}\dot{A}} = (e_{max} - 1) \cdot ts_{max} \qquad t_{\dot{E}} = max\{{}_{Start}t_K, {}_{Ende}t_K\} \cdot b_{max}$$

Beispiel für die Berechnung der maximalen V-Zykluszeit Zur Veranschaulichung der obigen Berechnungsvorschriften folgt die Betrachtung eines einfachen Beispiels. Hierfür wird das Beispiel-Szenario aus Abschnitt 5.1.3 (64 Tasks verteilt auf 64 Kerne) erneut aufgegriffen. Zu den ursprünglichen Grid-Parametern werden zusätzlich konkrete Werte für die maximalen Datenmengen Db_{max}, Dz_{max} und Dk_{max} sowie für die Anzahl lokal gespeicherter Tasks l_{max} angenommen. Zur besseren Übersichtlichkeit sind alle relevanten Parameter noch einmal aufgeführt:

$Db_{max} := 512$ KBytes,	Maximale Datenmenge einer Taskbeschreibung
$Dz_{max} := 128$ KBytes,	Maximale Datenmenge eines Taskzustandes
$Dk_{max} := 40$ KBytes/sec,	Maximale Taskkommunikation pro Sekunde
$l_{max} := 6$,	Maximum lokal gespeicherter Tasks pro Kern
$e_{max} := 2$,	Maximum an Taskübernahmen pro Kern
$v_{max} := 8$,	Maximum an Task-Verwandtschaften
$o_{max} := 1$,	Maximum zeitgleich, optimierter Tasks pro Kern

Um das Hormonaufkommen zu beziffern wird zunächst der Aufbau eines Datenhormons vorgestellt. Ähnlich zu den Verwaltungshormonen des KHS setzt sich ein Datenhormon aus einer eindeutigen Absenderkennung (Header) und den eigentlichen Transferdaten (D_p Payload) zusammen:

	1 Byte	Hormontyp
	1 Byte	Kern ID (x,y-Koordinate)
	1 Byte	Task ID
	509 Byte	Nutzdaten = D_p Payload
Σ	512 Byte	

Durch den obigen Aufbau kann nun die Anzahl an Datenhormonen (D) berechnet werden die für den Transfer der Komponenten und der Kommunikationsdaten einer

Task maximal benötigt wird.

$$
\begin{aligned}
b_{max} &:= \frac{512 \cdot 2^{10} \text{ Bytes}}{509 \text{ Bytes}} &&= 1031 \text{ D}, && \text{pro Taskbeschreibung} \\
z_{max} &:= \frac{128 \cdot 2^{10} \text{ Bytes}}{509 \text{ Bytes}} &&= 258 \text{ D}, && \text{pro Taskzustand} \\
k_{max} &:= \frac{40 \cdot 2^{10} \text{ Bytes/sec}}{509 \text{ Bytes}} &&= 81 \text{ D/sec}, && \text{pro Taskkommunikation}
\end{aligned}
$$

Durch einsetzen der Grid-Parameter in Gleichung 5.15 und 5.16 erhalten wir zunächst die folgenden Mengen an Datenhormonen die in Start- und Endzustand schlimmsten Falls transferiert werden müssen.

$$
\begin{aligned}
{}_{\boldsymbol{Start}}\boldsymbol{Ds_{max}} &= 6 \cdot 1031 = 6186 \text{ D} \\
{}_{\boldsymbol{Start}}\boldsymbol{De_{max}} &= 2 \cdot 1031 = 2062 \text{ D}
\end{aligned}
$$

$$
\begin{aligned}
{}_{\boldsymbol{Ende}}\boldsymbol{Ds_{max}} &= 6 \cdot (1031 + 258) + 1 \cdot (1031 + 258) + 2 \cdot 8 \cdot 81 = 10319 \text{ D} \\
{}_{\boldsymbol{Ende}}\boldsymbol{De_{max}} &= 2 \cdot (1031 + 258) + 2 \cdot 8 \cdot 81 = 3874 \text{ D}
\end{aligned}
$$

Durch weiteres einsetzen der berechneten Hormonmengen in Gleichung 5.17 und 5.17 lassen sich die Ausführungszeiten für die Verarbeitung der Datenhormone in den Pipelines (DSP, DRP) berechnen. Für die Pipeline-Zykluszeiten verwenden wir die in Kapitel 6 ermittelnden Werte der Zielplattform mit $t_{CDS} = 16.64\,\mu s$ und $t_{CDR} = 6.4\,\mu s$. Somit erhalten wir für die Kommunikationszeit in Start- und Endzustand die folgenden Werte.

$$
{}_{\boldsymbol{Start}}\boldsymbol{t_K} = \frac{102.97 \times 10^3\,\mu s + 13.21 \times 10^3\,\mu s}{1031 \text{ D}} = 112.69\,\mu s/\text{D}
$$

$$
{}_{\boldsymbol{Ende}}\boldsymbol{t_K} = \frac{171.74 \times 10^3\,\mu s + 24.81 \times 10^3\,\mu s}{1031 + 258 \text{ D}} = 152.48\,\mu s/\text{D}
$$

$$
\begin{aligned}
{}_{Start}t_{Ds} &= (3 + 6186 - 1) \cdot 16.64\ \mu s &&= 102.97 \times 10^3\,\mu s \\
{}_{Start}t_{De} &= (3 + 2062 - 1) \cdot 6.4\ \mu s &&= 13.21 \times 10^3\,\mu s \\
{}_{Ende}t_{Ds} &= (3 + 10319 - 1) \cdot 16.64\ \mu s &&= 171.74 \times 10^3\,\mu s \\
{}_{Ende}t_{De} &= (3 + 3874 - 1) \cdot 6.4\ \mu s &&= 24.81 \times 10^3\,\mu s
\end{aligned}
$$

Durch das Maximum in Gleichung 5.19 erhalten wir die obere Schranke für die Kommunikationszeit eines einzelnen Datenhormons im Netz.

$$
\boldsymbol{t_K} = max\{112.69, 152.48\}\ \mu s/\text{D} = 152.48\,\mu s/\text{D}
$$

Letztlich erhalten wir durch Einsetzen der Ergebnisse in Gleichung 5.20 die finale Worst-Case V-Zykluszeit für das aufgeführte Beispiel-Szenario. Für die Bedienzeit ts_{max} wurde ein Zeitscheibenintervall von 20 ms pro Task gewählt, das ebenfalls in Kapitel 6 verwendet wird.

$$\begin{aligned} \boldsymbol{t_{VC}} &= 100.95\,\mu s + 152.48\,\mu s/D \cdot (1031 + 258\ D) + \\ &\quad + 2 - 1 \cdot 20 \times 10^3\,\mu s \\ \\ &= 216.65 \times 10^3\,\mu s = 216.65\,ms \end{aligned}$$

Abschließend können mittels der errechneten Zykluszeiten t_{HC} und t_{VC} die Wort-Case Laufzeiten für die Selbst-X Eigenschaften aus Kapitel 4 angegeben werden.

Durch Einsetzen der Zykluszeiten in Gleichung 4.22 erhalten wir für eine klassische oder aggressive Selbst-Konfiguration die folgenden Laufzeiten:

$$\textbf{Selbst-Konfiguration} := \begin{cases} (32+8) \cdot 0.31\,ms + 216.65\,ms & = 229.05\,ms : \text{klassisch} \\ 32 \cdot 0.31\,ms + 216.65\,ms & = 195.28\,ms : \text{aggressiv} \end{cases}$$

Für die obere Schranke der Selbst-Optimierung einer oder aller Tasks ergeben sich mittels Gleichung 4.33 die folgenden Laufzeiten:

$$\textbf{Selbst-Optimierung} := \begin{cases} 0.31\,ms + 216.65\,ms & = 185.67\,ms : \text{eine Task} \\ (32+8) \cdot 0.31\,ms + 216.65\,ms & = 197.76\,ms : \text{alle Tasks} \end{cases}$$

Mittels Gleichung 4.36 und 4.37 erhalten wir für die reaktive und proaktive Selbst-Heilung aller Tasks des Beispiel-Szenarios die folgenden Laufzeiten. Für die reaktive Heilung gilt eine Hormon-Verfallszeit mit $a = 1$ sowie eine Task-Antwortzeit mit $b = 1$.

$$\textbf{Selbst-Heilung} := \begin{cases} (1+32+8) \cdot 0.31\,ms + & = 229.46\,ms : \text{Reaktiv} \\ \quad +216.65\,ms + 0.1\,ms & \\ (32+8) \cdot 0.31\,ms + 216.65\,ms & = 229.05\,ms : \text{Proaktiv} \end{cases}$$

6. Messungen auf der SoC Zielplattform

6.1. Vorstellung der Zielplattform

Zu Test- und Evaluationszwecken wurde ein funktionsfähiger Prototyp erstellt, der einen vollwertigen Systembetrieb von unterschiedlichen SoC-Szenarien in einer rekonfigurierbaren Hardwareumgebung ermöglicht. Für dessen Umsetzung wurden die vorgestellten Implementierungen aus Kapitel 5 auf eine FPGA-Zielplattform[1] portiert. Alle Systemkomponenten des Prototyps werden daher ausschließlich durch den Einsatz synthesefähiger VHDL-Modelle implementiert. Die Ausnahme bilden Hardwarekomponenten (Speichermodule, I/O-Controller etc.) die vorgefertigt auf der Zielplattform bereitgestellt werden.

Für den Testbetrieb wird ein Szenario bestehend aus einer Menge von ω Kernen in einem NxM Grid instantiiert und durch zwei serielle Verbindungsnetzwerke miteinander vernetzt. Abbildung 6.1 zeigt den strukturellen Aufbau eines beispielhaften 6x6 SoC-Szenario mit insgesamt 36 Kernen auf der Zielplattform. Die instantiierten Kerne werden durch einen Ressourcen schonenden VHDL-Mikroprozessor abgebildet, welcher eigens im Zuge dieser und weiterer Forschungsarbeiten ([vRSH+15]) als Soft-Core entwickelt wurde und als μSoC^3 (**M**icro **S**ystem on **C**hip **C**omputing **C**ore) bezeichnet wird. Es handelt sich dabei um eine leistungsfähige 32bit RISC-Architektur die individuell den Systemanforderungen angepasst wird und wahlweise über eine skalare bis superskalare Befehlsverarbeitung verfügt. In Publikation [BLBB13] wurde eine Erweiterung für mehrfädige Befehlsverarbeitung zur Steigerung der Durchsatzraten vorgestellt. Die Mikroarchitektur wurde für den Einsatz minimalistischer Linux-Distributionen ab Kernelversion 2.6 optimiert. Während des Testbetriebs erfolgt auf allen Kernen der Einsatz eines μCLinux- oder CoreLinux-Derivats. Die Bereitstellung lokaler KHS- und HTV-Instanzen erfolgt uniform durch die beiden Schaltwerke AHS*LiHaDe* und TPS*LiHaDe*. Ein hybrider KHS-Betrieb durch Hardware- und Softwareinstanzen (AHS*lib*) ist aber dennoch möglich und wird in Abschnitt 6.3 evaluiert. Die Kommunikation und Vernetzung zwischen den Systemkomponenten erfolgt über die Schnittstellenbibliotheken der AHS-API. Der Transfer von Steuer- und Datenhormonen erfolgt getrennt über zwei serielle Punkt-

[1] Die Zielplattform des Prototyps umfasst das Xilinx Virtex-6 C6VLX240T FPGA, integriert auf dem ML605 Evaluation-Board mit einem Systemtakt von 250 MHz.

zu-Punkt Verbindungsnetzwerke (C-SCN, D-SCN). Das senden von Hormonen erfolgt auf jedem Kern zugriffsunabhängig und störungsfrei über zwei exklusive Datenleitungen (C-Lane, D-Lane). Zeitgleich erfolgt das Empfangen durch passive Überwachung der Transfers auf den Datenleitungen der restlichen $\omega - 1$ Kerne. Somit skaliert der Gesamtaufwand für ein beliebiges SoC-Szenario linear mit einer Anzahl von 2ω Datenleitungen. Dies erlaubt eine maximale Transferrate von 250Mbit/s pro Datenleitung und garantiert die Einhaltung der in Kapitel 5 getätigten Aussagen zur maximalen Kommunikationszeit t_K von Steuer- und Datenhormonen im Netz.

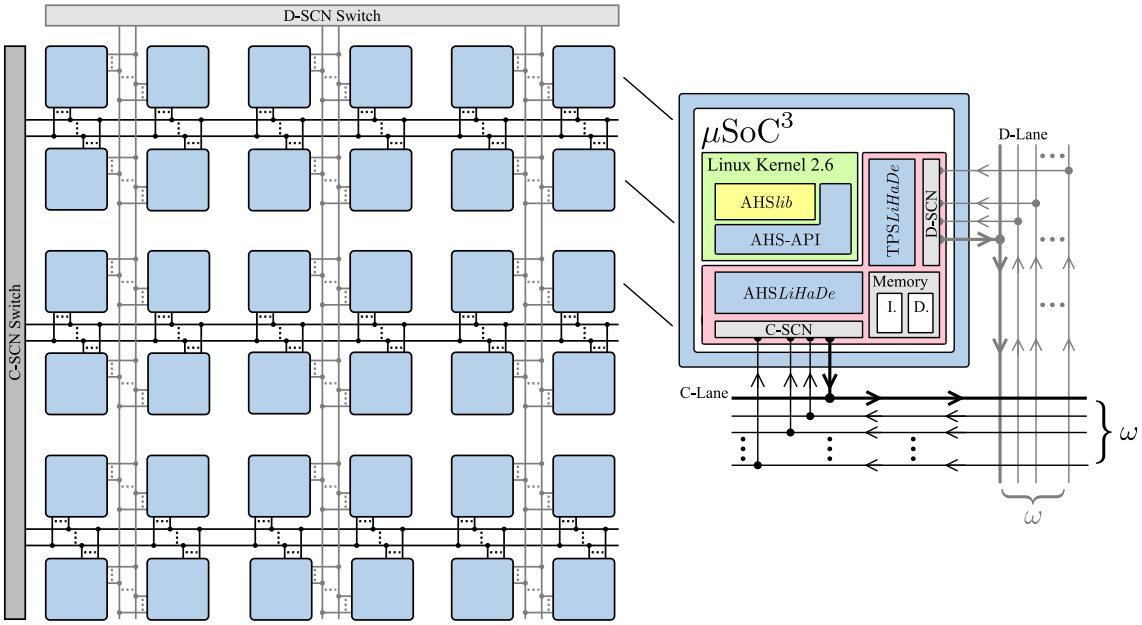


Abbildung 6.1.: SoC-Struktur auf der FPGA-Zielplattform

Um die Selbst-Heilung und Selbst-Optimierung des Systems mittels des Prototypen evaluieren zu können, müssen für jedes Szenario zeitliche Ereignisse definiert werden, die zu Ausfällen oder zur Verschlechterung einzelner Kerne an bestimmten Zeitpunkten führen. Hierfür wird auf jedem Kern ein zugehöriger Event-Handler erzeugt, der durch die Basisklassen der AHS-API bereitgestellt wird und die Überwachungssignale (Temperatur, Taskfehler) der lokalen Monitoreinheiten zunehmend negativ beeinflusst. So kann beispielsweise ein kritischer Anstieg der Kerntemperatur oder die Zunahme von Taskfehlern innerhalb der Ausführung simuliert werden, was letztlich zum Anstieg des zugehörigen Monitor-Suppressors führt. Die hierfür verwendeten Fehlermodelle sind in Anhang A.1 aufgeführt. Für die Erhebung der Messreihen in den Abschnitten 6.3, 6.4 und 6.5 wurden verschiedene SoC-Szenarien miteinander verknüpft und nacheinander auf dem Prototypen evaluiert. Diese werden im folgenden Abschnitt nun genauer vorgestellt.

6.2. Vorstellung der Evaluationsszenarien

Zur Evaluation der Selbst-X Eigenschaften werden verschiedene Szenarien mit unterschiedlichen Gridkonfigurationen verwendet. Jedes Szenario berücksichtigt eine unterschiedliche Anzahl an Kernen im Netz (ω), für das eine Menge von Tasksets mit verschiedenen Größen (m) validiert wird. Die Wahl der Gridparameter erfolgt unter der Annahme, dass sich jeder Kern für jede Task eines Sets eignet und interessiert ($m_{max} = m$). Die Anzahl lokaler Monitoring-Einheiten (l_{max}) wurde pro Kern auf zwei (Temperatur, Taskfehler) begrenzt.

Grid	Kerne (ω)
2 x 2	4
3 x 3	9
4 x 4	16
4 x 5	20
5 x 5	25
6 x 5	30
6 x 6	36

Für die oben aufgeführten Szenarien werden Tasksets mit den folgenden Größen und einem maximalen Verwandtschaftsgrad von $v_{max} = 8$ pro Task berücksichtigt.

Set (m)	Verwandtschaftsgrad (v_{max})
16	8
24	8
32	8
40	8
48	8
56	8
64	8

Des Weiteren gelten für die Taskkomponenten und die Taskkommunikation die folgenden Größen an Datenmengen und Datenraten im Netz.

Taskbeschreibung	(Db)	128-512 KBytes
Taskzustand	(Dz)	64-128 KBytes
Taskkommunikation	(Dk)	20-40 KBytes/sec

Zur Demonstration der Zuverlässigkeit gegenüber Task- und Kernausfällen wurde eine maximale Ausfallsicherheit des gesamten Systems von bis zu 50% vorgesehen.

Dies bedeutet, das die Anzahl tolerierbarer Kernausfälle ω_f in jedem Szenario der Hälfte der zugehörigen Grid-Größe ω entspricht.

$$\text{Max. Anzahl tolerierbarer Kernausfälle} \quad \omega_f = \left\lceil \frac{\omega}{2} \right\rceil$$

Für die Obergrenze lokaler Taskübernahmen e_{max} gilt daher die folgende Berechnungsvorschrift.

$$\text{Max. Anzahl lokaler Taskübernahmen} \quad e_{max} = \left\lceil \frac{m}{\omega - \omega_f} \right\rceil$$

Des Weiteren ist für die Gewährleistung der oben genannten Ausfallsicherheit eine redundante Sicherung aller Taskbeschreibungen im Netz erforderlich. Dadurch wird sichergestellt, dass bei jedem Systemausfall dennoch jede Taskbeschreibung mindestens einmal im Netz lokal gespeichert und im System global verfügbar ist. Hierfür werden die Tasks zusätzlich und unabhängig von der initialen Speicherung ω_f mal im Netz gesichert. Dabei gilt es zu berücksichtigen, dass die Sicherung gleichermaßen zur initialen Speicherung homogen auf allen Kernen im Netz verteilt erfolgt und sich die lokalen Taskmengen L_γ und R_γ voneinander disjunkt unterscheiden. Dabei definiert L_γ wie bereits in Kapitel 5.2 aufgeführt, die initial gespeicherte Taskmenge von Kern C_γ. Die Sicherungsmenge R_γ hingegen definiert die Taskmenge, die aufgrund der redundanten Sicherung zusätzlich von C_γ gespeichert wird. Abbildung 6.2 zeigt eine Veranschaulichung der redundanten Sicherung aller Taskbeschreibungen für das 2x2 Szenario und einem Set von 16 Tasks. Im Falle einer reaktiven Selbst-Heilung

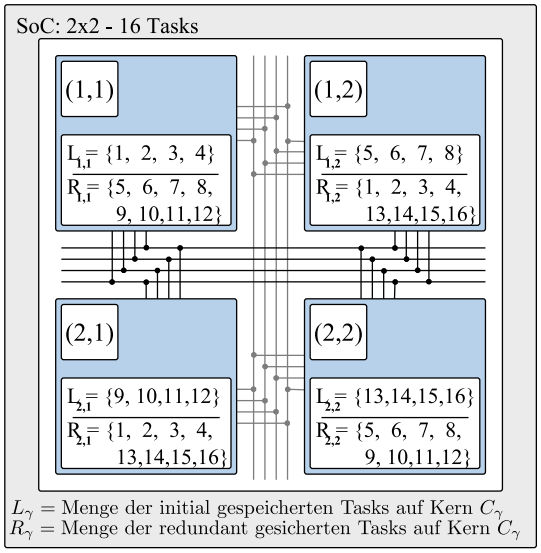


Abbildung 6.2.: Taskredundanz für ein 2x2 Szenario mit 16 Tasks

von Task T_i gilt dann, dass das Reaktionsverhalten des V-Zyklus schlimmsten Falls der doppelten Reaktionszeit $t_{S\dot{Z}}$ in Gleichung 4.36 entspricht ($b = 1$). Dies ist genau dann der Fall, wenn die initiale Speicherung von T_i im System durch den Ausfall von Kern Cb_i verloren geht und die erneute Bereitstellung der Beschreibung aus der Sicherungsmenge von T_i erfolgt. Der Transfer erfolgt dann nach Ablauf der regulären Reaktionszeit $t_{S\dot{Z}}$. Dies soll sicherstellen, dass die Bereitstellung der Beschreibung auch bei reaktiver Selbst-Heilung in erster Instanz durch Kern Cb_i erfolgt, sofern dieser vom Ausfall nicht betroffen ist. Gleichermaßen erfolgt die Sicherung von optionalen Taskzuständen im Netz, falls die Tasks ein solches Sicherungsverhalten benötigen. Dies ist innerhalb der aufgeführten Szenarien aber nicht der Fall, weshalb bei reaktiver Selbst-Heilung auf den Transfer von Zustandsdaten verzichtet wird.

6.3. Messung der Selbst-Konfiguration

Für die Messung der Selbst-Konfiguration wurden alle Szenarien mit allen Tasksets aus Abschnitt 6.2 kombiniert und auf dem Prototypen ausgewertet. Für jeden Testbetrieb wurden die Zeitpunkte erfasst, wenn eine Task aus dem jeweiligen Set nach Systemstart die Ausführung erreicht. Die Selbst-Konfiguration gilt als beendet wenn auch die letzte Task eines Sets die Taskausführung erreicht. Nach Abschluss der Messreihen wurden die aufgezeichneten Monitoring-Daten aus der Zielplattform ausgelesen und analysiert.

Abbildung 6.3 zeigt die Ergebnisse aller Messreihen die zur besseren Darstellung durch eine Oberfläche interpoliert wurden. Zu sehen ist die zeitliche Gesamtdauer der Selbst-Konfiguration, die zusammengefasst für alle Szenarien in Tabelle 6.1 aufgeführt ist. Wie farblich zu erkennen ist, liegt die Mehrzahl aller Messungen (blaugefärbte Punkte) unter einer Gesamtdauer von 100 Millisekunden (rot bis gelblicher Farbbereich). Lediglich 1/4 aller Messungen führen zu einem Zeitverhalten von mehr als 200 Millisekunden. Hervorzuheben sind die Messreihen für das 2x2 Szenario mit insgesamt 4 Kernen. Aufgrund der höheren Taskanzahl die jeder Kern in diesem Szenario bei zunehmenden Taskset ausführen muss (e_{max}=16), nimmt die Kommunikationszeit für Steuer- und Datenhormone stark zu, so dass ab einer Set-Größe von 48 Tasks die Kurve abrupt beginnt steil anzusteigen. Hingegen ist der Kurvenverlauf bei den restlichen Szenarien von einem linearen Anstieg geprägt und verhält sich moderat gegenüber der Anzahl der zu verarbeitenden Tasks.

Abbildung 6.4 zeigt den Vergleich zu den in Kapitel 4.2 und 5 abgeleiteten Worst-Case Zeitschranken. Wie erwartetet, wurden für alle Messreihen die oberen Schranken eingehalten.

Abbildung 6.5 zeigt die Ergebnisse für einen hybriden Testbetrieb mit unterschiedlichen KHS-Instanzen im Netz. Hierfür wurden in allen Szenarien die Hälfte aller lokalen KHS-Instanzen durch Softwareinstanzen der Bibliothek AHS*lib* ersetzt. Anders als bei dem in Abbildung 6.3 gezeigten Testbetrieb mit ausschließlich Hardwarein-

stanzen des AHS*LiHaDe*, ist die Gesamtdauer der Selbst-Konfiguration in Tabelle 6.2 bei allen Messreihen um ein Vielfaches höher als in Tabelle 6.1. Dies liegt daran, dass die Gesamtdauer bei hybridem Betrieb von der deutlich größeren H-Zykluszeit der AHS*lib* auf Systemebene dominiert wird. Dies ist auf die höhere Rechen- und Kommunikationszeit von Steuerhormonen auf Systemebene zurückzuführen.

Abbildung 6.6 zeigt zur besseren Verdeutlichung den Vergleich zwischen einem reinen Hardware und hybriden KHS-Testbetrieb für das 6x6 Szenario mit 64 Tasks. Zu sehen ist die Zeitdauer der Taskverwaltung (KHS) gegenüber der Taskverarbeitung (HTV). Die pinke Kurve zeigt den zeitlichen Verlauf der KHS-Taskverteilung. Die Blaue den Verlauf der Taskausführung seitens der HTV. Die Selbst-Konfiguration ist genau dann beendet, wenn sich beide Kurven im Schnittpunkt treffen (grüne Linie). Bei der hybriden Variante ist zu sehen, dass die Gesamtdauer weitestgehend von der Zeitspanne bis zum Erreichen der KHS-Taskverteilung (rote Linie) dominiert wird (H-Zykluszeit 95 ms). Hingegen ist bei der reinen Hardwarevariante die Taskverteilung bereits nach 26 Millisekunden beendet, weshalb die Gesamtdauer von der HTV-Taskausführung dominiert wird (H-Zykluszeit 0.16 ms).

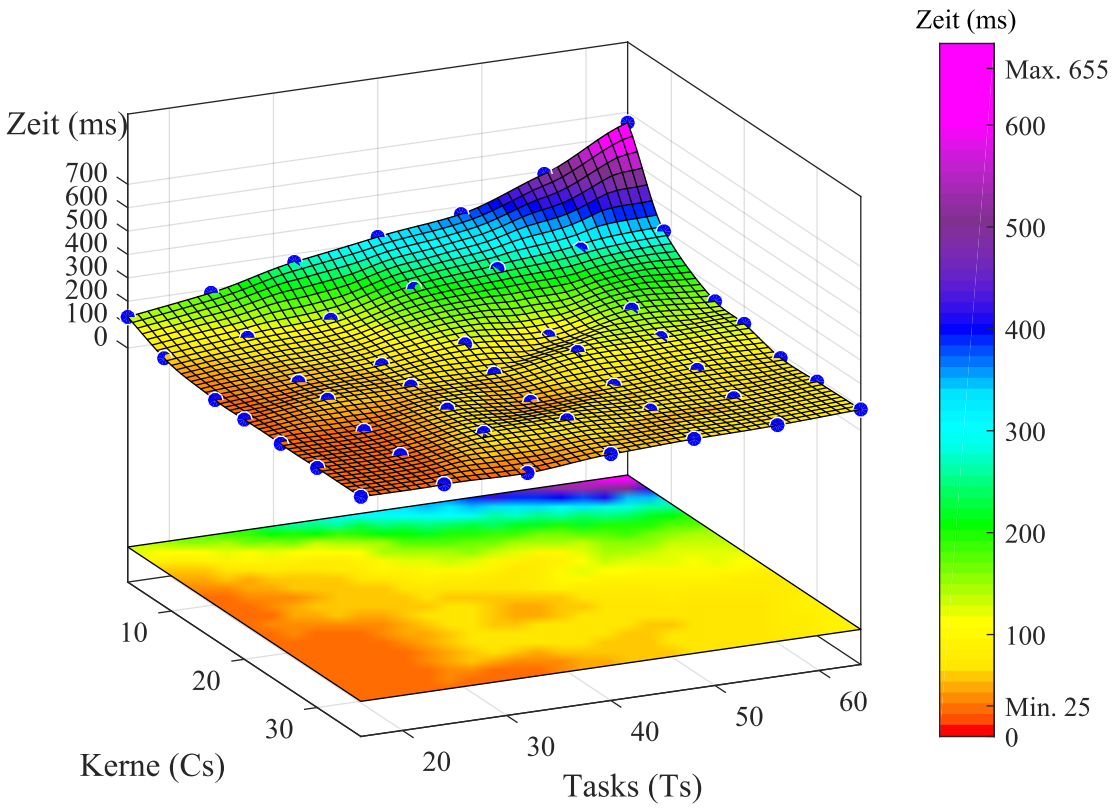


Abbildung 6.3.: Messreihen für die Gesamtdauer der Selbst-Konfiguration

Kerne \ Tasks	16	24	32	40	48	56	64
2x2	133,01	184,79	265,47	321,29	367,44	487,96	655,54
3x3	59,63	95,65	121,66	202,22	235,87	268,18	294,18
4x4	25,55	50,31	71,76	109,70	91,76	155,54	139,54
4x5	24,26	57,91	63,17	65,63	108,12	116,39	125,63
5x5	23,44	26,93	67,50	48,97	66,54	83,35	82,62
6x5	23,80	26,96	70,69	73,67	63,99	65,20	85,19
6x6	24,13	25,99	23,16	52,53	66,46	73,94	90,04

Zeit (ms)

Tabelle 6.1.: Messwerte für die Gesamtdauer der Selbst-Konfiguration in Abbildung 6.3

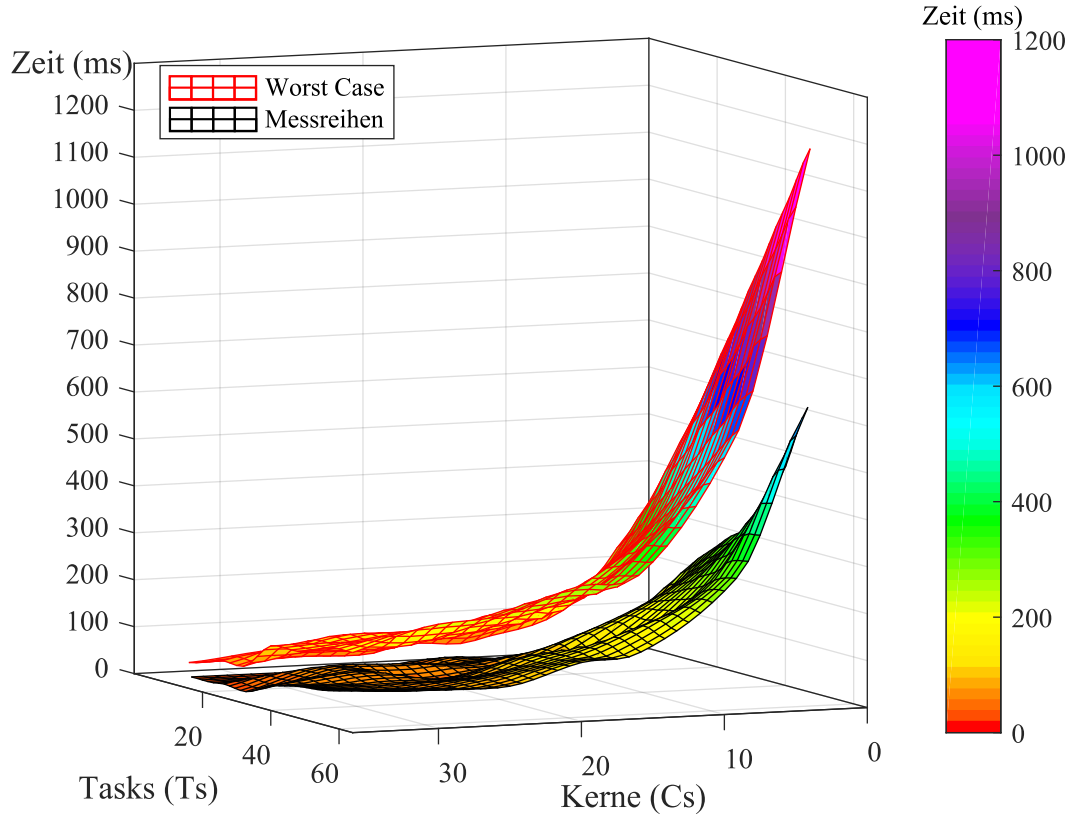


Abbildung 6.4.: Zeitschranken für die Selbst-Konfiguration (rot) vs. Messreihen (schwarz)

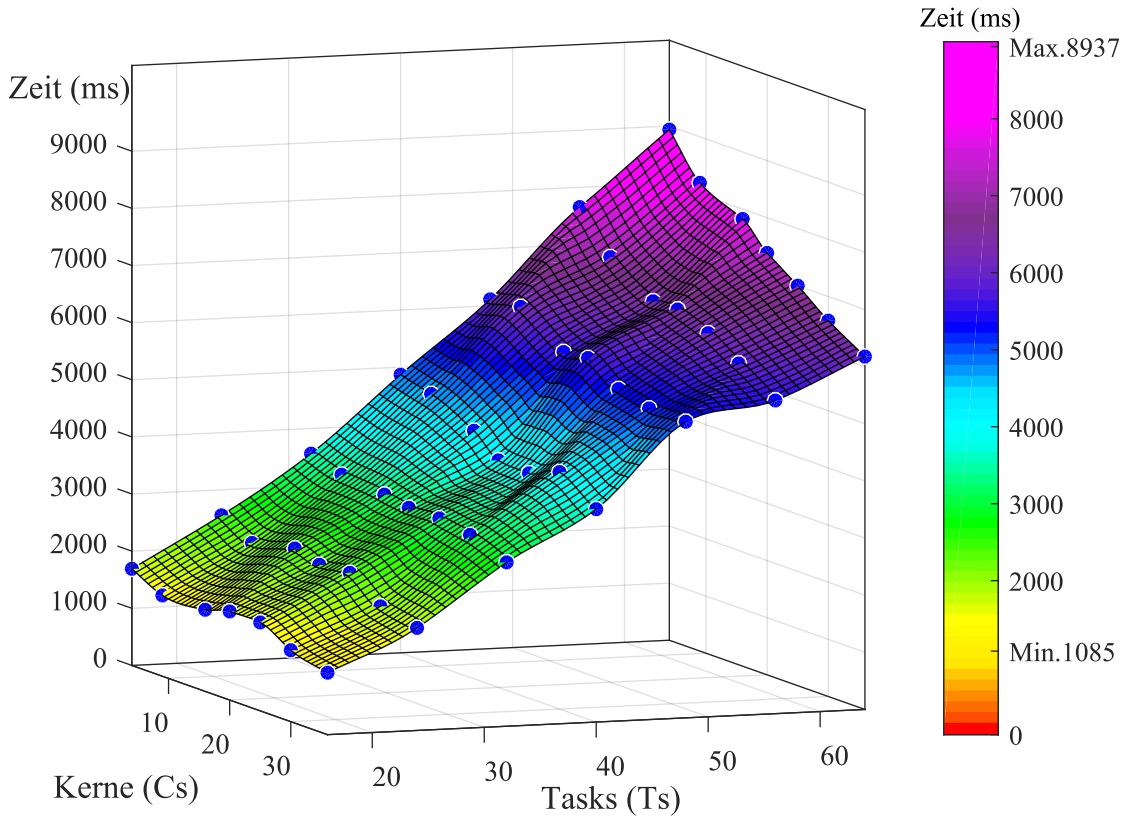
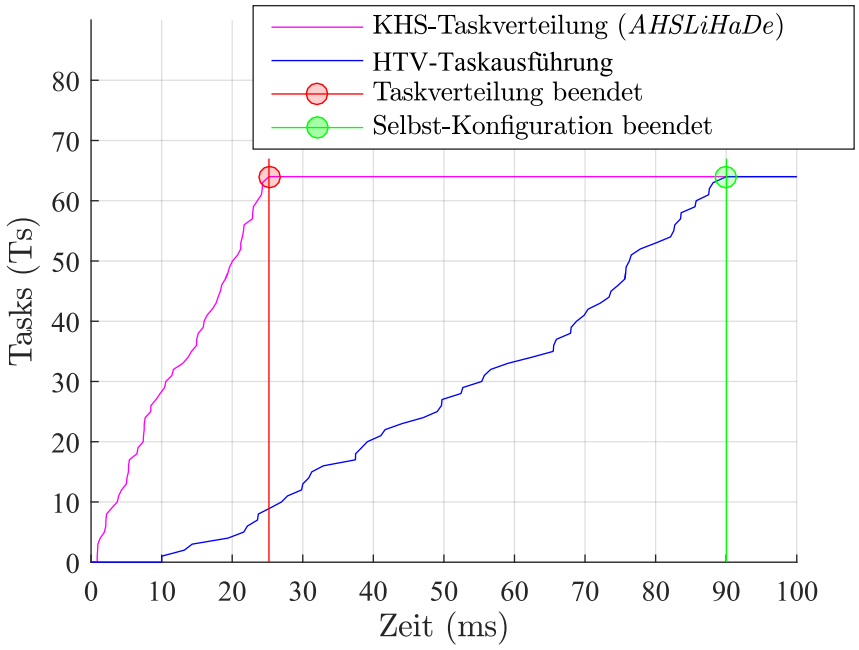


Abbildung 6.5.: Messreihen für die Selbst-Konfiguration mit hybrider KHS-Implementierung

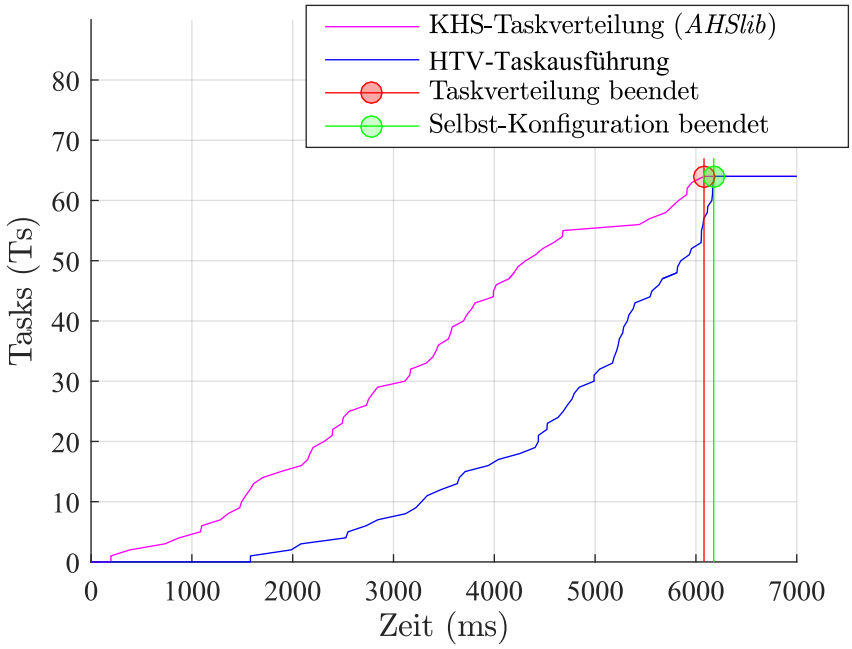
Kerne \ Tasks	16	24	32	40	48	56	64
2x2	1690,82	2553,71	3559,71	4868,61	6112,96	7654,71	8936,83
3x3	1414,40	2258,96	3381,12	4719,73	6170,10	6968,34	8194,54
4x4	1428,83	2427,08	3297,37	4340,70	5645,34	6459,87	7827,10
4x5	1550,75	2294,13	3217,86	3974,81	5692,20	6467,79	7383,92
5x5	1547,27	2341,53	3222,04	3937,28	5338,50	6242,40	6999,72
6x5	1250,81	1945,58	3121,55	4148,01	5188,18	5902,33	6578,01
6x6	1084,03	1795,34	2867,91	3725,96	5182,34	5481,01	6176,40

Zeit (ms)

Tabelle 6.2.: Messwerte für die Gesamtdauer der Selbst-Konfiguration in Abbildung 6.5



(a) Hardware KHS-Testbetrieb



(b) Hybrider KHS-Testbetrieb

Abbildung 6.6.: Gegenüberstellung Hardware vs. hybridem KHS-Testbetrieb für das 6x6 Szenario mit 64 Tasks

6.4. Messung der Selbst-Heilung

Zur Demonstration der Selbst-Heilung wird ein Ausfall von 50% des gesamten Systems simuliert. Dabei werden während des Testbetriebs nacheinander 2 Ausfälle impliziert, bei denen zunächst 25% und anschließend 50% aller Kerne des Prototyps deaktiviert werden. Dies führt dazu, dass die durch den H-Zyklus gesendeten Übernahme-Suppressoren der betroffenen Tasks im System wegfallen, weshalb diese nach Ablauf der Hormon-Verfallszeit in Gleichung 4.36 (a = 1 x H-Zykluszeit) neu umverteilt werden. Durch das redundante Sicherungskonzept aus Abschnitt 6.2 ist nach jedem Ausfall sichergestellt, dass die Taskinstanzen unabhängig von der Permutation der ausgefallenen Kerne nach erneuter Verteilung erneut lokal wiederhergestellt und ausgeführt werden können. Die Sicherung von Zuständen wird dabei nicht berücksichtigt, weshalb schlimmsten Falls bei reaktiver Heilung einer Task erneut die zugehörige Beschreibung zum Zielort transferiert werden muss, bevor die Taskinstanz innerhalb des V-Zyklus neu erzeugt und ausgeführt werden kann. Die genaue Anzahl an Tasks die durch den ersten und zweiten Ausfall affektiert werden ist je nach Szenario und Größe des Tasksets unterschiedlich. Dies hängt damit zusammen, dass die Anzahl der lokal ausgeführten Tasks auf einem ausfallbetroffenen Kern C_γ dynamisch ist und maximal $e_\gamma = e_{max}$ entspricht (siehe Abschnitt 6.2). Eine maximale Übernahme von e_{max} Tasks pro Kern tritt aber erst nach Ablauf und Heilung des zweiten Ausfalls ein, da die Last-Suppressoren so gewählt wurden, dass zu jedem Systemzeitpunkt seitens des KHS eine homogene Verteilung aller Tasks auf die verbleibenden Kerne des Prototyps erfolgt. Die Anzahl der ausfallbehafteten Tasks ist daher bei beiden Ausfällen deutlich geringer als e_{max}. Zur einfacheren Illustration werden lediglich die Messergebnisse für ein Set mit 64 Tasks gezeigt. Die Anzahl der ausfallbetroffenen Tasks und Kerne ist in der folgenden Tabelle aufgeführt. Für die anderen Tasksets ergeben sich ähnliche Werte.

Grid	Ausfall 1:25% (Ts-Cs)	Ausfall 2:50% (Ts-Cs)
2x2	16-1	22-1
3x3	24-3	22-2
4x4	16-4	24-4
4x5	20-5	25-5
5x5	21-7	24-6
6x5	24-7	21-8
6x6	18-9	27-9

Abbildung 6.7 zeigt das Zeitverhalten der reaktiven Selbst-Heilung als 3D-Darstellung, wobei die einzelnen Messreihen der Szenarien durch farblich gekennzeichnete Kurven eingezeichnet wurden. Zu sehen sind die beiden Taskausfälle zum Zeitpunkt t=1000 ms (grüne Ebene) und t=3000 ms (rote Ebene). Die Ausfall- und

Heilungsphase ist als rot-gelb gefärbter Farbbereich gekennzeichnet. Sobald die Kurve eines Szenarios nach Eintritt eines Ausfalls die Ebene mit 64 Tasks erreicht, befindet sich das System erneut in einem funktionsfähigen Konfigurationszustand bei dem alle Tasks ausgeführt werden. Zu erkennen ist, dass sich das Zeitverhalten des zweiten Ausfalls gegenüber dem ersten durch eine längere Gesamtdauer unterscheidet. Dies lässt sich einerseits auf die Schwere des zweiten Ausfalls zurückführen, bei dem die Anzahl der ausfallbehafteten Tasks insgesamt höher ist. Des Weiteren lässt sich dieses Verhalten auf das gesteigerte Datenaufkommen pro Kern zurückführen. Da die Anzahl der lokal ausgeführten Tasks wie erwartet mit jedem Ausfall zunimmt, nimmt auch der Transferaufwand seitens der Taskkommunikation lokal zu. Dies führt zu einer Verschlechterung der Kommunikationszeit von Datenhormonen und letztlich der V-Zykluszeit.

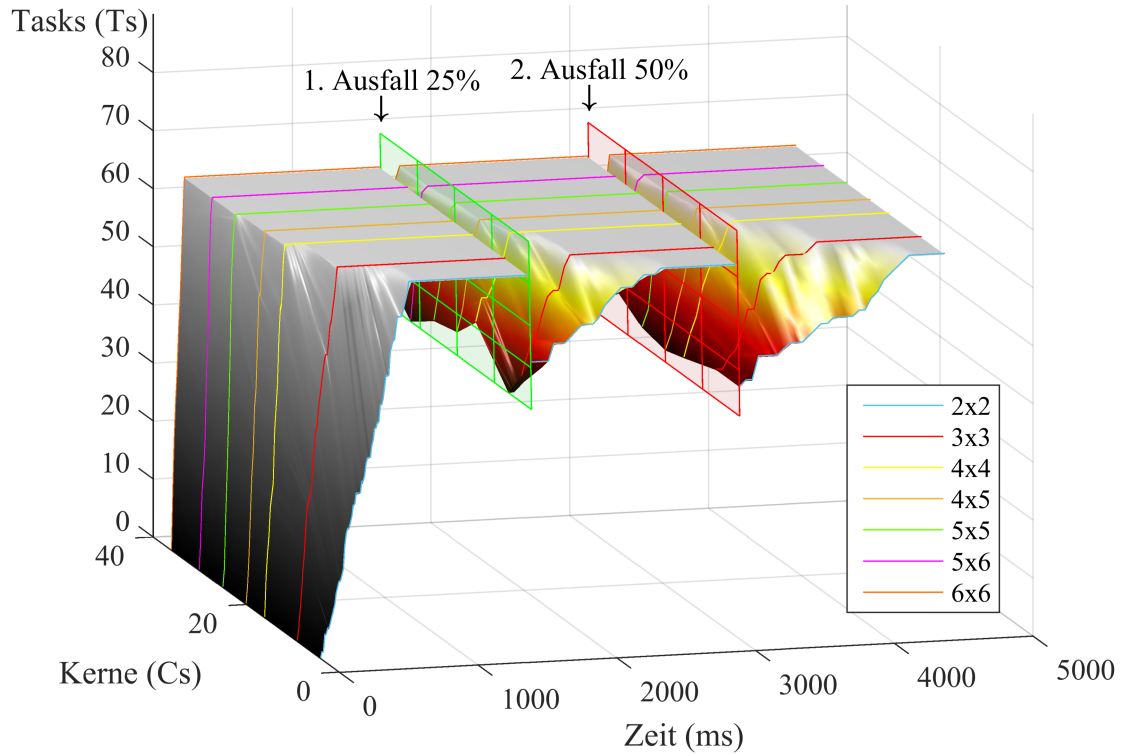


Abbildung 6.7.: Messreihen für die reaktive Selbst-Heilung

Abbildung 6.8 verdeutlicht den Sachverhalt durch den Vergleich der Zeitspannen für die Selbst-Heilung während des ersten und zweiten Ausfall. Die Zeitdauer einer Heilungsphase wurde jeweils durch einen farblichen Balken hervorgehoben. Zusätzlich wurden die oberen Zeitschranken der Selbst-Heilung aus Kapitel 4.4 für jede Messreihe durch einen grünen und roten Punkt auf der Messlinie eingezeichnet. Wie erwartet wurden alle Zeitschranken eingehalten. Tabelle 6.3 zeigt die Zusammenfassung aller Messwerte für die Zeitdauer aller Heilungsphasen.

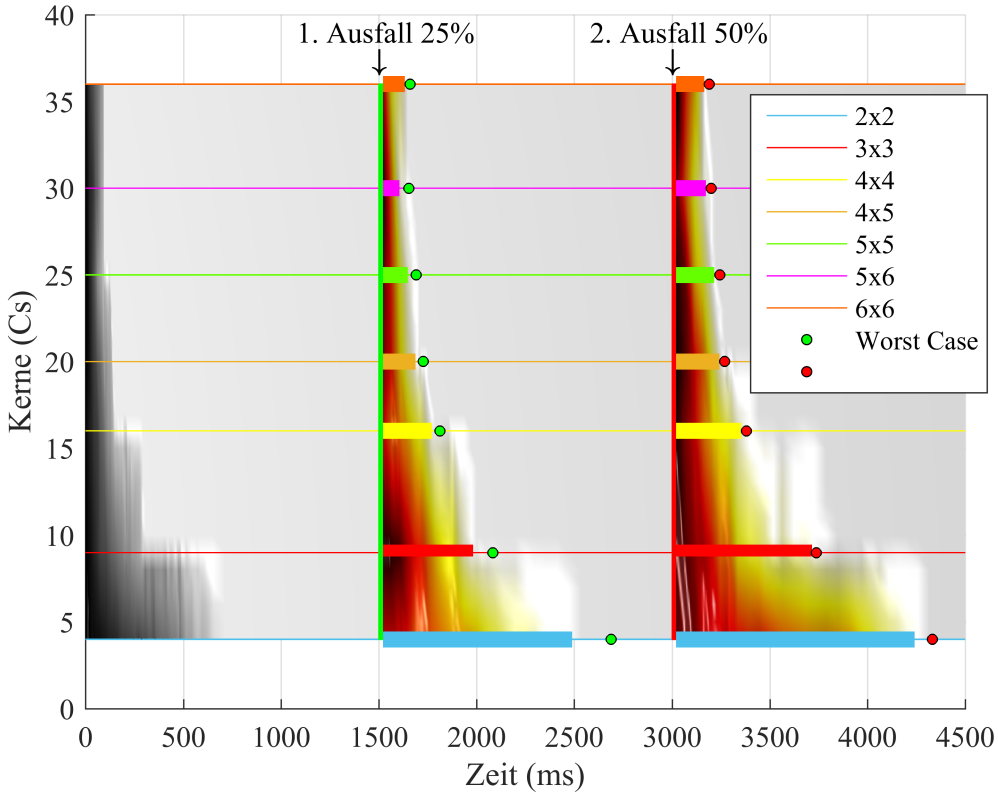


Abbildung 6.8.: Zeitdauer für die reaktive Selbst-Heilung

Kerne	Ausfall 25%	Ausfall 50%
2x2	989,26	1241,01
3x3	483,26	715,24
4x4	270,07	352,84
4x5	187,08	240,72
5x5	148,36	214,66
6x5	105,01	171,75
6x6	133,64	164,86

Zeit (ms)

Tabelle 6.3.: Messwerte für die Zeitdauer der reaktiven Selbst-Heilung in Abbildung 6.8

6.5. Messung der Selbst-Optimierung

Für die Messung der Selbst-Optimierung wird eine dauerhafte Verschlechterung von 50% des gesamten Systems simuliert. Hierfür werden während des Testbetriebs zwei lokale Monitoring-Suppressoren gesendet, welche die Eignung der betroffenen Kerne um 50% verschlechtert. Die Emission der Monitoring-Suppressoren erfolgt auf den Kernen des Prototyps zeitlich uniform und erst nach Ablauf der Selbst-Konfiguration. Dadurch entsteht ein Ungleichgewicht in den Hormonpegeln der verschlechterten Kerne, wodurch deren Tasks zunehmend auf die besser geeigneten des Systems umverteilt werden. Die Optimierung erfolgt periodisch innerhalb eines statisch definierten Optimierungsintervalls, dessen Zeitdauer der minimalen Intervallänge aus Kapitel 4.3.0.5 entspricht. Daraus ergibt sich eine Sequenz aus Optimierungsintervallen $I_1 ... I_n$ bis die Taskzuordnung entsprechend der System-Verschlechterung optimiert wurde und erneut den stabilen Zustand erreicht. Die Anzahl benötigter Optimierungsintervalle n wird beherrscht von der Gesamtmenge der Tasks die in Folge der Verschlechterung umverteilt werden, sowie von der Taskanzahl die pro Intervall angeboten und optimiert wird. Um einen vollständigen Abzug aller Tasks von den schlechteren Kernen vorzubeugen, wurden die Last-Suppressoren so erhöht, dass schlimmsten Falls 50% der betroffenen Tasks auf die besser geeigneten Kerne umverteilt werden, bevor das Ungleichgewicht in den Hormonpegeln ausbalanciert wird. Dies entspricht, unter der Berücksichtigung der herbeigeführten Verschlechterung, einer größtmöglichen Optimierung des gesamten Tasksets von 30%. Dementsprechend wurde die lokale Übernahmekapazität e_{max} aus Abschnitt 6.2 für die hiesigen Messungen in allen Szenarien wie folgt reduziert.

$$\text{Max. lokaler Taskübernahmen} \quad e_{max} = \frac{\lceil m \cdot 0.7 \rceil}{\omega - \lceil \frac{\omega}{2} \rceil}$$

Des Weiteren wurde die lokale Anzahl der zeitgleich optimierten Tasks pro Kern mit $o_{max} = 1$ begrenzt, weshalb ein Kern in jedem Intervall nur eine Task aus seiner Menge anbietet. Die größtmögliche Taskanzahl i_{max}, die während eines Intervalls von den schlechteren Kernen zeitgleich angeboten und optimiert wird, entspricht somit zugleich der herbeigeführten Verschlechterung des Systems.

$$\text{Max. Optimierungen pro Intervall} \quad i_{max} = o_{max} \cdot \left\lceil \frac{\omega}{2} \right\rceil$$

Im Gegensatz zur Selbst-Konfiguration und -Heilung werden bei den hiesigen Messreihen zur Optimierung auch die optionalen Zustandsdaten von Tasks berücksichtigt. Dadurch können innerhalb einzelner Optimierungssequenzen die größtmöglichen Datenmengen für Taskkomponenten transferiert werden. Dennoch sind die auftretenden V-Zykluszeiten ähnlich wie bei den vorangegangen Messreihen in Abschnitt 6.3 und 6.4, da die Anzahl der lokal ausgeführten Tasks niedriger ist.

Dadurch verkürzt sich im Gegenzug die Kommunikationszeit von Datenhormonen, weshalb sich die Transferzeit trotz der höheren Datenmengen kaum verschlechtert.

Zum besseren Verständnis werden, wie auch bei der Selbst-Heilung nur die Ergebnisse der Messreihen für ein Set mit 64 Tasks gezeigt. Die Kerndaten der Messreihen sind für alle Szenarien in der folgenden Tabelle aufgeführt. Für die anderen Sets ergeben sich ähnliche Werte.

Grid	Optimierungen (Ts - i_{max})	Intervalle (n)
2x2	16 - 2	8
3x3	18 - 5	5
4x4	16 - 8	3
4x5	15 - 10	3
5x5	13 - 13	2
6x5	15 - 15	2
6x6	14 - 18	1

Abbildung 6.9 zeigt die Selbst-Optimierung aus Sichtweise der Gruppe von Tasks, die auf den besser geeigneten Kernen ausgeführt werden. Die Optimierungsintervalle $I_1 - I_8$ wurden jeweils durch blau gefärbte Flächen und Linien eingezeichnet. Für die Intervallänge wurde eine uniforme Zeitspanne mit $t_{opt} = 2000\,\text{ms}$ gewählt. Dies entspricht annähernd der in Abschnitt 4.3.0.5 veranschlagten minimalen Intervallänge unter Berücksichtigung der Kompensationsphase von Tasks bei einer unterbrechungsbehafteten Optimierung[2] mit $t_{opt} \geq t_{VC} + t_{\dot{Z}\dot{A}} \geq 2071\,\text{ms}$. Wie zu erkennen ist, nimmt die Anzahl an Taskausführung auf den besser geeigneten Kernen mit jedem Intervall zu. Dies impliziert im Gegenzug, das die Taskverarbeitung auf den schlechteren Kernen abnimmt, da während des Testbetriebs keine Mehrfachübernahmen von Tasks erlaubt sind. Je nach Szenario ist die Menge der benötigten Intervalle unterschiedlich. Während bei dem 6x6 Szenario die Optimierung bereits nach einem Intervall beendet ist, dauert es bei dem 2x2 Szenario insgesamt 8 Intervalle bis das System erneut den balancierten Zustand erreicht. Dies lässt sich auf die Menge der zeitgleich optimierten Tasks i_{max} zurückführen, die bei dem 2x2 Szenario mit $i_{max} = 2$ Optimierungen pro Intervall deutlich niedriger ist, als bei dem 6x6 Szenario mit $i_{max} = 8$ (siehe Tabelle oben). Nach Ablauf von Intervall I_8 erfolgt bei allen aufgeführten Messreihen keine weitere Optimierung von Tasks, da die ungleichen Hormonpegel im Zuge der Intervalle $I_1 - I_8$ kompensiert wurden und die für den Testbetrieb gewählten Hormonwerte die KHS-Stabilitätskriterien aus Forschungsarbeit [vR12] erfüllen. Die Taskzuordnung bleibt deshalb stabil und es treten keine Oszillationseffekte auf.

[2]Die Werte $t_{VC} = 1035.6\,\text{ms}$ und $t_{\dot{Z}\dot{A}} = 1035.55\,\text{ms}$ entsprechen der größtmöglichen Worst Case Zeit gemäß den Berechnungsvorschriften aus Kapitel 4.3 für das 2x2 Szenario bei einem Set von 64 Tasks und einem Kompensationsverhältnis mit $c_\sigma = 1$.

Abbildung 6.10 zeigt die Zeitdauer der Selbst-Optimierung für alle Intervalle. Zum Vergleich wurden die zugehörigen Worst Case Zeitschranken aus Kapitel 4.3.0.5 für jedes Szenario durch einen roten Punkt auf der Datenlinie eingezeichnet. Wie erwartet wurden für jedes Intervall die Zeitschranken eingehalten. Die Zusammenfassung aller Messwerte ist in Tabelle 6.4 aufgeführt.

Die beiden Schaubilder in Abbildung 6.11 zeigen die Auswirkungen der Selbst-Optimierung auf das Verhalten der Taskausführung für das Intervall I_1. Diese werden durch die am längsten andauernde Optimierungssequenz einer Task hervorgerufen, die innerhalb des Intervalls unterbrechungsbehaftet optimiert wird. Das Schaubild 6.11a zeigt für jedes Szenario den zeitlichen Fortschritt der optimierten Taskausführung im Vergleich zur nicht optimierten Variante, gemessen anhand der verarbeiteten Instruktionsmenge. Die Ausfallzeit, die bei der optimierten Ausführung (durchgezogene Linie) aufgrund der Unterbrechung entsteht, ist im Schaubild durch eine Stagnation der verarbeiteten Instruktionsmenge zu sehen. Hingegen nimmt der Taskfortschritt bei nicht optimierter Ausführung (gestrichelte Linie) kontinuierlich zu. Nachdem die Optimierungssequenz beendet ist, erreicht die optimierte Ausführung rechtzeitig vor Ablauf des Optimierungsintervalls bei allen Messreihen die Kompensation, was durch den Schnittpunkt der beiden Linien (roter Punkt) dargestellt ist. Dies ist zugleich der Zeitpunkt ab dem sich die optimierte Taskausführung aus Sichtweise der Taskausführung zunehmend gegenüber der nicht optimierten amortisiert. Das Schaubild 6.11b zeigt den zugehörigen Verlauf der Ausfallmenge, die während der Ausfallzeit seitens der optimierten Taskausführung entsteht. Diese nimmt nach Beginn der Optimierung zunächst stark zu, erreicht nach Ablauf der Sequenz ihren Höchststand (im Scheitelpunkt) und wird dann im Zuge der Kompensation vollständig abgebaut. Für die restlichen Intervalle $I_2 - I_8$ ergibt sich ein ähnliches Verhalten und die Kompensation wird jeweils vor Ablauf des jeweiligen Intervalls erreicht. Für die aufgeführten Messreihen kann daher festgehalten werden, dass die gewählte Zeitspanne $t_{opt} = 2000\,\text{ms}$ mit einem Kompensationsverhältnis von $c_\sigma = 1$ als minimale Intervallänge ausreichend ist.

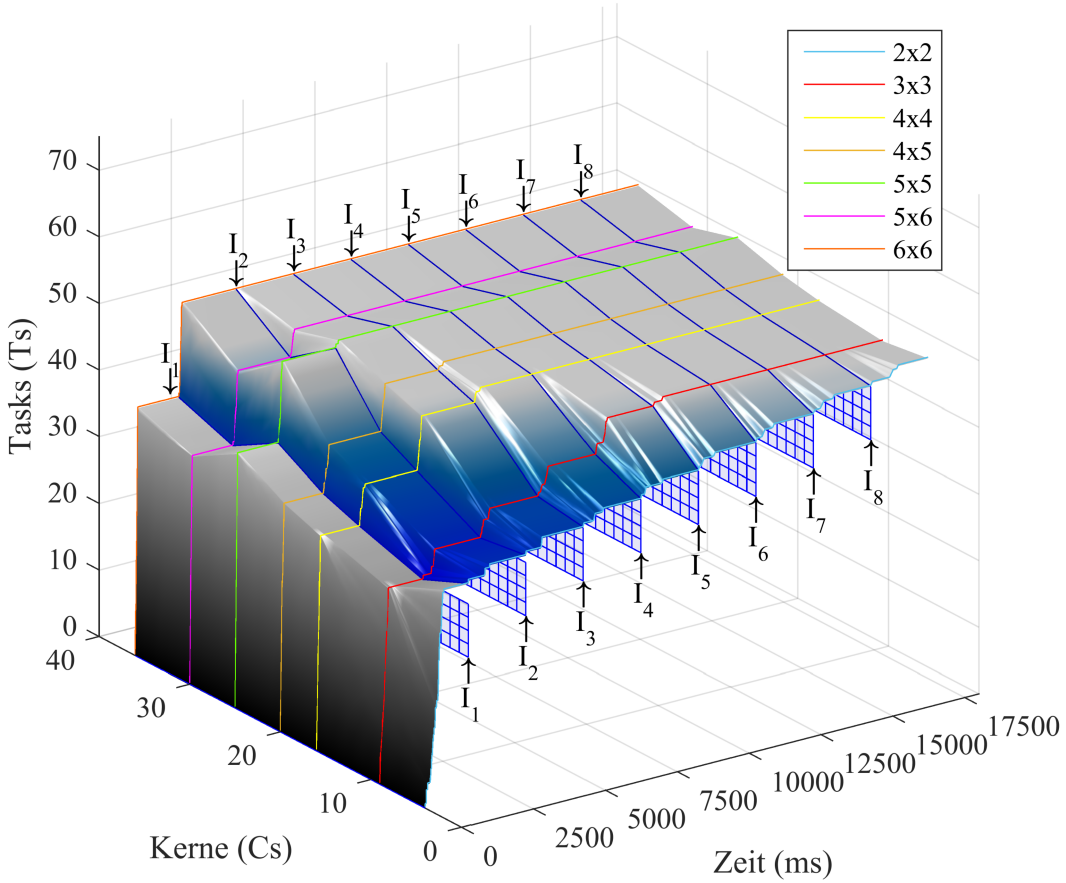


Abbildung 6.9.: Messreihen für die Selbst-Optimierung

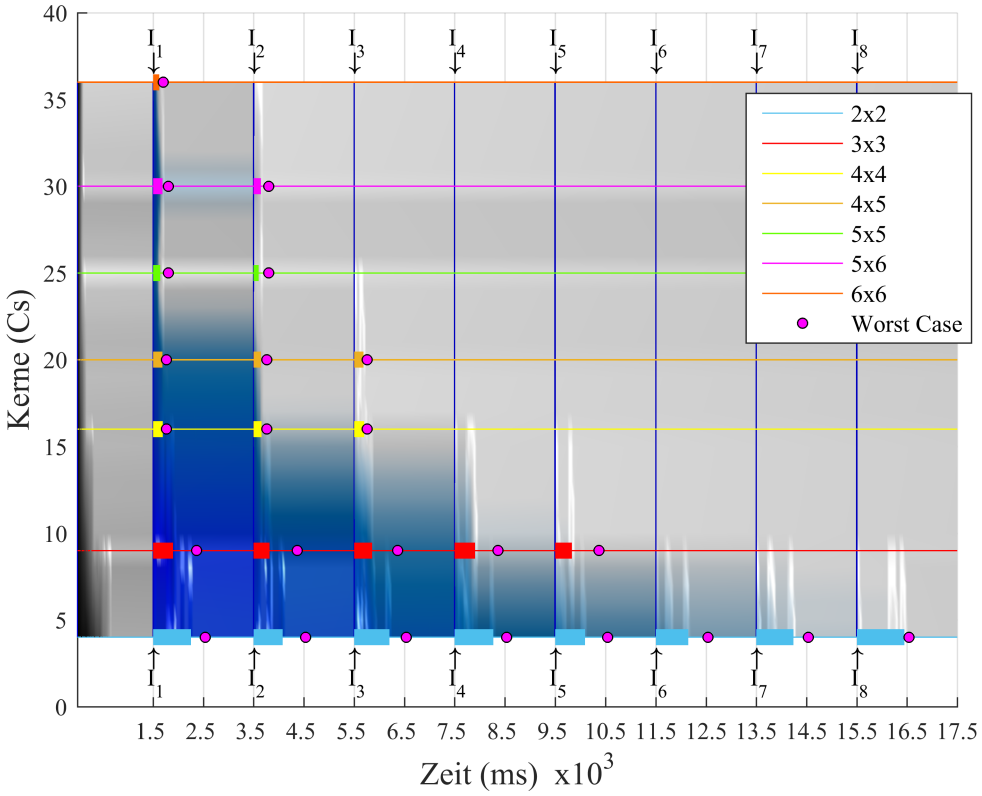
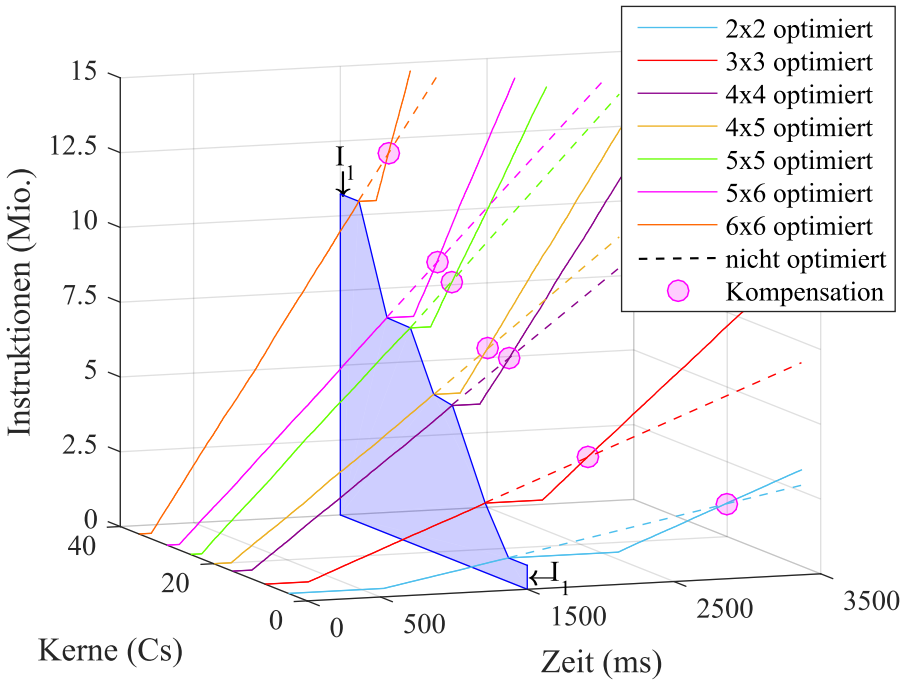


Abbildung 6.10.: Zeitdauer für die Selbst-Optimierung

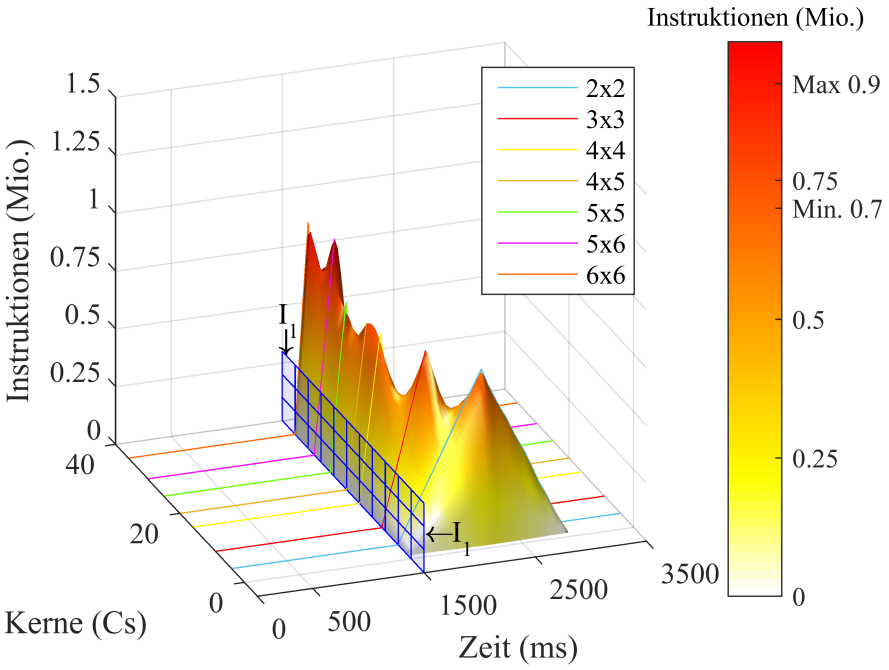
Kerne \ Intervalle	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
2x2	749,56	574,63	702,33	769,06	589,08	645,20	738,13	950,04
3x3	393,88	318,80	361,13	407,05	326,14	-	-	-
4x4	188,71	157,79	196,25	-	-	-	-	-
4x5	186,99	141,36	185,05	-	-	-	-	-
5x5	139,21	101,63	-	-	-	-	-	-
6x5	183,27	145,07	-	-	-	-	-	-
6x6	122,00	-	-	-	-	-	-	-

Zeit (ms)

Tabelle 6.4.: Messwerte für die Zeitdauer der Selbst-Optimierung in Abbildung 6.10



(a) Vergleich optimierter vs. nicht optimierter Ausführung



(b) Verlauf der Ausfallmenge bei optimierter Ausführung

Abbildung 6.11.: Vergleich und Analyse der optimierten Taskausführung im Intervall I_1

7. Abgrenzung und Vergleich zu anderen Systemen

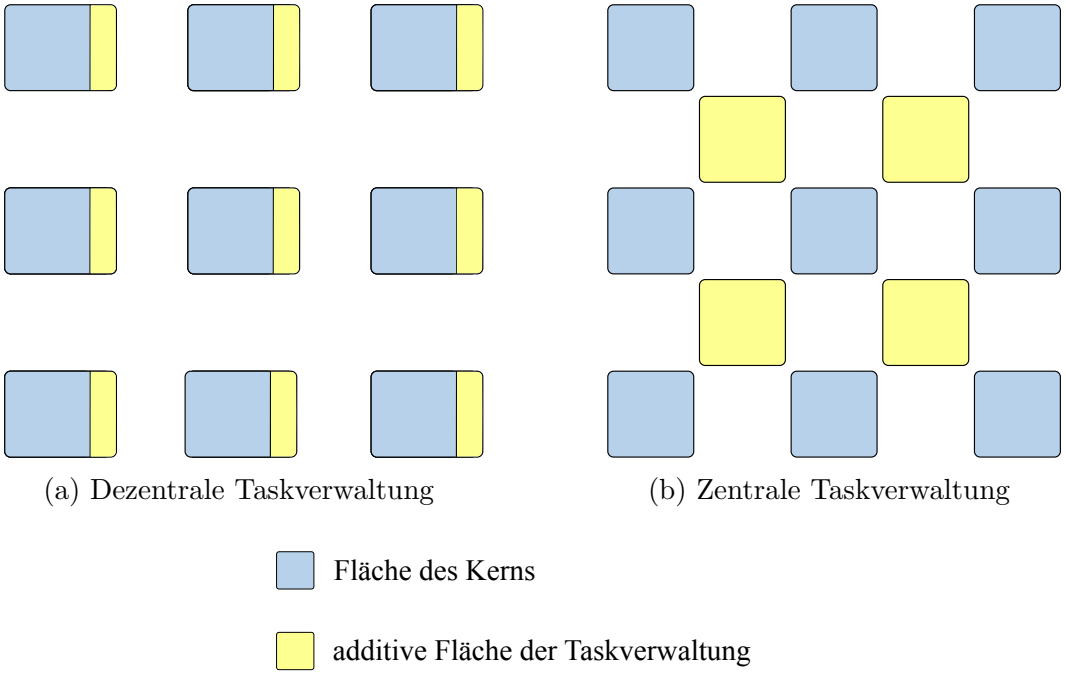
In diesem Kapitel erfolgt die Betrachtung weiterer Ansätze für die Umsetzung zuverlässiger Taskverarbeitungssysteme auf Multi-Core Architekturen. Dabei sollen die quantitativen Eigenschaften dieser Ansätze formal analysiert werden. Der Fokus liegt auf der Umsetzung und Funktionsweise des Taskverwaltungsmechanismus, der mittels einer entsprechenden Methode die zuverlässige Zuordnung von Tasks im System übernimmt und sicherstellt. Durch die Betrachtung anderer Task verteilender Methoden sollen die Unterschiede zwischen den Systemen erörtert werden, so dass letztlich eine Gegenüberstellung mit dem in dieser Arbeit verwendeten Konzept einer organischen Taskverwaltung (KHS) möglich

Nachfolgend werden in Abschnitt 7.1 die grundlegenden Modelle einer zuverlässigen Taskverwaltung auf Multi-Core Ebene vorgestellt. Anschließend werden in Abschnitt 7.2 die Taskverwaltungsmethoden anderer Systeme erläutert. Zusammenfassend erfolgt dann in Abschnitt 7.3 ein Vergleich mit dem KHS.

7.1. Zuverlässiges Taskverwaltungsmodell

Das Funktionsprinzip eines zuverlässigen Taskverwaltungsmodell auf Multi-Core Ebene kann aus Sichtweise der Taskzuordnung entweder zentral oder dezentral erfolgen.
Bei einem dezentralen Prinzip erfolgt die globale Taskzuordnung aller Tasks durch das Mitwirken der Task verarbeitenden Kerne im Netz. Die Implementierung des Verteilungsmechanismus erfolgt daher als integrativer Bestandteil auf jedem Kern, wodurch eine symmetrische Anordnung gleichwertiger und lokaler Instanzen des Verwaltungsmechanismus im Netz entsteht. Im Betriebszustand kommunizieren die Instanzen miteinander und tauschen ihre lokalen Taskinformationen untereinander aus. Auf Basis dieser Information entscheidet jede Instanz selbst, welche Task der verwaltete Kern lokal übernimmt. Durch den Austausch der lokalen Taskinformation zwischen den Instanzen aller Kerne, erfolgt ein globale Optimierung bezüglich der gesamten Taskverteilung. Unter Berücksichtigung der Annahme, dass jeder Kern die Ausführung einer jeden Task ermöglicht, ist die Funktionsweise der Taskverwaltung sichergestellt, solange mindestens ein Kern im Netz verfügbar ist, dessen Verwaltungsinstanz funktionsfähig ist. Abbildung 7.1a zeigt die Implementierung ei-

ner vollständig dezentralen Taskverwaltung für ein 3x3 Multi-Core Grid.
Bei einer zentralen Funktionsweise hingegen erfolgt die Taskzuordnung durch den Einsatz zusätzlicher Steuer-/Kontrolleinheiten. Die Implementierung des Verteilungsmechanismus erfolgt dann in redundanter Form auf einer zusätzlichen Menge von sogenannten Controllern. Dadurch entsteht im Netz eine asymmetrische Anordnung von Task verteilenden Controllern sowie Task verarbeitenden Kernen. Die globale Taskzuordnung wird daher ausschließlich von den Controllern übernommen und auch in diesen gespeichert. Das Funktionsprinzip ist sichergestellt solange mindestens ein Controller und ein Kern im Netz verfügbar ist. Abbildung 7.1b zeigt die Implementierung einer zentralen Taskverwaltung mittels redundanter Controller für ein 3x3 Multi-Core Grid.



(a) Dezentrale Taskverwaltung (b) Zentrale Taskverwaltung

Abbildung 7.1.: Vergleich der Taskverwaltungsmodelle

Das in dieser Arbeit vorgestellte Prinzip einer organisch-inspirierten Taskverwaltung mittels des KHS entspricht vollständig dem dezentralen Modell. Der Aufbau ist vollends symmetrisch, da das KHS gleichermaßen auf allen Kernen software- oder hardwareseitig ausgeführt wird. Dadurch erfolgt die globale Optimierung der Taskzuordnung wobei nur die lokale Zuordnung auf jedem Kern gespeichert wird.

Auf Basis der beiden oben aufgeführten Modelle, erfolgt nun die Betrachtung der Taskverwaltung in anderen Systemen. Je nach System wird jeweils eines der beiden Modelle genutzt. Des Weiteren gibt es Systeme deren Taskverwaltung auf Mischformen der beiden Modelle zurückgreift, so dass teilweise ein dezentrales/zentrales Funktionsprinzip entsteht.

7.2. Aktuelle Methoden zur Taskverwaltung

Es folgt die Betrachtung einer Auswahl zuverlässiger Taskverwaltungsmethoden anderer Multi-Core Systeme. Neben dem eigentlichen Funktionsmodell werden auch die Unterschiede zum KHS diskutiert. Des Weiteren werden alternative dezentrale Lösungen zum KHS vorgestellt.

7.2.1. Fehlertolerantes Multi Controller System (MCS)

Dieser Ansatz stammt aus dem Bereich der fehlertoleranten Regelungstechnik und wird aufgrund seiner einfachen Implementierung am häufigsten in heutigen Multi-Core Systemen eingesetzt. Das Prinzip entspricht dem Modell einer zentralen Verwaltung, wobei der eigentliche Mechanismus zur Taskverteilung redundant in mehreren Controllern implementiert und ausgeführt wird. Die Selbst-Konfiguration erfolgt mittels des folgenden Prinzips: Die Controller wählen aus ihrer Menge einen Stellvertreter (Manager) aus, der die aktive Rolle der Taskverteilung im Netz übernimmt. Dazu überprüfen die Controller gegenseitig die Funktionsbereitschaft des Anderen mittels eines zyklisch gesendeten Controller-Status. Dieser meldet den aktiven oder inaktiven Betriebszustand eines Controllers an alle Anderen. Das Auswahlverfahren verzichtet daher auf den Einsatz komplexer Maklerprotokolle, so dass direkt über den entsprechenden Status ein funktionsfähiger Manager bestimmt wird. Dieser empfängt die von den Kernen gesendeten Zustandsdaten, die anders als die von den Controllern gesendeten Statussignale umfassendere Informationen über die Taskeignung und den lokalen Zustand (Lastwert, Temperatur etc.) eines Kerns beinhalten. Nach Auswertung der Kerndaten sowie der geltenden Taskverwandtschaften führt der Manager die Zuordnung aller Tasks in bestimmte Task-Cluster durch. Abschließend überträgt der Manager die von ihm ermittelte Taskzuordnung an die anderen Controller im Netz. Die Selbst-Optimierung fungiert ausschließlich aus Sichtweise der Taskzuordnung. Dazu überprüft der Manager im eingeschwungenen Zustand die von den Kernen und Tasks gesendeten Zustandsdaten. Bei einer Verschlechterung der Daten führt der Manager eine Umverteilung der Tasks durch und synchronisiert diese Änderungen mit den anderen Controllern. Die Selbst-Heilung hingegen fungiert in zweierlei Hinsicht. Fällt der gegenwärtige Manager aus, so wird im nächsten Zyklus ein Nachfolger gewählt, der die Taskverwaltung auf Basis der redundant gesicherten Taskzuordnung fortsetzt. Durch das Ausbleiben der Zustandsdaten von Kernen und Tasks, detektiert der Manager den Ausfall entsprechender Komponenten und führt eine Neuzuordnung der betroffenen Taskmenge auf die verbleibenden Kerne durch. Eine proaktive Heilung von Kernen und Tasks kann im Zuge der Selbst-Optimierung erfolgen.

In Literatur [WZS+14] wird ein MCS zum Einsatz auf einem Multi-Core Cluster vorgestellt. Im Gegensatz zur reinen Single-Chip-Architektur besteht das Systemmodell aus mehreren Multi-Core Rechenknoten, die über ein Netzwerk miteinander

verbunden sind. Das System kann daher als hierarchisches Modell aufgefasst werden, welches die Taskverteilung auf zwei Ebenen durch zwei Arten von Task-Controllern implementiert. Auf erster Ebene erfolgt die Partitionierung der gesamten Taskmenge auf die einzelnen Knoten durch einen globalen Controller (GS). Dieser wird redundant in mehreren Rechenknoten implementiert und durch Auswahlverfahren ernannt. Anschließend erfolgt auf zweiter Ebene die lokale Verteilung der partitionierten Taskmenge in den Rechenknoten mittels lokaler Controller. Diese werden in allen Knoten redundant implementiert und es wird ebenfalls ein lokaler Stellvertreter ernannt. Das Verteilungsverfahren das auf beiden Ebenen verwendet wird, ist ein dynamischer Load-Balancing Ansatz (Work-stealing, Work-sharing). Dabei werden Tasks zwischen über- und unterbelasteten Kernen oder Knoten wechselseitig verschoben um insgesamt eine homogene Lastverteilung auf allen Rechenknoten zu erzielen.

In Literatur [SJPS09] und [CM08] werden alternative Algorithmen für ein MCS nach dem Nearest-Neighbour (NN) und dem First-Free (FF) Prinzip vorgestellt. Die Zuordnung von Tasks erfolgt anhand der Netz-Distanzen auf die nächstgelegenen oder freien Nachbarkerne um letztlich den Kommunikationsaufwand zu minimieren.

7.2.2. Auktionsbasiertes Multi-Agenten-System (MAS)

Bei einem auktionsbasierten Multi-Agenten-System handelt es sich um einen dynamischen Ansatz, der eine Taskverteilung durch das kollektive Zusammenspiel zwischen mehreren Agenten im System umsetzt. Dabei wird auf Multi-Core Ebene zwischen zwei Agenten-Rollen unterschieden. Der Makler (Broker) ist ein Agent, der für die Verteilung von Tasks im System verantwortlich ist und mittels Ausschreibeverfahren durchführt. Der Klient ist ein Agent, der an dem Ausschreibeverfahren des Maklers teilnimmt, um die Zuordnung von Tasks zwischen sich, dem Makler und den restlichen Klienten zu bestimmen. Der Makler fungiert als Initiator, Teilnehmer, Entscheidungs-Träger und -Empfänger des Ausschreibeverfahrens. Die Klienten als Teilnehmer und Entscheidungsempfänger. Der Makler stellt daher einen Klienten höherer Ordnung dar. Die Implementierung von Agenten erfolgt durch Hardware- oder Softwareinstanzen, die lokal auf allen Kernen ausgeführt werden. Da sich die funktionale Komplexität des Maklers deutlich von der des Klienten abhebt, wird die Makler fähige Instanz nur auf einer begrenzten Auswahl an Kernen integriert. Die Anzahl potentiell verfügbarer Makler ist somit im Netz limitiert. Dadurch entsteht eine Mischform der beiden Taskverwaltungsmodelle aus Abschnitt 7.1, die eine teilweise dezentrale Taskverwaltung durch das Zusammenspiel zwischen lokalen Klienten und einem globalen Makler implementiert. Die Sequenz der Selbst-Konfiguration ist wie folgt: Gemäß eines vorgegebenen Handlungs-Protokolls (bspw. ContractNet [Smi80]) eröffnet der Makler für jede zu verteilende Task eine Auktion. Innerhalb dieser übermitteln die Klienten Ihre Task-Gebote an den Makler. Das Gebot eines Klienten errechnet sich aus der Task-Eignung und dem lokalen Zustand (Lastwert,

Temperatur etc.) des verwalteten Kerns. Anschließend fällt der Makler auf Basis der Gebote sowie der gelten Taskverwandtschaften für jede Task die Zuordnung (Task-Zuschlag). Dadurch erfolgt die Speicherung der Taskzuordnung gleichermaßen in den Klienten und dem Makler. Zu berücksichtigen ist, dass die Selbst-Heilung nicht zu den Kern-Eigenschaften eines MAS zählen. Um eine Selbst-Heilung zu realisieren, müssen die Klienten wie auch der Makler zyklische ihren Status austauschen und wechselseitig überprüfen. Dies kann durch das Senden zusätzlicher Zustandsdaten erfolgen. Dadurch lassen sich Kern- und Task-Ausfälle seitens beider Agenten detektieren und durch eine erneute Ausschreibung der betroffenen Taskmenge heilen. Auch der Ausfall des Maklers kann so detektiert und durch die Wahl eines neuen Maklers (Auktion) behoben werden. Hingegen erfolgt die Selbst-Optimierung auch ohne den Austausch weiterer Zustandsdaten.

In Literatur [ERHH11] wird ein MAS zur Balancierung der thermischen Verlustleistung in 3D Multi-Core Architekturen vorgestellt. Mehrere Agenten unterschiedlichen Typs kommunizieren auf unterschiedlichen Ebenen miteinander, um letztlich eine Taskzuordnung unter Berücksichtigung der aktuell herrschenden Temperaturverhältnisse durchzuführen. Dadurch wird das Auftreten lokaler Hot-Spots auch bei maximaler Auslastungen des Systems vermieden. Aus Sichtweise von Agenten, handelt es sich um einen hierarchisches Konzept, dass durch ein übergeordnetes Auswahlverfahren die Selektion verschiedener Makler-Agenten implementiert, die im Netz redundant verfügbar sind. Ähnlich dem KHS werden Single-Point-of-Failure Quellen vollständig vermieden. Allerdings beschränkt sich das System auf eine Temperatur geregelte Zuordnung von Tasks.

7.2.3. Alternative, Dezentrale Lösungen

Neben dem KHS gibt es weitere Methoden die eine vollständig dezentrale Taskverwaltung implementieren. Im Folgenden wird der Einsatz von dynamischem Load-Balancing und einer weiteren Middleware SoC-Architektur als alternative, dezentrale Lösung diskutiert.

7.2.3.1. Dynamisches Load-Balancing (DLB)

Load Balancing ist eines der ältesten Basis-Verfahren das zur Verteilung von Tasks in Multi-Core Systemen eingesetzt wird. Die Zuordnung von Tasks erfolgt unter Berücksichtigung der aktuellen Rechenauslastung auf allen Kernen, so dass letztlich eine lastgerechte Gesamtverteilung im Netz erzielt wird. Der Mechanismus ist einfach und verwendet bei jeder Taskzuordnung die lokalen Lastwerte aller Kerne als Entscheidungskriterium. Die Umsetzung des eigentlichen Mechanismus kann entweder als zentrales oder vollständig dezentrales Modell aus Abschnitt 7.1 erfolgen. Des Weiteren existieren dynamische Balancing-Methoden die neben dem Lastverhalten weitere Kriterien zur Verteilung von Tasks verwenden (Kommunikationsaufwand,

Energieverbrauch etc.). Im Folgenden werden nun zwei unterschiedliche Balancing-Methoden mit dezentralem Modell vorgestellt.

In [SDD06] wird eine Lastverteilung vorgestellt, die in Kombination mit der agenten-basierten Tankverwaltungsstruktur in [Cao04] auf Mehrkernprozessoren eingesetzt werden kann. Die Umsetzung der Taskverwaltung erfolgt durch den Einsatz softwarebasierter Agenten, die auf jedem Kern erzeugt werden. Das Prinzip ist dabei wie folgt: Benachbarte Agenten treten miteinander in Verbindung und tauschen Informationen über das Lastverhalten ihrer Kerne aus. Anschließend entscheidet der Agent, dessen Kern den kleinsten Lastwert aufweist, auf Übernahme der betroffenen Task und teilt dies den umliegenden Agenten mit. Anders als bei einem auktionsbasierten Multi-Agenten Ansatz aus Abschnitt 7.2.2, verhandeln die Agenten nicht mittels eines Maklers über die Zuordnung von Tasks, sondern entscheiden direkt selbst auf Basis der vorliegenden Lastinformation. Auch nutzt ein Agent lediglich die Lastinformation seiner Nachbarn um den Lastwert seines Kerns auszugleichen. Selbst-Optimierung von Tasks ist garantiert, solange mindestens ein Agent in der Nachbarschaft erreichbar ist. Damit ein Agent seinen Aktionsradius nach eigenem Ermessen erweitern kann, erzeugt dieser optional Helfer-Tasks die einen nächst entfernten Agenten außerhalb der Nachbarschaft aufspüren können. Dadurch kann eine bessere globale Lastverteilung erzielt werden. Die Entstehung lokaler Optima wird aber auch damit nicht vollständig vermieden. Des Weiteren werden im Gegensatz zum KHS keine weiteren Kriterien für die Taskverwaltung berücksichtigt. Auch das Verhalten der Selbst-Heilung fungiert ausschließlich reaktiv. Für eine proaktive Heilung ist ein zusätzlicher Austausch vitaler Systemparameter zwischen den Agenten erforderlich, der aktuell nicht implementiert wird.

Der Partikel Ansatz in [HS95] folgt dem Beispiel eines natürlich inspirierten Ansatzes um einen dynamischen Lastausgleich in einem Multi-Core System zu erzielen. Dabei wird die Taskverteilung durch ein wechselseitiges Zusammenspiel verschiedener physikalischer Kräfte umgesetzt die zwischen den Tasks und Kernen herrschen. Hierbei werden die Tasks abstrakt als Partikel und die Kerne als lokale Massezentren aufgefasst. Zwischen diesen beiden Komponenten werden dann die herrschenden Gravitations-, Binde- und Reibungskräfte umgesetzt, die jeweils lokal von den Massezentren verrechnet und ausgewertet werden. Die Gravitationskraft entspricht dem inversen Lastwert der von einem bestimmten Massezentrum (Kern) auf ein entsprechendes Partikel (Task) wirkt. Ein Kern der ein höheren Lastwert als seine Nachbarn aufweist, hat im Gegensatz zu diesen auch folglich eine geringere Gravitation auf die Task. Die Bindekraft die zwischen den Partikeln herrscht, entspricht dem Kommunikationsverhalten verwandter Tasks. Die Reibungskraft repräsentiert den Transferaufwand und reguliert die Anzahl der Migrationen einer Task. Dieses Balancing-Verfahren ist in seinen Eigenschaften dem KHS sehr ähnlich, da nicht nur Lastverhalten sondern auch weitere Aspekte bei der Zuordnung mit einfließen und die Implementierung ausschließlich dezentral erfolgt. Allerdings läuft dieser Ansatz ebenfalls Gefahr, bei der Zuordnung in ein lokales Optimum zu verfallen, da

der Wirkungsbereich der Gravitation auf Nachbarzentren (Kerne) begrenzt ist. Des Weiteren ist Selbst-Optimierung und -Heilung von Tasks prinzipiell möglich, aber nicht expliziert definiert.

7.2.3.2. CAR-SoC

Das CAR-SoC Forschungsprojekt in [KMUU06] umfasst die Umsetzung eines weiteren Middleware gestützten SoC-Ansatz mit Selbst-X Eigenschaften. Im Gegensatz zu der in dieser Arbeit vorgestellten Systemarchitektur, erfolgt die Umsetzung der Taskverwaltung mittels zweier Regelkreise die einerseits lokal auf jedem Kern und anderseits global auf Middleware-Ebene ausgeführt werden. Auf Hardwareebene erfolgt die Integration von Kernen durch eine mehrfädige Mikroprozessorarchitektur (CareCore) die einen echtzeitfähigen Hardware-Scheduler bereitstellt. Auf Systemebene eines Kerns erfolgt die Integration eines echtzeitfähigen Betriebssystems (CarOS) in Kombination mit einer Kern übergreifenden Middleware (CARISMA,[NB08]). Der erste Regelkreis der Taskverwaltung wird zentral von jedem Kern ausgeführt und dient zur Überwachung dessen Zustands. Dazu wird nebenläufig ein softwarebasierter *Autonomic Manager* ausgeführt, der die etwaigen Systemparameter (Lastwert, Temperatur etc.) des Kerns überwacht. Der zweite Regelkreis übernimmt die Umsetzung der Taskverteilung auf Middleware-Ebene und unter Berücksichtigung der Zustandsdaten des *Autonomic Manager* im ersten Regelkreis. Hier kommt ein Multi-Agenten Ansatz mit Contract-Net-Protokoll zum Einsatz, wobei die Umsetzung von Maklern und Klienten im Netz statisch verteilt erfolgt. Im Gegensatz zum KHS erfolgt die Durchführung der Selbst-Eigenschaften (Konfiguration, Heilung und Optimierung) aufgrund des zentralen Autonomic Managers wie auch der unterschiedlichen Agentenrollen nur teilweise dezentral. Des Weiteren sind für die Handlungsprinzipien des zentralen *Autonomic Manager* im ersten Regelkreis keine Echtzeitanforderungen vorgesehen.

7.3. Quantitativer Vergleich gegenüber dem KHS

Es folgen quantitative Vergleichsanalysen bezüglich des Flächen- und Kommunikationsaufwands, sowie der Rechen- und Echtzeitfähigkeit zwischen dem KHS als vollständig dezentrales Modell, dem MCS als zentrales Modell und dem MAS als teilweise dezentrales Modell aus Abschnitt 7.2. Die Vorgehensweise in den nachfolgenden Abschnitten gliedert sich wie folgt:

- Abschnitt 7.3.1: Vergleichsanalyse des Flächenaufwandes vs. Ausfallsicherheit
- Abschnitt 7.3.2: Vergleichsanalyse des Kommunikationsaufwandes
- Abschnitt 7.3.3: Vergleichsanalyse des Rechenaufwands

- Abschnitt 7.3.4: Vergleichsanalyse von Echtzeitaspekten
- Abschnitt 7.3.4: Zusammenfassung der Ergebnisse

7.3.1. Vergleichsanalyse des Flächenaufwands vs. Ausfallsicherheit

Nachfolgend wird der Flächen-Mehraufwand, der durch die Implementierung des Taskverwaltungsmechanismus auf Chip-Ebene entsteht, im Verhältnis zur gesteigerten Ausfallsicherheit betrachtet. Hierbei wird eine vollständig, hardwarebasierte Implementierung der drei Methoden gemäß den in Abschnitt 7.1 vorgestellten Modellen berücksichtigt. Im Einzelnen folgt nun die Betrachtung der Ausfallsicherheit sowie des Flächenaufwands. Abschließend erfolgt die Gegenüberstellung beider Systemeigenschaften.

7.3.1.1. Definition und Annahmen zur Ausfallsicherheit

Für die nachfolgende Analyse soll gelten, dass sich die Gesamtausfallsicherheit des Systems ausschließlich durch die Anzahl der tolerierbaren Kernausfälle definiert. Die Ausfallsicherheit entspricht daher der Ausfalltoleranz von Kernen im System. Unter Berücksichtigung der Anzahl von w_f tolerierbaren Kernausfällen lässt sich die Ausfallsicherheit somit wie folgt definieren:

$$\text{Ausfallsicherheit des Systems} \quad R_f = \frac{\omega_f}{\omega} \tag{7.1}$$

Des Weiteren lassen sich bei einer Taskverteilung durch das KHS, MCS oder MAS die folgenden Aussagen gegenüber der Funktionsfähigkeit des Systems ableiten:

- KHS: Die Taskverteilung erfolgt in lokalen KHS-Instanzen und ist funktionsfähig solange mindestens ein Kern im Netz verfügbar ist.
  ⇒Der Systembetrieb ist gewährleistet, solange nicht mehr als ω_f tolerierbare Kerne im System ausfallen.

- MCS: Die Taskverteilung erfolgt durch einen stellvertretenden Controller und ist funktionsfähig solange mindestens einer der n_c Controller im Netz verfügbar ist.
  ⇒Der Systembetrieb ist sichergestellt, solange nicht mehr als ω_f tolerierbare Kerne und nicht mehr als $n_c - 1$ Controller im System ausfallen.

- MAS: Die Taskverteilung erfolgt mittels lokaler Agenten und ist funktionsfähig solange mindestens einer der n_m Makler fähigen Kerne im Netz verfügbar ist.
  ⇒Der Systembetrieb ist sichergestellt, solange nicht mehr als ω_f tolerierbare Kerne und nicht mehr als $n_m - 1$ Makler fähige Kerne im System ausfallen.

Damit bei einer Taskverteilung mittels MCS oder MAS die Ausfallsicherheit R_f aus Gleichung 7.1 nicht beeinträchtigt wird, muss die Anzahl der Controller n_c und der Makler fähigen Kerne n_m größer sein als die der tolerierbaren Kernausfälle w_f. Andernfalls könnte das System bereits ab einer geringeren Anzahl von w_f Kernausfällen nicht mehr funktionsfähig sein. Für die Häufigkeiten n_c und n_m gilt deshalb:

$$n_c - 1 \geq \omega_f \quad \Rightarrow \quad n_c \geq w_f + 1 \tag{7.2}$$

$$n_m - 1 \geq \omega_f \quad \Rightarrow \quad n_m \geq w_f + 1 \tag{7.3}$$

7.3.1.2. Definitionen und Annahmen zum Flächenaufwand

Der Flächen-Mehraufwand der durch die Implementierung der Methoden KHS, MCS oder MAS entsteht, definiert sich über den additiven Flächenanteil A der zusätzlichen, Task verteilenden Komponenten (KHS-Instanzen, Controller, Agenten) im Netz. Diesbezüglich können für die drei Verwaltungsmethoden KHS, MCS und MAS die folgenden Abschätzungen abgeleitet werden:

- KHS: Die Implementierung erfolgt symmetrisch mittels lokaler KHS-Instanzen auf allen Kernen. Dadurch wird die prozentuale Fläche eines Kerns um einen statischen Flächenanteil a_d vergrößert, so dass für den Flächenaufwand des KHS gilt:

$$A_{KHS} = a_d \tag{7.4}$$

- MCS: Die Implementierung erfolgt in n_c zusätzlichen Controllern. Unter der Annahme das der Flächenanteil eines Controllers gleich dem eines vollwertigen Kerns entspricht[1] und der Verwendung von Gleichung 7.2, gilt für den prozentualen Flächenaufwand des MCS das folgende Verhältnis:

$$\begin{aligned} A_{MCS} \geq \frac{n_c}{\omega} &\geq \frac{\omega_f}{\omega} + \frac{1}{\omega} \\ &= R_f + \frac{1}{\omega} \end{aligned} \tag{7.5}$$

- MAS: Die Implementierung erfolgt asymmetrisch mittels verschiedener Agenten auf allen Kernen. Dadurch vergrößert sich die Fläche eines Kerns um einen agentenspezifischen Anteil. Unter der Annahme das der Flächenanteil von Klienten im Verhältnis zu dem von Maklern vernachlässigbar klein ist und der Flächenanteil eines Maklers gleich dem eines vollwertigen Kerns entspricht[2],

[1] Diese Annahme erfolgt unter Berücksichtigung einer experimentellen VHDL-Implementierung eines MCS in [BBP13] die dies bestätigt. Für den Flächenaufwand des Controllers konnte die 1.3x Flächenmenge eines μSoC^3 Kern aus Abschnitt 6.1 berechnet werden.

[2] Diese Annahme erfolgt unter Berücksichtigung einer experimentellen VHDL-Implementierung eines MAS in [BBP13] die dies bestätigt. Für den Flächenaufwand des Maklers konnte die 1.5x Flächenmenge eines μSoC^3 Kern aus Abschnitt 6.1 berechnet werden.

gilt unter Verwendung von Gleichung 7.3 für den prozentualen Flächenaufwand des MCS das folgende Verhältnis:

$$A_{MAS} \geq \frac{n_m}{\omega} \geq \frac{\omega_f}{\omega} + \frac{1}{\omega} \\ = R_f + \frac{1}{\omega} \tag{7.6}$$

Da bei der obigen Flächenabschätzung des MAS der Flächenanteil lokaler Klienten-Instanzen vernachlässigt wurde kann angenommen werden, dass dessen tatsächlicher Flächenaufwand größer sein muss als in Gleichung 7.6 definiert. Unter Berücksichtigung dieser Annahme kann somit festgehalten werden, dass der Flächenaufwand eines MAS mindestens genauso groß ist wie der eines MCS und es gilt:

$$\Rightarrow \quad A_{MCS} \leq A_{MAS}$$

7.3.1.3. Gegenüberstellung und Zusammenfassung

Es folgt der Flächenvergleich im Verhältnis zur Ausfallsicherheit zwischen KHS, MCS und MAS. Dabei ist zu berücksichtigen das die Betrachtungsweise des Flächenaufwands sowie der Ausfallsicherheit stets in Prozent erfolgt. Damit sich das KHS betriebene System aus Sichtweise des Flächen-Mehraufwands gegenüber dem MCS und MAS amortisiert, muss gelten:

$$A_{KHS} \leq A_{MCS} \leq A_{MAS}$$

Durch Einsetzen des Flächenaufwands aus Gleichung 7.4, 7.5 und 7.6 erhalten wir die folgende Gesetzmäßigkeit:

$$\Rightarrow \quad a_d \leq R_f + \frac{1}{\omega} \\ \Rightarrow \quad R_f \geq a_l - \frac{1}{\omega} \tag{7.7}$$

Daraus lassen sich für den Flächenaufwand des KHS zwei Grenzwerte ableiten:

$$a_d := \begin{cases} < \frac{1}{\omega} & : \text{ Der Flächenaufwand des KHS ist stets kleiner als der von MCS und MAS} \\ > 1 + \frac{1}{\omega} & : \text{ Der Flächenaufwand des KHS ist stets größer als der von MCS und MAS} \end{cases} \tag{7.8}$$

Für jeden anderen Wert von a_d ist das Flächenverhältnis zwischen KHS, MCS und MAS abhängig von der Höhe der gewählten Ausfallsicherheit und der Anzahl ω an Kernen im Netz. Bei der Umsetzung des KHS betrieben Prototyps

aus Kapitel 6.1 wurde durch Synthese des VHDL-Modells AHS*LiHaDe* für den Flächenaufwand a_d des KHS ein Wert von weniger als 20% ermittelt[3]. Durch Einsetzen dieser oberen Schranke mit $a_d \leq 20\%$ in Gleichung 7.7 lässt sich ableiten, dass der Flächenaufwand des KHS ab einer Ausfallsicherheit von 20% stets kleiner ist als der von MCS und MAS. Abbildung 7.2 verdeutlicht diesen Zusammenhang und zeigt den Flächenaufwand für alle drei Methoden im Verhältnis zur Ausfallsicherheit. Gemäß Gleichung 7.4 ist der Flächenaufwand des KHS statisch. Bei MCS und

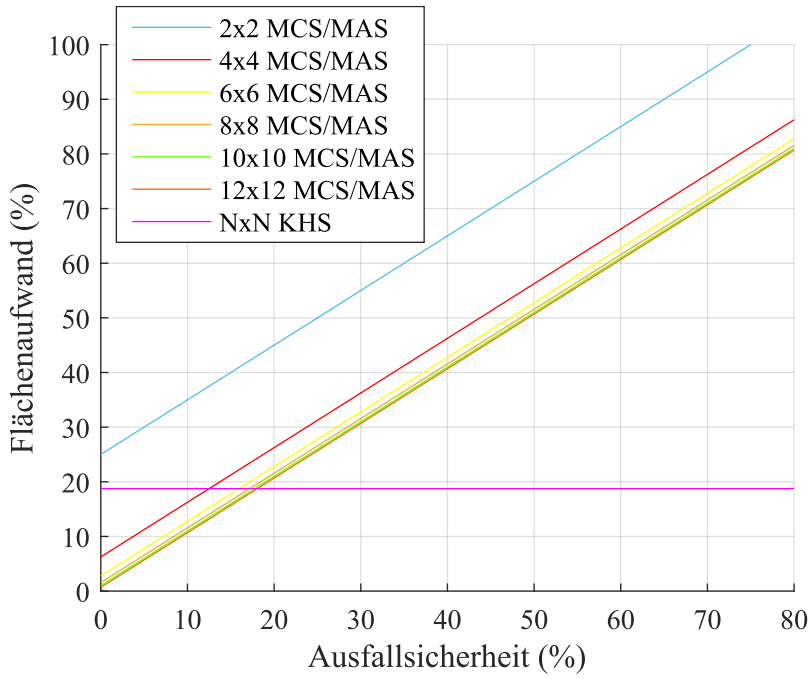


Abbildung 7.2.: Flächenaufwand von KHS, MCS und MAS im Verhältnis zur Ausfallsicherheit

MAS ist der Flächenaufwand proportional zur Höhe der Ausfallsicherheit und der Anzahl an Kernen im Netz. Wird eine Ausfallsicherheit von weniger als 20% gewählt, ist das Flächenverhältnis zwischen KHS, MCS und MAS stark unterschiedlich und wird primär von der Kernanzahl ω dominiert. Dies lässt sich durch die beispielhafte Betrachtung des 2x2 und 12x12 Szenario verdeutlichen. Da bei dem 2x2 Szenario ($\omega = 4$ Kerne) der untere Grenzfall aus Gleichung 7.8 mit $a_d = 0.19 < \frac{1}{\omega} = 0.25$ gilt, ist der Flächenaufwand des KHS unabhängig von der Höhe der Ausfallsicherheit stets kleiner als bei MCS und MAS. Hingegen tritt der Flächenvorteil des KHS bei dem 12x12 Szenario ($\omega = 144$ Kerne) erst ab einer Ausfallsicherheit von mehr als 19% ein. Bei immer größeren Werten von ω gilt jedoch, das sich der Flächenvorteil

[3]Für das VHDL-Modell des AHS*LiHaDe* gilt bei einer skalaren, superskalaren und mehrfädigen Architektur des μSoC[3] ein Flächenanteil a_d von 9.75%, 15% und 18.75%

des KHS stets ab einer Ausfallsicherheit von 20% gegenüber dem von MCS und MAS auszahlt.

Fazit: Das KHS erzielt bereits ab einem geringen Maß an Ausfallsicherheit einen deutlich niedrigeren Flächenverbrauch als die Methoden MCS und MAS. Dies gilt, weil die Wahl einer Ausfallsicherheit von mehr als 20% im Kontext heutiger Systemanforderungen seitens Fehlertoleranz (beschleunigte Alterung durch Strahlung, Integrationsdichte etc.) sehr realistisch erscheint.

7.3.2. Vergleichsanalyse des Kommunikationsaufwandes

Es folgt die Analyse des Kommunikationsaufwands von KHS, MCS und MAS im eingeschwungenen Endzustand. Dies bedeutet, dass die Verteilung von Tasks abgeschlossen ist und sich der Kommunikationsaufwand allein durch das Kommunikationsverhalten der Task verwaltungsbezogenen Komponenten (KHS-Instanzen, Controller, Makler, Tasks) innerhalb eines vorgegebenen Kommunikationszyklus definiert. Bezüglich dem KHS entspricht dies der gesendeten Hormondatenmenge pro H-Zyklus, bei MCS und MAS der gesendeten Zustandsdatenmenge innerhalb eines vergleichbaren Kontrollzyklus mit gleicher Periodendauer. Des Weiteren ist zu berücksichtigen, dass der optionale Mehraufwand der im Zuge einer Selbst-Optimierung und -Heilung von Tasks entsteht, bei allen drei Methoden vernachlässigt wird. Dies ist realistisch, da die Durchführung der beiden Vorgänge im Vergleich zur Anzahl der Kontrollzyklen eher selten erfolgt.

- KHS: Gemäß dem Hormonaufkommen aus Abschnitt 5.2.3 setzt sich die Kommunikationslast aus der gesendeten Hormonmenge zwischen Kernen und Tasks zusammen. Neben den insgesamt m Suppressoren, die aufgrund der Taskübernahmen global zwischen allen Kernen im Netz gesendet werden, sendet jeder Kern schlimmsten Falls $\varphi \cdot e_{max} \cdot v_{max}$ lokale Akzeleratoren an die benachbarten Kerne um die Erzeugung von Organ-Clustern zu begünstigen. Wird die Bildung von Organ-Clustern deaktiviert, kann die Datenmenge seitens der Akzeleratoren mit $v_{max} = 0$ vernachlässigt werden. Folglich kann der Kommunikationsaufwand pro H-Zyklus unter Berücksichtigung der Datenmenge D_{hv} von Steuerhormonen wie folgt definiert werden:

$$C_{KHS} = D_{hv} \cdot (m + \varphi \cdot e_{max} \cdot v_{max}) \tag{7.9}$$

- MCS: Der Kommunikationsaufwand definiert sich durch die Gesamtmenge an Zustandsdaten die zyklisch von Controllern und Tasks im Netz gesendet werden. Innerhalb eines Kommunikationszyklus sendet jede der m übernommenen Tasks ihren Zustand an den stellvertretenden Manager. Des Weiteren sendet jeder der n_c Controller seinen Zustand an die Anderen. Unter Berücksichtigung

der Zustandsdatenmenge D_c und D_t für Controller und Tasks, gilt für den zyklischen Kommunikationsaufwand eines MCS daher:

$$C_{MCS} = m \cdot D_t + n_c \cdot D_c \tag{7.10}$$

- MAS: Ähnlich zu MCS definiert sich der Kommunikationsaufwand durch die Zustandsdatenmenge die von den Agenten und Tasks im Netz gesendet wird. Da sich das Aktionsverhalten der Agenten ausschließlich auf die Phasen der Selbst-Konfiguration, -Heilung und -Optimierung beschränkt, wird deren Kommunikationsverhalten im eingeschwungenen Zustand ausschließlich durch das des Maklers bestimmt. Dieser sendet in jedem Kommunikationszyklus seinen Zustand an die restlichen $n_m - 1$ Makler fähigen Kerne im Netz. Des Weiteren sendet jede der m übernommenen Task zyklisch ihren Zustand an den Makler. Unter Berücksichtigung der Zustandsdatenmengen D_m und D_t von Maklern und Tasks gilt für den zyklischen Kommunikationsaufwand des MAS somit:

$$C_{MAS} = m \cdot D_t + D_m \tag{7.11}$$

Es folgt der Vergleich zwischen dem Kommunikationsaufwand aus Gleichung 7.9, 7.10 und 7.11. Dabei werden für die Datenmengen D_m, D_c, D_t und D_{hv} die folgenden Annahmen getroffen:

- D_m: Die Zustandsdatenmenge eines Maklers D_m beschränkt sich auf ein einfaches Statussignal, dass zur Überprüfung der Funktionsbereitschaft des Maklers dient und mit $D_m \leq 1$ Byte sehr gering ausfällt.
- D_c: Der Zustand eines Controllers setzt sich aus einem Zustandswert und einer Controller-ID mit insgesamt $D_c < 2$ Byte zusammen.
- D_t: Die Zustandsdatenmenge einer Task setzt sich aus dem eigentlichen Zustandswert sowie einer zusätzlichen Absenderkennung (Kern- u. Task-ID) mit insgesamt $D_t \geq 3$ Byte zusammen.
- D_{hv}: Die Datenmenge eines Steuerhormons gleicht dem des Taskzustands und umfasst neben dem eigentlichen Hormonwert sowie der Absenderkennung noch den Hormontyp mit insgesamt $D_{hv} \geq 3$ Byte.

Unter Berücksichtigung der obigen Annahmen kann das folgende Größenverhältnis zwischen den Datenmengen abgeleitet werden.

$$\begin{aligned} \Rightarrow \quad & D_m \leq D_c \\ \Rightarrow \quad & D_t \sim D_{hv} \end{aligned}$$

Unter Verwendung des obigen Größenverhältnis, lässt sich der Unterschied im Kommunikationsaufwand bei den drei Methoden am einfachsten mittels Subtraktion der

Gleichungen 7.9, 7.10 und 7.11 darstellen. Dabei wird die optionale Bildung von Organ-Clustern seitens des KHS mit $v_{max} = 0$ vernachlässigt:

$$\begin{aligned} C_{MCS} - C_{KHS} &= (m \cdot D_t + n_c \cdot D_c) - (D_{hc} \cdot (m + \varphi \cdot e_{max} \cdot v_{max})) \\ &= n_c \cdot D_c \end{aligned}$$

$$\begin{aligned} C_{MAS} - C_{KHS} &= (m \cdot D_t + D_m) - (m \cdot D_t + D_m) \\ &= D_m \end{aligned}$$

Wie sich anhand der Differenzen erkennen lässt, verursacht das KHS einen geringfügig kleineren Kommunikationsaufwand als das MCS und MAS sofern die Organ-Bildung im Netz vollständig deaktiviert ist ($v_{max} = 0$). Wird diese hingegen aktiviert, produziert das KHS einen höheren Aufwand als das MCS und MAS. Dies lässt sich darauf zurückführen, dass das Clustern von Tasks innerhalb des KHS mittels zusätzlicher Akzeleratoren erfolgt, was hingegen bei MAS und MCS zentral durch die Agenten und Controller selbst erfolgt und deren Zustandsdaten deutlich kleiner sind als die der Akzeleratoren ($D_m < n_c \cdot D_c < \varphi \cdot e_{max} \cdot v_{max} \cdot D_{hv}$).

Um ein besseres Verständnis über die erbrachten Ergebnisse zu bekommen, erfolgt die Betrachtung eines konkreten Beispiel-Szenarios aus Publikation [vRB10]. Dieses umfasst die Simulation eines autonom gesteuerten Fahrzeugs mit insgesamt 13 Tasks verteilt auf 4 Kernen. Hierbei konnten für die drei Methoden KHS, MCS und MAS die in der folgenden Tabelle aufgeführten Messwerte bezüglich Kommunikationsaufwand ermittelt werden:

	KHS	MCS	MAS	
Kommunikationsaufwand C	3.45	2.45	2.37	KBytes/sec
Taskkommunikation Dk	29.19	28.44	28.10	KBytes/sec
Anteil C/Dk	11.82	8.61	8.43	%

Um eine hohe Reaktionsfähigkeit gegenüber der autonomen Fahrzeugsteuerung zu gewährleisten, wurde bei allen drei Methoden eine Zykluszeit von 100 ms gewählt. Die gemessenen Werte entsprechen daher der Gesamtdatenmenge die in 10 Zyklen gesendet wurden. Wie sich erkennen lässt, ist der Kommunikationsaufwand des KHS im Vergleich zu dem von MCS und MAS um knapp 1/3 größer und berücksichtigt das Clustern von Tasks ($v_{max} = 4$). Im Vergleich zur Taskkommunikation ist der Kommunikationsaufwand aber bei allen drei Methoden ähnlich und fällt mit weniger als 12% sehr gering aus.

Fazit: Der Kommunikationsaufwand ist bei allen drei Methoden bis auf geringfügige Abweichungen ähnlich, wobei tendenziell das KHS aufgrund seiner vollständig dezentralen Arbeitsweise einen höheren Aufwand mit sich bringt, falls das Clustern von Tasks erfolgt.

7.3.3. Vergleichsanalyse des Rechenaufwands

Es folgt die Vergleichsanalyse des Rechen-Mehraufwands, der ausschließlich bei einer nebenläufigen Durchführung des Taskverwaltungsmechanismus auf Systemebene anfällt. Bei einer vollständigen Implementierung der Taskverwaltung in Hardware (wie in Abschnitt 7.3.1), entsteht hingegen kein zusätzlicher Rechenaufwand seitens der Taskverteilung. Deshalb wird für die hiesige Analyse eine softwarebasierte Implementierung der drei Methoden berücksichtigt, so dass im Gegenzug der Flächenaufwand aus Abschnitt 7.3.1 stark reduziert wird oder komplett entfällt. Der Gesamtrechenaufwand der bei einer Taskverwaltung mittels KHS, MCS und MAS auf Systemebene entsteht, setzt sich aus zwei Verarbeitungsschritten zusammen:

1. Schritt: Umfasst den Rechenaufwand der für die Verarbeitung der Kommunikationsdaten aus Abschnitt 7.3.2 anfällt. Hierzu zählt das Senden und Empfangen von Steuerhormonen oder Zustandsdaten der Task verwaltungsbezogenen Komponenten (Tasks, Kerne, Controller, Agenten). Da die Kommunikationsdatenmengen gemäß Abschnitt 7.3.2 bei allen drei Methoden annähernd gleich groß sind erfolgt die Annahme, dass der Rechenaufwand der für die Verarbeitung der Daten anfällt ebenfalls vergleichbar erscheint [4].

2. Schritt: Umfasst den Rechenaufwand der für die Berechnung der eigentlichen Taskverteilung benötigt wird. Hierzu zählt das Auswerten der empfangenen Hormone oder Zustandsdaten aus Schritt 1., die weitere Analyse von Taskverwandtschaften und letztlich das Entscheiden über die Zuordnung der Tasks in die vorliegenden Task-Cluster.

Da der Aufwand in Schritt 1. bei allen drei Methoden als vergleichbar angenommen wird, definiert sich der Unterschied im Rechenaufwand allein über die Berechnungen in Schritt 2. Diese werden im Einzelnen nun genauer betrachtet:

- KHS: Die Berechnung der Taskzuordnung erfolgt direkt in den lokalen KHS-Instanzen der Kerne. Dabei überprüft ein Kern für jeden seiner beworbenen Tasks seinen Eignungswert mit den jeweils höchstwertig Empfangenen. Unter Berücksichtigen das sich ein Kern maximal für m_{max} Tasks interessiert, müssen schlimmsten Falls m_{max} Eignungswerte miteinander verglichen werden. Für das Clustern von Tasks sind keine weiteren Berechnung erforderlich, da die herrschenden Taskverwandtschaften aufgrund der nebenläufigen Verrechnung der Organ-Akzeleratoren mit den Eignungswerten bei jedem Vergleich mit einfließen können. Der Berechnungsaufwand des KHS ist somit proportional zur

[4] Diese Annahme erfolgt unter Berücksichtigung der experimentellen Evaluationsergebnisse in Publikation [vRB10], die diese bestätigen. Für die Methoden KHS, MCS und MAS konnte auf Systemebene ein annähernd vergleichbarer Rechenaufwand für das Verarbeiten der Kommunikationsdatenmenge ermittelt werden.

maximalen Anzahl an Tasks m_{max} für die sich ein Kern interessiert.

$$P_{KHS} \sim m_{max}$$

- MAS: Die Berechnung der Taskzuordnung erfolgt allein durch den stellvertretenden Manager, der rekursiv die Taskeignung von allen Kernen überprüft. Hierbei müssen für jeden der ω Kerne jeweils m Tasks geprüft werden. Des Weiteren müssen für das Clustern von Tasks die auf den Nachbarkernen herrschenden Taskverwandtschaften berücksichtigt werden. Dazu werden auf jedem der φ_{max} benachbarten Kerne v_{max} Tasks auf Verwandtschaft geprüft. Somit gilt für den Berechnungsaufwand des MCS:

$$P_{MCS} \sim m \cdot \omega \cdot (1 + \varphi \cdot v_{max})$$

- MCS: Die Berechnung der Taskzuordnung erfolgt durch den Makler und ist vergleichbar zum Rechenvorgang des Managers bei MCS. Gleichermaßen muss der Makler für jeden der m Tasks die Gebote der ω Klienten vergleichen. Des Weiteren werden für das Clustern von Tasks deren Verwandtschaften auf benachbarter Klienten überprüft und durch eine Gewichtung der Gebote berücksichtigt. Der Berechnungsaufwand ist daher proportional zu dem von MCS:

$$P_{MAS} = P_{MCS} \sim m \cdot \omega \cdot (1 + \varphi \cdot v_{max})$$

Mittels der obigen Analyse kann nun der folgende Vergleich gezogen werden: Der Rechenaufwand für die Verarbeitung der Kommunikationsdaten in Schritt 1., ist bei allen drei Methoden aufgrund der vergleichbar hohen Datenmengen aus Abschnitt 7.3.2 annähernd gleich, wobei das KHS bei aktivem Clustern von Tasks tendenziell einen geringfügig höheren Rechenaufwand aufweist. Hingegen fällt der Rechenaufwand bei der Taskzuordnung in Schritt 2. seitens des KHS deutlich niedriger aus als bei MCS und MAS. Durch die lokale Überprüfung der Taskeignung, ist der Aufwand des KHS proportional zur maximalen Taskanzahl auf die sich Kern im Netz bewirbt. Hingegen ist der Rechenaufwand bei MCS und MAS aufgrund der zentralen und rekursiven Überprüfung aller Taskeignungen proportional zur Anzahl an Tasks, Kernen sowie der geltenden Task-Verwandtschaftsbeziehungen im Netz.

Fazit: Das KHS verursacht bei einer zunehmenden Anzahl von Tasks und Kernen einen zunehmend geringeren Rechenaufwand als MCS und MAS.

7.3.4. Vergleichsanalyse von Echtzeitaspekten

Es folgt die Vergleichsanalyse der Echtzeitfähigkeit von Selbst-X Eigenschaften bei KHS, MCS und MAS. Wie bereits in der früheren Forschungsarbeit [vR12] gezeigt

wurde, unterstützt das KHS aufgrund seiner schrittweisen Taskzuordnung in aufeinanderfolgenden H-Zyklen die folgenden zyklisch bemessenen Zeitschranken bezüglich der Selbst-Konfiguration, -Heilung und -Optimierung von m Tasks:

$$\text{Selbst-Konfiguration} := h \tag{7.12a}$$

$$\text{Selbst-Heilung} := h + a \tag{7.12b}$$

$$\text{Selbst-Optimierung} := h \tag{7.12c}$$

$$h := \begin{cases} m_{max} + v_{max} & \leq 2m - 1 \quad : \text{klassisch} \\ m_{max} & \leq m \quad : \text{aggressiv} \end{cases}$$

$$a := \text{Hormon-Verfallszeit}$$

Da die eigentliche Berechnung der Taskzuordnung bei MCS und MAS ebenfalls schrittweise in aufeinanderfolgenden Kommunikationszyklen erfolgt, können für deren Selbst-X Eigenschaften gleichermaßen zyklisch bemessene Zeitschranken angegeben werden. Dabei gilt für die Taskzuordnung innerhalb eines Kommunikationszyklus das folgende Funktionsprinzip:

- Die innerhalb eines Kommunikationszyklus berechnete Taskzuordnung wird explizit an alle Kerne im Netz kommuniziert.
- Durch die zyklische Berechnung der Taskzuordnung wird die Reaktions- und Funktionsfähigkeit von Kernen nicht beeinträchtigt, so dass deren Auswirkungen auf das Verhalten von Kernen gleichermaßen in allen darauffolgenden Zuordnungen berücksichtigt werden können. Dadurch ist es möglich mindestens eine Task in jedem Zyklus fest zuzuordnen, was letztlich zu einer Gesamtverteilung von m Tasks in m Kommunikationszyklen führt.

Unter Berücksichtigung einer zyklisch basierten Taskzuordnung gemäß dem obigen Funktionsprinzip, lassen sich für die Selbst-X Eigenschaften von MCS und MAS gleichermaßen die in Gleichung 7.12a, 7.12b und 7.12c aufgeführten Zeitschranken ableiten.

Fazit: Bezüglich der Echtzeitfähigkeit können für alle drei Methoden harte Zeitschranken garantiert werden, die eine Selbst-Konfiguration, -Heilung und -Optimierung von m Tasks in m Zyklen sicherstellen.

7.3.5. Zusammenfassung der Ergebnisse

Abschließend können aus den hiesigen Vergleichsanalysen sowie dem grundlegenden Funktionsprinzip der Taskverwaltungsmethoden aus Abschnitt 7.2 die folgenden Ergebnisse und Schlüsse zwischen den Methoden KHS, MCS und MAS zusammengefasst und gezogen werden.

Bei der Betrachtung des Funktionsprinzips der Methoden MCS und MAS in Abschnitt 7.2 wurde deutlich, dass nur das KHS die Umsetzung eines vollständig dezentralen Taskverwaltungsmodells bietet. Dadurch ist die Taskverteilung mittels KHS absolut fehlertolerant (kein Single-Point-of-Failure) und nutzt die vorgegebenen Redundanzen des Systems maximal aus, was zu einer höheren Ausfallsicherheit des Systems beiträgt. Dagegen erfolgt bei MCS und MAS eine zentrale oder nur teilweise dezentrale Taskverwaltung die mittels Controllern und Agenten umgesetzt wird und nur durch den Einsatz zusätzlicher Redundanz fehlertolerant ist. Weiterführend bietet das KHS und MCS im Bereich der Selbst-Eigenschaften eine vollwertige Umsetzung der Selbst-Konfiguration, -Heilung und -Optimierung. Bei MAS hingegen stellt die Heilung lediglich einen integrativen Bestandteil dar, der mittels zusätzlicher Kommunikation von Zustandsdaten zwischen Tasks und Makler-Agenten umgesetzt werden kann.

Betreffs des Flächenaufwands im Vergleich zur Ausfallsicherheit in Abschnitt 7.3.1 lässt sich festhalten, dass eine Hardware-Implementierung des KHS einen statischen Flächenaufwand benötigt. Hingegen ist der Flächenverbrauch bei MCS und MAS abhängig von der Höhe der Ausfallsicherheit sowie der Größe des Systems (Kernanzahl). Daraus folgt, dass das KHS bereits ab einer geringen Ausfallsicherheit von 20% einen deutlich niedrigeren Flächenverbrauch erzielt als MCS und MAS.

Bei der Betrachtung des Kommunikationsaufwands in Abschnitt 7.3.2 wurde deutlich, dass alle drei Methoden einen annähernd vergleichbaren Aufwand aufweisen. Lediglich beim Clustern von Tasks konnte seitens des KHS eine tendenziell höherer Aufwand nachgewiesen werden, der aber im Vergleich zur Taskkommunikation vernachlässigbar ist.

In Abschnitt 7.3.3 konnte gezeigt werden, dass bei einer softwarebasierten Implementierung der drei Methoden ein unterschiedlicher Rechenaufwand auf Systemebene anfällt. Dabei konnte das KHS den geringsten Aufwand vorweisen, der proportional zur Anzahl der verteilenden Taskmenge ist. Im Gegenzug konvergiert der Flächenaufwand bei allen drei Methoden gegen Null.

Gegenüber der Echtzeitfähigkeit von Selbst-X Eigenschaften konnten in Abschnitt 7.3.4 bei allen drei Methoden harte Zeitschranken nachgewiesen werden.

Tabelle 7.1 zeigt eine Zusammenfassung und Gegenüberstellung der Ergebnisse aus den Vergleichsanalysen.

Abschließendes Fazit zum Vergleich des KHS gegenüber MCS und MAS: Das KHS bietet unter Berücksichtigung aller Aspekte einen wesentlichen Vorteil gegenüber einem MCS oder MAS betriebenen System. Durch die vollständig dezentrale Taskverteilung werden alle Selbst-Eigenschaften als integraler Bestandteil umgesetzt und in Echtzeit ausgeführt. Des Weiteren gilt bereits ab einer geringen Ausfallsicherheit im System ein deutlich niedrigerer Flächenaufwand seitens einer hardwarebasierten Implementierung lokaler KHS-Instanzen. Im Gegenzug ist bei einer softwarebasierten Implementierung des KHS der Rechenaufwand für die Taskverteilung

proportional zur Taskanzahl und somit wesentlich kleiner als bei MCS und MAS.

	Methoden		
	KHS	MCS	MAS
Modell	dezentral symmetrisch	zentral asymmetrisch	teils dezentral asymmetrisch
Task-Cluster	passiv durch Akzeleratoren	aktiv durch Berechnung	aktiv durch Berechnung
Konfiguration	integral H-Zyklen	integral Kernzustand	integral Auktion
Heilung	integral H-Zyklen	integral Taskzustand	integrativ Taskzustand
Optimierung	integral H-Zyklen	integral Kernzustand	integral Auktion
Fläche vs. Ausfallsicherheit	$R_f \geq 20\%$ $\Rightarrow A_{KHS} < A_{MCS}$	$R_f < 20\%$ $\Rightarrow A_{MCS} < A_{KHS}$	$R_f < 20\%$ $\Rightarrow A_{MAS} < A_{KHS}$
Kommunikation	11.82%	8.61%	8.43%
Rechenzeit	$\sim m_{max}$	$\sim m \cdot \omega$	$\sim m \cdot \omega$
Echtzeitfähigkeit	m Zyklen	m Zyklen	m Zyklen

Tabelle 7.1.: Gegenüberstellung der Ergebnisse aus den Vergleichsanalysen in Abschnitt 7.3.1, 7.3.2, 7.3.3 und 7.3.4 für KHS, MCS und MAS

8. Zusammenfassung und Ausblick

Diese Arbeit stellt ein organisches Taskverarbeitungssystem vor, dass die zuverlässige Verwaltung und Verarbeitung von Tasks auf Multi-Core basierten SoC-Architekturen umsetzt. In Kapitel 1 wurde die Motivation für die Entwicklung des Systems vorgestellt. Aufgrund der zunehmenden Integrationsdichte treten bei der planaren Fertigung vermehrt Nebeneffekte auf, die im Systembetrieb zu Fehler und Ausfällen von Komponenten führen, was die Zuverlässigkeit der SoCs zunehmend beeinträchtigt. Bereits ab einer Fertigungsgröße von weniger als 100 nm ist eine drastische Zunahme von Elektromigration und der Strahlungssensitivität zu beobachten. Gleichzeitig nimmt die Komplexität (Applikations-Anforderungen) weiter zu, wobei der aktuelle Trend auf eine immer stärkere Vernetzung von Geräten abzielt (Ubiquitäre Systeme). Um diese Herausforderungen autonom bewältigen zu können, wurde das biologisch inspirierte Systemkonzept in Kapitel 2 vorgestellt. Dieses bedient sich der Eigenschaften und Techniken des menschlichen endokrinen Hormonsystems und setzt ein vollständig dezentrales Funktionsprinzip mit Selbst-X Eigenschaften aus dem des Organic Computing Bereich um. Die Durchführung dieses organischen Funktionsprinzips erfolgt in zwei getrennten Regelkreisen, die gemeinsam die dezentrale Verwaltung und Verarbeitung von Tasks übernehmen. Der erste Regelkreis wird durch das künstliche Hormonsystem (KHS) abgebildet und führt die Verteilung aller Tasks auf die verfügbaren Kerne durch. Die Verteilung erfolgt durch das Mitwirken aller Kerne und berücksichtigt deren lokale Eignung und aktueller Zustand. Anschließend erfolgt die Synchronisation mit dem zweiten Regelkreis, der durch die hormongeregelte Taskverarbeitung (HTV) abgebildet wird und einen dynamischen Task-Transfer gemäß der aktuellen Verteilung vollzieht. Dabei werden auch die im Netz verfügbaren Zustände von Tasks berücksichtigt und es entsteht ein vollständiger Verarbeitungspfad, ausgehend von der initialen Taskzuordnung, hinweg über den Transfer der Taskkomponenten, gefolgt von der Erzeugung der lokalen Taskinstanz bis zum Start des zugehörigen Taskprozess auf dem jeweiligen Kern. In Kapitel 6 wurde die System-Implementierung durch modulare Integration spezifischer Hardware- und Software-Komponenten vorgestellt. Dadurch kann das System entweder vollständig in Hardware, Software oder in hybrider Form betrieben und genutzt werden. Mittels eines FPGA-basierten Prototypen, konnten die formal bewiesenen Zeitschranken aus Kapitel 4 durch Messungen in realer Systemumgebung bestätigt werden. Die Messergebnisse zeigen herausragende Zeitschranken bezüglich der Selbst-X Eigenschaften, die für andere Regelungstechniken nur schwer zu erzielen sind. Des Weiteren zeigt der quantitative Vergleich gegenüber anderen Systemen, das

der hier gewählte dezentrale Regelungsansatz bezüglich Ausfallsicherheit, Flächen- und Rechenaufwand deutlich überlegen ist.

Fazit: Die Verwendung des organischen Taskverarbeitungssystems führt zu einer erheblichen Steigerung der Zuverlässigkeit von SoCs, wobei die natürlichen Redundanzen der Hardwarearchitektur intelligent und effizient ausgeschöpft werden. Das System hat signifikante Vorteile gegenüber anderen Systemen. Durch die vollständig dezentrale Taskverteilung werden alle Selbst-X Eigenschaften als integraler Bestandteil umgesetzt und in Echtzeit ausgeführt. Des Weiteren gilt bereits ab einer geringen Ausfalltoleranz von Kernen ein deutlich niedrigerer Flächen- und Rechenaufwand gegenüber Multi-Agenten und -Controller basierten Systemen.

Für das erforschte Taskverarbeitungssystem lassen die folgenden Kern-Eigenschaften festhalten:

- Es nutzt die vorgegebene Ausfalltoleranz von Kernen maximal aus, so dass der Systembetrieb sichergestellt ist, solange genügend Task verarbeitende Kerne im System verfügbar sind.
- Es setzt die die Selbst-X Eigenschaften -Konfiguration, -Optimierung in autonomer Form um.
- Es verzichtet auf den Einsatz zentraler Steuerungsmechanismen, so dass der Bestandteil eines Single of Point Failure vollständig vermieden wird.
- Es erkennt selbständig einen sich verändernden Systemzustand, der mittels der Selbst-X Eigenschaften -Heilung und -Optimierung re- oder proaktiv adaptiert wird.
- Es setzt die hier aufgeführten Anforderungen Ressourcen schonend um (geringer Speicherverbrauch, geringe Rechenzeit, Flächenaufwand etc.)
- Es ist Skalierungs fähig und kann auch für große SoC-Designs mit einer hohen Anzahl an Tasks und Kernen verwendet werden.

8.1. Ausblick auf mögliche Entwicklungen

Für weitere Entwicklung lassen die die folgenden Forschungsziele festhalten:

Mixed-Signal Taskverarbeitung: Das hormongeregelte Taskverarbeitungsmodell erlaubt den dynamischen, hormongestützten Transfer von Tasks im Netz. Das System eignet sich daher auch hervorragend für die Mixed-Signal Verarbeitung. Für

eine umfassende Signalbereich übergreifende Verarbeitung sind neben dem Zustandstransfer und der Instanzerzeugung eventuell auch weitere Transformationsschritte von Taskkomponenten zu berücksichtigen. Damit eine Task auf analogen wie auch digitalen Kernen plattformunabhängig ausgeführt werden kann, sind bei einer heterogenen Kern-Architektur unterschiedliche Beschreibungen einer Task notwendig. Diese könnten entweder durch eine einheitlich interpretationsfähige Beschreibungsform direkt im System abgelegt werden oder mittels lokaler Just-in-Time Kompilierung in das entsprechende Kern-Format überführt werden. Das gleiche gilt für Zustandsdaten, die unterschiedlich zu interpretieren sind. Gegebenenfalls müssten beim Zustandstransfer interne Transformationsprozesse ergänzt werden.

Hierarchisches Design: Derzeit beschränkt sich der bisherige Ansatz auf SoC Designs (Single-Chip Variante). Eine Erweiterung auf verteilte Systeme könnte weitere Erkenntnisse über die Skalierbarkeit des organischen Funktionsprinzips für große Maßstäbe liefern (> 1000 Kerne). Hierfür müsste das eindimensionale Funktionsverhalten durch ein hierarchisches Prinzip ergänzt werden, das zunächst die übergeordnete Verteilung aller Tasks auf Chip-Cluster übernimmt und anschließend die Cluster interne Verteilung und Verarbeitung von Tasks mittels des hier entwickelten Systems umsetzt. Für einen verteilten organischen Ansatz sind bereits erste Forschungsergebnisse vorzuweisen [LBB15].

Fehlerhafte Kommunikation: Derzeit beschränkt sich die Ausfallsicherheit des Systems auf das Fehlverhalten von Tasks und Kernen. Um die Gesamt-Zuverlässigkeit weiter zu erhöhen, ist eine Ausweitung des organischen Konzepts auf das globale Kommunikationsnetzwerk erforderlich. Derzeit führt der Ausfall einer lokalen Kommunikationsschnittstelle zum Ausfall des zugehörigen Kerns, auch wenn dieser weiterhin funktionsfähig ist. Des Weiteren kann durch fehlerhaft übertragene Hormone das organische Funktionsprinzip gestört werden. Hier gilt es weitere Techniken zu erforschen, die eine zuverlässige Hormonkommunikation gewährleisten. Eine Möglichkeit wäre eine sekundäre, hormongestützte Verwaltung von redundanten Kommunikationsschnittstellen und -Kanälen.

Analyse des Energieverbrauchs: Um Aussagen über den Energieverbbauch des Systems treffen zu können, sind weitere Praxistests unter verschiedenen Szenarien notwendig. Hierfür muss zunächst ein entsprechendes Energiemodell erstellt werden, mit welchem sich anhand der Dimensionierung von Systemparametern der Leistungsverbrauch des Systems vorab vorhersagen lässt. Dadurch wären weitere Aussagen über die Verwendbarkeit gegenüber thermisch wie auch Energie kritischen SoC-Applikationen möglich.

A. Anhang

A.1. Permanente und Transiente Fehlermodelle

Die Simulation von Task- und Kernausfällen auf dem Prototypen in Kapitel 6, erfolgt durch zwei stochastische Prozesse die innerhalb sogenannter Event-Handler der KHS-API zur Verfügung gestellt werden. Der erste Prozess modelliert eine spezifische Fehlerrate, welche das Alterungs- und Temperaturverhalten sowie die Strahlungsbelastung von Kernen modelliert. Der zweite Prozess simuliert dann beim Erreichen der maximalen Fehlerrate (1) eines Fehlermodells einen oder mehrere Task- oder Kernausfälle. Deren Selektion erfolgt über eine statische Wahrscheinlichkeit, die für jede Task und jeden Kern individuell vergeben werden kann.

Temperatur behaftetes Fehlermodell Das reaktive Modell A.1 dient zur Berechnung der temperaturabhängigen Fehlerrate die als Funktion der Kerntemperatur definiert ist. Liegt die Temperatur eines Kerns unterhalb des definierten Schwellwerts $Temp_{low}$, befindet sich diese im zulässigen Bereich und die Fehlerrate ist Null. Wird hingegen der untere Schwellwert überschritten, befindet sich die Kerntemperatur zunehmend außerhalb des zulässigen Wertebereichs und die Basis-Fehlerrate $Base_{Temp}$ nimmt linear in Abhängigkeit des Temperaturgradienten $Grad_{Temp}$ zu. Erreicht die Temperatur den kritischen Schwellwert $Temp_{high}$, ist die Fehlerrate maximal was den Ausfall des Kerns in Folge von Überhitzung repräsentiert.

$$FR_{Temp} = \begin{cases} 0 & : Temp < Temp_{low} \\ Base_{Temp} + Grad_{Temp} \cdot (Temp - Temp_{low}) & : Temp_{low} \leq Temp \leq Temp_{high} \\ 1 & : Temp > Temp_{high} \end{cases} \tag{A.1}$$

Der Temperaturverlauf eines Kerns kann unter der Berücksichtigung der Gleichungen A.2, A.3 und A.4 modelliert werden. Gleichung A.2 modelliert die Kerntemperatur zu einem beliebigen Zeitschritt t als lineare Funktion des Temperatugradienten, in der $Temp(t)$ rekursiv aus dem vorangegangenen Niveau $t-1$ und der aktuellen Änderungsrate $\triangle Temp(t)$ berechnet wird. Die Änderungsrate $\triangle Temp(t)$ wird in Gleichung A.3 als Differenz zweier Temperaturniveaus berechnet, die durch Zufuhr und Abgabe von Verlust- und Wärmeleistung erzeugt werden. Das Niveau $TempLoad(t-1)$ (A.4) stellt die thermische Belastung des Kerns durch die Taskverarbeitung zum Zeitschritt $t-1$ dar. Die Differenz der beiden Niveaus $Temp(t-1)$

und $Temp_{Base}$ reguliert das Verhältnis zwischen Abgabe/Zufuhr von Wärmeleistung an oder aus der Umgebung (Luft, Öl etc.) erfolgt. Diese wird zusätzlich durch den Wärmeleitkoeffizienten $HeadCond$ beeinflusst. Der Faktor $Scale$ reguliert den eigentlichen physikalischen Einfluss der Verlust- und Wärmeleistung auf den Temperaturgradienten.

$$Temp(t) = Temp(t-1) + \triangle Temp(t) \tag{A.2}$$

$$\triangle Temp(t) = Scale \cdot TempLoad(t-1) - (Temp(t) - Temp_{Base}) \cdot HeatCond \tag{A.3}$$

$$TempLoad(t) = \sum_{i=1...m} TaskTempLoad_i \tag{A.4}$$

Alterungs behaftetes Fehlermodell Äquivalent zu A.1 kann die altersbedingte Ermüdung eines Kerns durch die zugehörige Fehlerrate in Gleichung A.5 ausgedrückt werden. Erreicht ein Kern die untere Altersgrenze Age_{low} treten zunehmend Alterungseffekte auf und die Fehlerrate $Base_{Age}$ nimmt stetig in Korrelation zum Alter Age und dem Gradienten $Grad_{Age}$ zu. Die obere Grenze Age_{high} repräsentiert die maximale Betriebsdauer bis zum totalen Ausfall des Kerns als Folge altersbedingter Ermüdung.

$$FR_{Age} = \begin{cases} 0 & : Age < Age_{low} \\ Base_{Age} + Grad_{Age} \cdot (Age - Age_{low}) & : Age_{low} \leq Age \leq Age_{high} \\ 1 & : Age > Age_{high} \end{cases} \tag{A.5}$$

Der Alterungsprozess eines Kerns zum Zeitschritt t wird durch ein Temperatur gekoppeltes Modell in Gleichung A.6 und A.7 definiert. Die Geschwindigkeit der Alterung wird über den Gradienten $\triangle Age(t)$ reguliert, der in Abhängigkeit des Systemzustands entweder eine pro- oder regressive Wirkung auf die Alterung des Kerns ausübt. Befindet sich der Kern im aktiven Zustand, ist der Gradient $\triangle Age(t)$ positiv und die Alterung beschleunigt sich je dichter sich die Kerntemperatur am kritischen Niveau $Temp_{high}$ befindet. Der Einfluss der Kerntemperatur auf den Gradienten wird mittels $AgeTempInfluence$ reguliert. Befindet sich der Kern hingegen im passiven Zustand (IDLE, OFF), ist der Gradient negativ solange $Temp \neq Temp_{high}$ gilt. Die Alterung schrumpft, was eine Reaktivierung des Kerns im Falle einer vorzeitigen Ermüdung durch die Temperatur ermöglicht. Diese wird über den Parameter

$AgeRecover$ beeinflusst.

$$Age(t) = Age(t-1) + \triangle Age(t) \tag{A.6}$$

$$\triangle Age(t) = \begin{cases} core\ active: \\ 1 - AgeTempInfluence \cdot (1 - \frac{Temp(t-1) - Temp_{low}}{Temp_{high} - Temp_{low}}) \\ \\ core\ inactive: \\ -1 + AgeRecover \cdot (1 - \frac{Temp(t-1) - Temp_{low}}{Temp_{high} - Temp_{low}}) \end{cases} \tag{A.7}$$

Single Event Upset Fehlermodell Die Modellierung temporärer Fehler in Folge von Ionisierung durch hochenergetische Teilchen wird in Gleichung A.8 durch eine statische Fehlerrate mit $Base_{SEU} = 0..1$ individuell für jeden Kern modelliert.

$$FR_{SEU} = Base_{SEU} \tag{A.8}$$

Literaturverzeichnis

[Ada84] J.H. Adams. The effects of solar flares on single event upset rates. In *IEEE Transactions on Nuclear Science Vol:31,Iss:6*, pages 1212–1216, December 1984.

[ANS04] Chris Wright Andrew N. Sloss, Dominic Symes. *ARM System Developer's Guide: Designing and Optimizing System Software*. Morgan Kaufman, Elsevier, San Francisco, USA, 2004.

[Aug83] Stan Augarten. *The Most Widely Used Computer on a Chip: The TMS 1000*. Ticknor and Fields, New Haven/York, USA, September 1983.

[BBB+06] Jürgen Becker, Kurt Brändle, Uwe Brinkschulte, Jörg Henkel, Wolfgang Karl, Thorsten Köster, Michael Wenz, and Heinz Wörn. Digital on-demand computing organism for real-time systems. In *19th International Conference on Architecture of Computing System (ARCS 2006)*, pages 230–245, January 2006.

[BBP13] Benjamin Betting, Uwe Brinkschulte, and Mathias Pacher. Evaluation and Superiority Analysis of a Decentralized Task Control Mechanism for Dependable Real-Time SoC Architecture. In *16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013)*, Paderborn, Germany, June 2013.

[BLBB13] Michael Bauer, Daniel Lohn, Benjamin Betting, and Uwe Brinkschulte. Design and evaluation of an adaptive real-time microprocessor. In *16th IEEE International Conference on Evolvable Systems (ICES 2013)*, Singapore, April 2013.

[Ble76] I. A. Blech. Electromigration in thin aluminum films on titanium nitride. In *Journal of Applied Physics Vol.47,Iss.4*, pages 1203–1208, April 1976.

[BP12] Uwe Brinkschulte and Mathias Pacher. An agressive strategy for an artificial hormone system to minimize the task allocation time. In *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW 2012)*, pages 188–195, Shenzen, China, April 2012.

[BPB12] Benjamin Betting, Mathias Pacher, and Uwe Brinkschulte. Development and Evaluation of a Self-Adaptive Organic Middleware for Highly Dependable System-on-Chips. In *8th International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, St. Maarten, Netherlands Antilles, March 2012.

[BPvR08] Uwe Brinkschulte, Mathias Pacher, and Alexander von Renteln. An artificial hormone system for self-organizing real-time task allocation in organic middleware. In *Organic Computing - Understanding Complex Systems*, pages 261–283, Berlin, Germany, 2008. Springer Berlin Heidelberg.

[BU07] Uwe Brinkschulte and Theo Ungerer. *Mikrocontroller und Mikroprozessoren Aufl.2*. Springer Berlin, Berlin, Germany, 2007.

[BvRBH13] Benjamin Betting, Julius von Rosen, Uwe Brinkschulte, and Lars Hedrich. A Highly Dependable Self-Adaptive Mixed-Signal Multi-Core System-on-Chip. In *26th International Conference on Architecture of Computing Systems (ARCS 2013)*, pages 122–133, Prague, Czech Republic, February 2013.

[BvRP08] Uwe Brinkschulte, Alexander von Renteln, and Mathias Pacher. Measuring the quality of an artificial hormone system based task mapping. In *Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems, Autonomics 2008*, page 32, Turin, Italy, September 2008.

[Cao04] Junwei Cao. Self-organizing agents for grid load balancing. In *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, pages 388–395, Pittsburgh, USA, November 2004.

[CM08] Ewerson Carvalho and Fernando Moraes. Congestion-aware task mapping in heterogeneous mpsocs. In *International Symposium on System-on-Chip (SOC 2008)*, pages 1–4, Tampere, Finland, November 2008.

[Dhi13] B.S. Dhillon. *Computer System Reliability: Safety and Usability*. CRC Press, Florida, USA, 2013.

[EFH09] T. Ebi, M. A. Al Faruque, and J. Henkel. Tape: Thermal-aware agent-based power economy for multi/many-core architectures. In *17th IEEE International Conference On Computer Aided Design*, pages 302–309, San Jose, USA, November 2009.

[ERHH11] Thomas Ebi, Holm Rauchfuss, Andreas Herkersdorf, and Jörg Henkel. Agent-based thermal management using real-time i/o communication

relocation for 3d many-cores. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation (PATMOS 2011)*, pages 112–121, Madrid, Spain, September 2011.

[FSH+09] Federico Faggin, Masatoshi Shima, Marcian Hoff, Hal Feeney, and Stanley Mazor. The mcs-4 - an lsi micro computer system [reprint]. In *IEEE Solid-State Circuits Magazine Vol.1,Iss.1*, pages 55–60, Santa Clara, USA, Winter 2009.

[Fur89] Steve Furber. *VLSI RISC Architecture and Organization (Electrical and Computer Engineering Series)*. CRC Press, New York, USA, 1989.

[Fur00] Steve Furber. *ARM System-on-Chip Architecture*. Addison Wesley, Boston, USA, 2000.

[Gol04] K. Golenhofen. *Basislehrbuch Physiologie: Lehrbuch, Kompendium, Fragen und Antworten 3. Aufl.* Urban & Fischer, München, 2004.

[HBB+11] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. Teich, N. Wehn, and H. J. Wunderlich. Design and architectures for dependable embedded systems. In *Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2011)*, pages 69–78, Taipeh, Taiwan, October 2011.

[HS95] Hans U. Heiss and Michael Schmitz. Decentralized dynamic load balancing: The particles approach. In *Information Sciences Vol.84,Iss.1-2*, pages 115–128, New York, USA, May 1995.

[Ibe15] Eishi H. Ibe. *Terrestrial radiation effects in ULSI devices and electronic systems*. Wiley, Singapore, 2015.

[II13] Inc. IC Insights. Mcclean report 2013. Technical report, IC Insights, Inc., Arizona, USA, January 2013.

[Joh93] G.H. Johnson. Simulating single-event burnout of n-channel power mosfet's. In *IEEE Transactions on Electron Devices Vol:40,Iss:5*, pages 1001–1008, May 1993.

[Kim11] Choong U. Kim. *Electromigration in Thin Films and Electronic Devices*. Woodhead Publishing, Camebridge, Great Britain, 2011.

[KMUU06] Florian Kluge, Jörg Mische, Sascha Uhrig, and Theo Ungerer. Car-soc - towards an autonomic soc node. In *ACACES 2006 poster abstracts*, page 268, L'Aquila, Italy, July 2006.

[KT98] B.D. Knowlton and C.V. Thompson. Simulation of the temperature and current density scaling of the electromigration-limited reliability of near-bamboo interconnects. In *Journal of Materials Research Vol:13,Iss:5*, pages 1164–1170, May 1998.

[Lan14] James A. Langbridge. *Professional embedded ARM development*. John Wiley & Sons, Indianapolis, USA, 2014.

[LBB15] Andreas Lund, Benjamin Betting, and Uwe Brinkschulte. Design and Evaluation of a bio-inspired, distributed Middleware for a Multiple Mixed-Core System on Chip. In *18th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2015)*, Auckland, New Zealand, April 2015.

[LPB11] Daniel Lohn, Mathias Pacher, and Uwe Brinkschulte. A generalized model to control the throughput in a processor for real-time applications. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2011)*, pages 83–88, Newport Beach, USA, March 2011.

[Ltd96] ARM Ltd. Arm 7100: Preliminary data sheet. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0035a/DDI0035A_7100_prelim_ds.pdf`, January 1996. [Online; accessed 03-January-2016].

[Ltd01] ARM Ltd. Arm7tdmi-s: Technical reference. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0234b/DDI0234.pdf`, September 2001. [Online; accessed 03-January-2016].

[Ltd14] ARM Ltd. Mobile and embedded/iot market overview and trends. `http://www.arm.com/files/event/1_2014_Physical_IP_Workshop_ARM_Embedded_Market.pdf`, June 2014. [Online; accessed 03-January-2016].

[MD03] Hans P. Messmer and Klaus Dembowski. *PC Hardwarebuch: Aufbau, Funktionsweise, Programmierung*. Addison Wesley Verlag, Bonn, Germany, 2003.

[Moo09] Gordon E. Moore. Cramming more components onto integrated circuits [reprint]. In *IEEE Solid-State Circuits Society Newsletter Vol.11,Iss.5*, pages 33–35, New York, USA, February 2009.

[Moo15] Gordon Moore. 50th anniversary of moore's prediction: Gordon moore - the man whose name means progress. Technical report, IEEE Spectrum, March 2015.

[MS14] ARM Ltd. Matt Spencer. The arm-powered mobile internet. `http://images.tmcnet.com/expo/devcon5/ppt-ny2014/K-2MattSpencer.pdf`, February 2014. [Online; accessed 03-January-2016].

[NB08] Manuel Nickschas and Uwe Brinkschulte. Carisma - a service-oriented, real-time organic middleware architecture. In *Journal of 6th International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS 2008)*, pages 654–663, Capri Island, Italy, October 2008.

[NCW+92] D.K. Nichols, J.R. Cross, R.K. Watson, H.R. Schwartz, and R.L. Pease. An observation of proton-induced latchup [in cmos microprocessor]. In *IEEE Transactions on Nuclear Science Vol:39,Iss:6*, pages 1654–1656, December 1992.

[Rot02] Andreas Roth. *Das Microcontroller Kochbuch MCS51*. MITP-Verlag, Bonn, Germany, 2002.

[Ryu99] Changsup Ryu. Microstructure and reliability of copper interconnects. In *IEEE Transactions on Electron Devices Vol:46,Iss:6*, pages 1113–1120, June 1999.

[SBN81] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *TMSI000/1200: Chip Architecture and Operation*. Mcgraw-Hill College, New York, USA, September 1981.

[Sch11] Coordinator H. Schmeck. Dfg spp 1183 - organic computing. `http://projects.aifb.kit.edu/effalg/oc/inhalte/index.php`, 2011. [Online; accessed 08-February-2016].

[SDD06] Mohsen Amini Salehi, Hossain Deldari, and Bahare Mokarram Dorri. Mlblm: A multi-level load balancing mechanism in agent-based grid. In *LNCS:Distributed Computing and Networking Vol.4308*, pages 157–162, Guwahati, India, December 2006.

[SJPS09] Amit Kumar Singh, Wu Jigang, Alok Prakash, and Thambipillai Srikanthan. Mapping algorithms for noc-based heterogeneous mpsoc platforms. In *12th Euromicro Conference on Digital System Design (DSD 2009)*, pages 133–140, Patras, Greece, August 2009.

[Smi80] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transactions on Computers Vol.C-29,Iss.12*, pages 1104–1113, Washington, USA, December 1980.

[SP82] Carlo H. Séquin and David A. Patterson. Design and implementation of risc i. Technical Report UCB/CSD-82-106, EECS Department, University of California, Berkeley, October 1982. [Online; accessed 03-January-2016].

[SPJ94] G.M. Swift, D.J. Padgett, and A.H. Johnston. A new class of single event hard errors [dram cells]. In *IEEE Transactions on Nuclear Science Vol:41,Iss:6*, pages 2043–2048, December 1994.

[(TM15] Transparency Market Research (TMR). System-on-chip (soc) market - global industry analysis, size, share, growth, trends and forecast 2015-2021. Technical report, Transparency Market Research (TMR), Albany, New York, July 2015.

[Vai10] Subramanian Vaidyanathan. Multiple gate field-effect transistors for future cmos technologies. In *IETE Technical Review Vol:27,Iss:6*, pages 446–454, December 2010.

[vR12] Alexander von Renteln. Eine organische middleware für verteilte echtzeitsysteme. In *Dissertation*, 2012.

[vRB10] Alexander von Renteln and Uwe Brinkschulte. Implementing and evaluating the ahs organic middleware - a first approach. In *13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORC 2010)*, pages 163–169, Sevilla, Spain, May 2010.

[vRSH+15] Julius von Rosen, Felix Salfelder, Lars Hedrich, Benjamin Betting, and Uwe Brinkschulte. A highly dependable self-adaptive mixed-signal multi-core system-on-chip architecture. In *Integration, the VLSI Journal, Volume 48*, pages 55–71, Amsterdam, Netherlands, January 2015. Elsevier.

[Wak79] John F. Wakerly. The intel mcs-48 microcomputer family: A critique. In *IEEE Computer Journal Vol.12,Iss.2*, pages 22–31, New York, USA, February 1979.

[Wal08] Jürgen Walter. *Mikrocomputertechnik mit der 8051-Controller-Familie*. Springer Berlin, Berlin, Germany, 2008.

[WZS+14] Yizhuo Wang, Yang Zhang, Yan Su, Xiaojun Wang, Xu Chen, Weixing Ji, and Feng Shi. An adaptive and hierarchical task scheduling scheme for multi-core clusters. In *Parallel Computing - System & Applications*, pages 611–627, Beijing, China, 2014. Elsevier B.V.

[Yiu13] Joseph Yiu. *Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors Aufl.3*. Newnes, Newton, USA, 2013.

Lebenslauf

Persönliche Daten

Vorname:	Benjamin
Nachname:	Betting
Nationalität:	Deutsch
Geburtsort:	Lörrach, Deutschland
Geburtsdatum:	02. Dezember 1985

Ausbildung

1992–2005	Schulausbildung Abschluss: Technisches Gymnasium (TG) Lörrach
2005–2006	Zivildienst in Fachklinik Haus Weitenau
2006–2011	Studium der Informatik an der Goethe-Universität Frankfurt 2009 Abschluss: Bachelor of Science 2011 Abschluss: Master of Science
2011–Heute	Wissenschaftlicher Mitarbeiter an der Professur für Eingebettete Systeme von Prof. Uwe Brinkschulte des Fachbereichs Informatik und Mathematik der Johann Wolfgang Goethe-Universität. Forschung im Bereich zuverlässiger SoCs.

Publikationen

[1] von Rosen, J., Salfelder, F., Hedrich, L., Betting, B., and Brinkschulte, U. A highly dependable self-adaptive mixed-signal multi-core system-on-chip architecture. In *Integration, the VLSI Journal, Volume 48*, pages 55–71, Amsterdam, Netherlands, January 2015. Elsevier.

[2] Lund, A., Betting, B., and Brinkschulte, U. Design and Evaluation of a bio-inspired, distributed Middleware for a Multiple Mixed-Core System on Chip. In *18th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2015)*, Auckland, New Zealand, April 2015.

[3] Betting, B. and Brinkschulte, U. A simulator to validate the concept of artificial dna for self-building embedded systems. In *1st IEEE International Workshop on Self-Improving System Integration (SISSY 2014)*, London, England, September 2014.

[4] Betting, B. and Brinkschulte, U. Analyzing the Overhead of Self-Optimization through Task Migration Within a Decentralized Task Control Mechanism for Dependable System-on-Chip Architectures. In *17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2014)*, pages 84–91, Reno, USA, June 2014.

[5] Brinkschulte, U., Pacher, M., von Renteln, A., and Betting, B. Organic real-time middleware. In *Self-Organization in Embedded Real-Time Systems*, pages 179–208, New York, USA, 2013. Springer New York.

[6] Betting, B., von Rosen, J., Brinkschulte, U., and Hedrich, L. A Highly Dependable Self-Adaptive Mixed-Signal Multi-Core System-on-Chip. In *26th International Conference on Architecture of Computing Systems (ARCS 2013)*, pages 122–133, Prague, Czech Republic, February 2013.

[7] Betting, B., Brinkschulte, U., and Pacher, M. Evaluation and Superiority Analysis of a Decentralized Task Control Mechanism for Dependable Real-Time SoC Architecture. In *16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013)*, Paderborn, Germany, June 2013.

[8] Bauer, M., Lohn, D., Betting, B., and Brinkschulte, U. Design and evaluation of an adaptive real-time microprocessor. In *16th IEEE International Conference on Evolvable Systems (ICES 2013)*, Singapore, April 2013.

[9] von Rosen, J., Betting, B., Brinkschulte, U., and Hedrich, L. Ein hochverlässliches, selbst-adaptives, Mixed-Signal Mehrkern-System-on-Chip. In *6. GMM/GI/ITG-Fachtagung für Zuverlässigkeit und Entwurf (ZuE 2012)*, Bremen, Deutschland, September 2012.

[10] Leineweber, C., Pacher, M., Betting, B., von Rosen, J., Brinkschulte, U., and Hedrich, L. Detection and Defense Strategies Against Attacks on an Artificial Hormone System Running on a Mixed Signal Chip. In *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2012)*, Shenzhen, China, April 2012.

[11] Betting, B., Pacher, M., and Brinkschulte, U. Development and Evaluation of a Self-Adaptive Organic Middleware for Highly Dependable System-on-Chips. In *8th International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, St. Maarten, Netherlands Antilles, March 2012.