FH Vorarlberg
University of Applied Sciences

Heterogenes Rechnen mit ARM und DSP Multiprozessor-Ein-Chip-Systemen

Masterarbeit
zur Erlangung des akademischen Grades

Master of Science in Engineering (MSc)

Fachhochschule Vorarlberg
Master Informatik

Betreut von
Prof. (FH) DI Patrick Ritschel

Vorgelegt von
Johannes Lang, BSc

Dornbirn, August 2020

Kurzreferat

Heterogenes Rechnen mit ARM und DSP Multiprozessor Ein-Chip-Systemen

Diese Arbeit untersucht ARM und DSP Multiprozessor Ein-Chip-Systeme von Analog Devices hinsichtlich deren Programmierung, Fähigkeiten und Limitierungen. Durch die Integrierung von unterschiedlichen Hardware-Beschleunigern und Prozessoren in Ein-Chip-Systeme wird echte Nebenläufigkeit ermöglicht. Allerdings wird durch die Integrierung mehrerer Prozessoren die Komplexität der Programmierung von Ein-Chip-Systemen erhöht. Im Zuge dieser Arbeit wird untersucht, was bei der Programmierung von ARM und DSP Ein-Chip-Systemen hinsichtlich der heterogenen Prozessoren und Peripheriebausteinen beachtet werden muss. Dabei werden zuerst die Gründe für heterogenes Rechnen und die Trendwende zu Multiprozessorsystemen erläutert. Anschließend wird der aktuelle Stand der Technik erarbeitet und Programmiermodelle beschrieben, die das Programmieren von heterogenen Multiprozessorsystemen vereinfachen. Überdies werden zwei Fallbeispiele gewählt, mit denen bedeutsame Eigenheiten der Programmierung eines Ein-Chip-Systems erarbeitet werden. Im ersten Fallbeispiel werden anhand der UART-Peripherie Erkenntnisse des Ein-Chip-Systems dargelegt, die praktische Auswirkungen auf die Verwendung des Systems haben. Im zweiten Fallbeispiel wird bei der Berechnung der schnellen Fourier Transformation das heterogene System auf dessen Rechenleistung untersucht. Dabei wird die Performanz des Hardware-Beschleunigers gegenüber unterschiedlichen Software-Bibliotheken verglichen und die verschiedenartigen Implementierungen analysiert. Zudem werden durch die Performanzanalyse die Einflüsse der Speicherhierachie des Ein-Chip-Systems ermittelt. Weiterhin wird gezeigt, dass sich die Bibliotheken von Analog Devices in deren Anwendung und Performanz voneinander unterscheiden. Außerdem wird veranschaulicht, dass je nach Anwendungsfall eine nicht für DSPs ausgelegte quelloffene Implementierung konkurrenzfähig zu den optimierten Bibliotheken von Analog Devices und dem Hardware-Beschleuniger ist. Zudem wird durch die Analyse der Mehraufwand ermittelt, der für die Konfiguration des Hardware-Beschleunigers aufgebracht werden muss. Dabei wird gezeigt, dass die Verwendung des Hardware-Beschleunigers erst ab einer bestimmten Anzahl an Abtastwerten rentabel ist. Abschließend werden die zwei Fallbeispiele für einen Konzeptnachweis verknüpft, der die Möglichkeiten des heterogenen Rechnens veranschaulicht.

Abstract

Heterogeneous Computing with an ARM and DSP multicore System-on-a-chip

This thesis investigates ARM and DSP multiprocessor system on chips from Analog Devices with respect to their programming, capabilities and limitations. The integration of different hardware accelerators and processors into a system on a chip allows true concurrency. However, the integration of multiple processors increases the complexity of programming such systems. This thesis focuses on the different aspects, which must be considered when programming a system on a chip with respect to its heterogeneous processors and peripherals. First the reasons for heterogeneous computing and the trend reversal to multiprocessor systems are explained. Then the current state of the art is worked out and programming models are described, which simplify the programming of heterogeneous multiprocessor systems. In addition, two case studies are shown, to illustrate important peculiarities of programming a system on a chip. In the first case study, the UART peripheral is used to demonstrate findings of the system on a chip that have practical implications for the use of the system. In the second case study, the heterogeneous system is examined for its computing power when calculating the fast Fourier transformation. The performance of the hardware accelerator is compared to different software libraries and their different implementations are analyzed. In addition, the performance analysis shows the influences of the memory hierarchy of the system on a chip. Furthermore, it is shown that the libraries of Analog Devices differ in their application and performance. The case study also illustrates that, depending on the application, an open source implementation not designed for DSPs is competitive to the optimized libraries of Analog Devices and the hardware accelerator. Moreover, the analysis determines the additional effort that must be invested for the configuration of the hardware accelerator. It is shown that the use of the hardware accelerator is only profitable when a certain number of samples is used. Finally, the two case studies are linked to a proof of concept that illustrates the possibilities of heterogeneous computing.

Inhaltsverzeichnis

1 Einleitung

Der Trend zu Mehrkernprozessoren und hohen Taktraten hat längst die Welt von eingebetteten Systemen erreicht (vgl. Analog Devices Inc. 2017, Kap. 1 S. 1). Durch die Integrierung von unterschiedlichen Prozessoren und Hardware-Beschleunigern in Ein-Chip-Systemen, nimmt die Schwierigkeit von eingebetteten Systemen hinsichtlich deren Programmierung zu. Abgesehen von der steigenden Komplexität ermöglicht der Einsatz von mehreren Prozessoren in Ein-Chip-Systemen die Umsetzung von echter Nebenläufigkeit. Zudem wird durch die Verwendung von spezialisierten Prozessoren die Effizienz hinsichtlich des Energieverbrauchs solcher Systeme verbessert, der speziell im Bereich der eingebetteten Systeme von enormer Bedeutung ist. (vgl. S. L. Harris und D. M. Harris 2016, S. 468 - 470) Diese Arbeit beschäftigt sich mit Mikroprozessor und DSP Ein-Chip-Systemen beispielhaft anhand des ADSP-SC582 aus der Produktreihe ADSP-SC58x von Analog Devices. Dabei werden die heterogenen Eigenschaften des ADSP-SC582 hinsichtlich dessen Programmierung, Fähigkeiten und Limitierungen für die Signalverarbeitung analysiert. Außerdem hat das Ein-Chip-System bis zum jetzigen Zeitpunkt keine nennenswerte wissenschaftliche Betrachtung erhalten, die das System auf dessen Komplexität, Programmierung und Verwendung hinsichtlich des heterogenen Rechnens untersucht.

1.1 Motivation

Angesichts der beschriebenen Eigenschaften des ADSP-SC582 und der fehlenden wissenschaftlichen Betrachtung wird aktuell bei einer in der Entwicklung stehenden Verstärkerplatine der Firma OMICRON lediglich der ARM Prozessor des ADSP-SC582 verwendet, da dieser für die derzeitigen Anliegen ausreichend Rechenleistung bereitstellt. Die Verstärkerplatine ist Bestandteil eines mobilen Prüf- und Diagnosegeräts aus der Energietechnik und wird zur Erzeugung und Messung von elektrischen Signalen genutzt. Das Prüfgerät ist von besonderer Bedeutung, da es ein universelles Prüfgerät für Applikationen im Bereich der Inbetriebnahme, Zustandsbewertung und Wartung von Betriebsmitteln ist. Durch dessen Vielseitigkeit können mit dem Prüfgerät andere Geräte ersetzt werden. Dadurch sind schnellere Testzeiten sowie die Einsparung von Kosten möglich. Momentan wird das Prüfgerät zur Zustandsbewertung von elektrischen Betriebsmitteln mit etablierten Messverfahren verwendet. Derzeit gibt es Pläne,

das Prüfgerät auch für Forschungszwecke zu verwenden. Dabei ist beispielsweise geplant, mit dem Prüfgerät neue Messverfahren zu entwickeln. Vorbereitend dafür soll im Zuge dieser Arbeit der ADSP-SC582 untersucht werden. Dabei wird das System, dessen Komponenten und ihre Interaktion charakterisiert, damit die Leistungsfähigkeit des Systems für Forschungszwecke im Bereich der Signalverarbeitung bewertet werden kann. Gleichermaßen muss die Programmierung und dessen Komplexität untersucht werden, für den Fall, dass der zusätzlich verfügbare Prozessor und die Hardware-Beschleuniger in Zukunft verwendet werden.

Funktionsweise der Verstärkerplatine

Bislang (Stand August 2020) wird bei dem in Abbildung 1.1 gezeigten System nur der ARM Cortex A5 Kern des ADSP-SC582 verwendet. Der ARM Kern gibt dem FPGA über das L3 Memory Interface und den externen Speicher Zielwerte vor. Die Zielwerte werden im FPGA in Echtzeit verarbeitet und zur Steuerung der Leistungselektronik verwendet. Zusätzlich werden durch den FPGA 19 Messkanäle der Leistungselektronik mit maximal 80 kHz abgetastet und im 10 kHz Takt über den externen Speicher bereitgestellt. Dabei wird der ARM Kern mit dem FPGA über einen GPIO-Interrupt synchronisiert, sodass dieser benachrichtigt wird, wenn neue Messergebnisse vorhanden sind. Weitere Aufgaben des ARM Prozessors sind die Kommunikation und Interaktion mit weiteren Mikrocontrollern des Prüfgeräts, die für die Betriebssicherheit oder Eingabegeräte verantwortlich sind.

1.2 Problemstellung und Zielsetzung

Abbildung 1.1 veranschaulicht vereinfacht das heterogene System der Verstärkerplatine, die den ADSP-SC582 verwendet. Obwohl das Blockschaltbild hinsichtlich des ADSP-SC582 bereits auf das Wesentliche reduziert wurde, ist immer noch sehr gut die Komplexität des Ein-Chip-Systems und dessen Komponenten zu erkennen. Zuerst sind die zwei ungleichen Prozessoren zu erwähnen, deren unterschiedliche Cache-Levels und eine verschiedenartige Anbindung an die Verbindungseinrichtung aufweisen. Zweitens sind verschiedene Speicher (L1, L2, System L2 Memory, externer Speicher) und deren ungleiche Anbindung zu erkennen. Des Weiteren können in der Abbildung ein Bruchteil der verfügbaren Peripheriegeräte ausgemacht werden. Darüber hinaus fasst der Hardware-Beschleuniger Block mehrere anwendungsspezifische Prozessoren zusammen. Letztendlich erschweren die heterogenen Komponenten die Programmierung und Verwendung des Systems. Aus dieser Tatsache resultiert eine Problemstellung, für die folgende Forschungsfrage formuliert wird.

“Was muss bei der Programmierung von ARM und DSP Ein-Chip-Systemen hinsichtlich der heterogenen Prozessoren und Peripheriebausteinen beachtet werden?”

Zielsetzung

Im Zuge dieser Arbeit sind die Gründe für heterogenes Rechnen und dessen aktueller Stand der Technik zu erörtern. Anschließend ist das in Abbildung 1.1 gezeigte System auf dessen Heterogenität zu untersuchen. Dabei sind der FPGA und die Leistungselektronik nicht Gegenstand dieser Untersuchung. Eine Betrachtung jeder einzelnen Komponente hinsichtlich deren Heterogenität und Leistungsfähigkeit für Forschungszwecke ist im Rahmen dieser Arbeit schlicht unmöglich. Zum einen, weil keine konkreten technischen Anforderungen an ein zukünftiges Messverfahren bekannt sind und zum anderen, weil es sich um ein Ein-Chip-System handelt, das eine Vielzahl an unterschiedlichen Komponenten aufweist. Vielmehr werden anhand der identifizierten Komponenten Fallbeispiele ausgewählt, sodass mithilfe dieser die bedeutendsten Feinheiten zur Programmierung und Verwendung des Ein-Chip-Systems erörtert werden.

1.3 Methodisches Vorgehen

Aufgrund der Zielsetzung wird zuerst der aktuelle Stand der Technik erarbeitet, der anschließend zur Analyse eines existierenden Systems und dessen heterogene Eigenschaften verwendet wird. Infolgedessen wird als Fallbeispiel die Umsetzung der schnellen Fourier Transformation (engl. Fast Fourier Transformation (FFT)) mithilfe des ADSP-SC582 untersucht. Zum einen wird dies aufgrund der Tatsache untersucht, dass die Berechnung einer FFT sehr rechenintensiv und für viele Bereiche in den Ingenieurwissenschaften von großer Relevanz ist. Beispielsweise findet die FFT Anwendung in Bereichen der Signalverarbeitung, Signalanalyse oder auch bei Kompressionsalgorithmen. Zum anderen bietet Analog Devices unterschiedliche und teilweise optimierte Bibliotheken zur Berechnung der FFT auf dem SHARC+ Kern des ADSP-SC582 und einen eigenen FFT Hardware-Beschleuniger. Weiterhin können durch das Ablegen der FFT Daten in unterschiedlichen Speichern die Einflüsse dieser auf die Speicherzugriffszeiten sowie deren Auswirkung auf die Performanz der unterschiedlichen Implementierungen von Analog Devices und dessen Hardware-Beschleuniger untersucht werden. Als weiteres Fallbeispiel wird von den Peripheriegeräten die Universal Asynchronous Receiver Transmitter (UART)-Peripherie gewählt, dabei wird deren Verwendung untersucht. Überdies wird analysiert, was beim Senden sowie Empfangen von serialisierten Daten zu berücksichtigen ist. Durch die Untersuchung des ARM Cortex A5, dem SHARC+ Kern, des FFT Hardware-Beschleunigers, des internen und externen Speichers sowie der UART Peripherie

werden somit alle Anbindungen an die Verbindungseinrichtung und die unterschiedlichen Interaktionen der Prozessoren betrachtet (siehe Abbildung 1.1). Infolgedessen werden die Problemstellungen und Merkmale der Programmierung von ARM und DSP Ein-Chip-Systemen identifiziert. Durch die Verknüpfung der beiden Fallbeispiele kann abschließend ein Konzeptnachweis erbracht werden, der ein mögliches Anwendungsbeispiel für die bestehende Verstärkerplatine und den ADSP-SC582 beschreibt.

Resultierend aus der beschriebenen Methodik werden quantitative sowie qualitative Ergebnisse bereitgestellt. Bei der Analyse der Speicherzugriffszeiten und Performanz werden quantitative Daten erhoben. Im Unterschied dazu findet durch die Betrachtung der heterogenen Merkmale des Systems und dessen Komponenten eine qualitative Betrachtung statt.

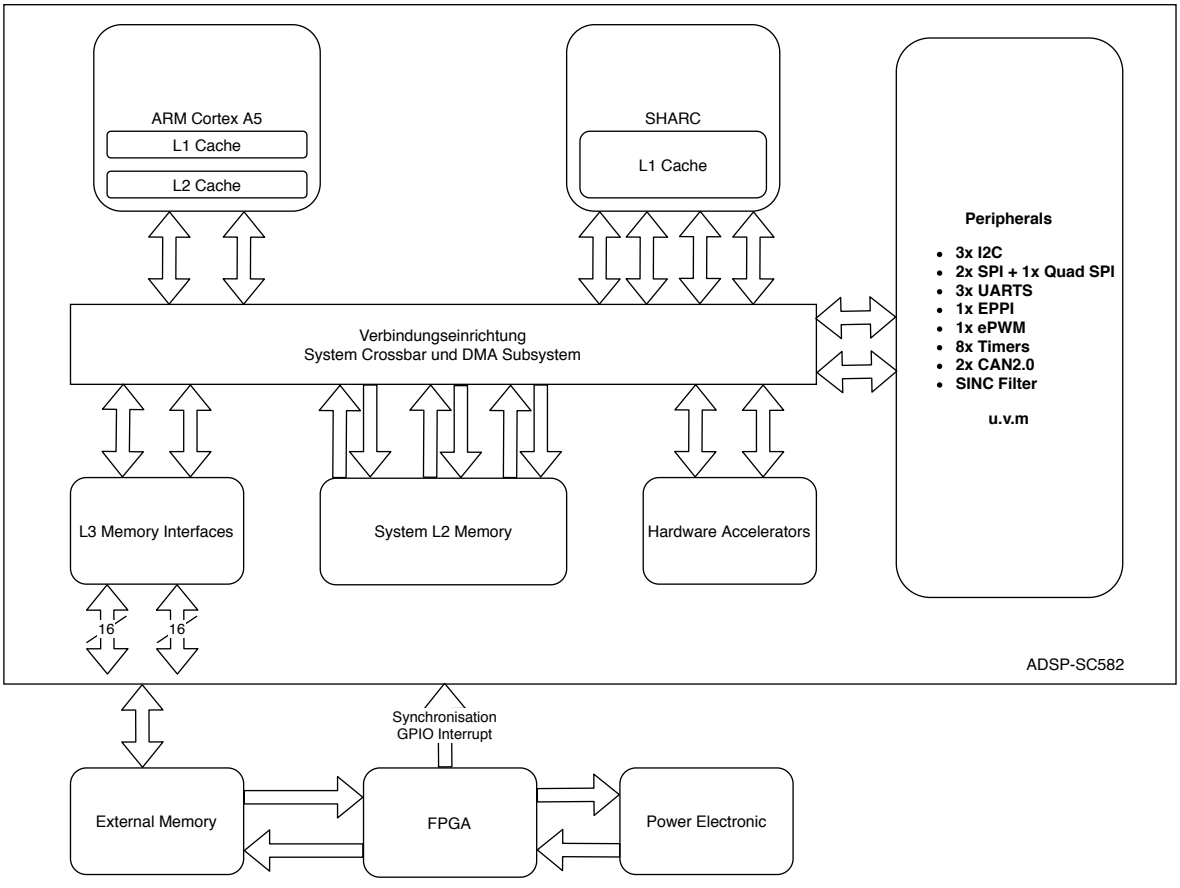


Abbildung 1.1: Heterogenes System mit dem ADSP-SC582
(In Anlehnung an Analog Devices Inc. 2017, Kap. 1 S. 1)

1.4 Nutzen der wissenschaftlichen Betrachtung

Bei der wissenschaftlichen Betrachtung werden unterschiedliche Komponenten und deren Abhängigkeiten eines Multiprozessor-Ein-Chip-Systems mit ARM und DSP Kern analysiert, sodass eine effiziente Signalverarbeitung möglich ist. Außerdem wird der aktuelle Stand der Technik erörtert und eine Literaturrecherche betrieben, sodass die Verknüpfung der Erkenntnisse in einem Domänenwissen resultiert. Durch die Analyse eines existierenden Systems und dessen Komponenten werden die praktischen Aspekte der Programmierung solch eines Systems sowie dessen Potential und Limitierungen zur Signalverarbeitung aufgezeigt. Aufgrund der Zunahme von heterogenen Prozessoren in eingebetteten Systemen bietet die Beantwortung der Forschungsfrage nützliches Wissen für Entwickelnde im Bereich der Softwareentwicklung von eingebetteten Systemen. Überdies werden durch die Beantwortung der Forschungsfrage beispielhaft anhand des ADSP-SC582 detaillierte Erkenntnisse erfasst, die überwiegend für die gesamte Produktreihe des ADSP-SC58x von Analog Devices und somit für einen Großteil der modernen DSPs gültig sind. Durch die wissenschaftliche Betrachtung werden neue Erkenntnisse hinsichtlich der Programmierung der ADSP-SC58x Produktreihe erarbeitet, die derzeit noch keine nennenswerte wissenschaftliche Betrachtung erfahren hat.

2 Heterogenes Rechnen

Werden in einem System unterschiedliche Prozessoren zum Lösen von Aufgaben verwendet, so kann dies mit dem Ausdruck heterogenes Rechnen bezeichnet werden (vgl. Zahran 2019, S. xiii). Dabei ist es unerheblich, ob es sich bei dem System um ein einzelnes oder ein vernetztes System handelt. Heterogenes Rechnen ist eine aufstrebende domänenübergreifende Disziplin in der Informatik, die sich aufgrund der unterschiedlichen Technologien und Anwendungsbereichen ergibt. Beispielsweise werden in eingebetteten Systemen zunehmend mehrere unterschiedliche Mikrocontroller oder Ein-Chip-Systeme mit mehreren Kernen eingesetzt. Aber auch im Bereich des modernen Cloud-Computing wird heterogenes Rechnen angewendet, indem beispielsweise zum einen herkömmliche CPUs und zum anderen GPUs zur Anwendung kommen. (vgl. Terzo 2019, S. 3) Das Handhaben der unterschiedlichen Prozessoren und deren Spezialisierung gegenüber homogenem Rechnen birgt allerdings zusätzliche Hürden für die Softwareentwicklung (vgl. Zahran 2019, S. xiii). Abbildung 2.1 veranschaulicht ein generisches heterogenes System.

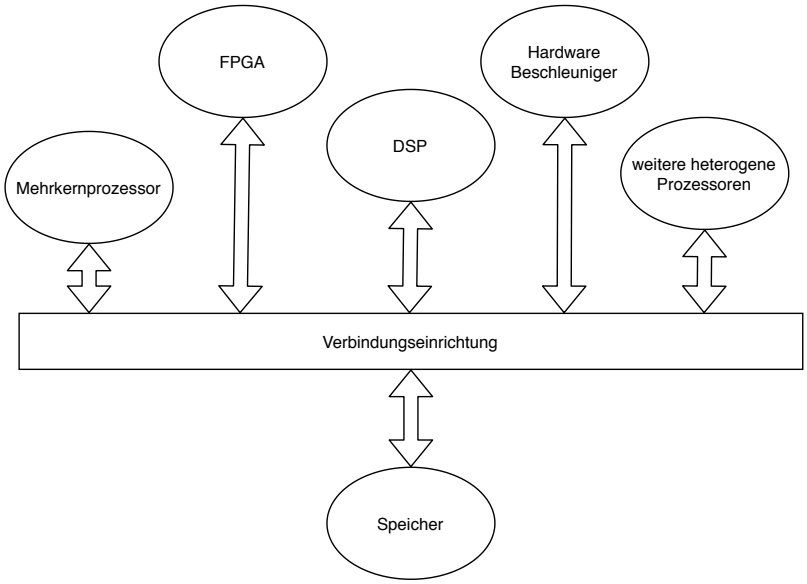


Abbildung 2.1: Generisches heterogenes System
(vgl. Zahran 2017)

In den folgenden Abschnitten werden die Gründe für heterogenes Rechnen mit Fokus auf Ein-Chip-Systeme von eingebetteten Systemen erläutert und die dabei entstehenden Herausforderungen dargelegt.

2.1 Gründe für Heterogenes Rechnen

Heterogenes Rechnen resultiert aus unterschiedlichen Problemstellungen und Anforderungen an die Applikation. Der wohl wesentlichste Grund des Trends zur Heterogenität ist, dass durch Erhöhen der Anzahl der Transistoren eines Prozessors auch die Leistungsaufnahme zunimmt (siehe folgender Unterunterabschnitt). Dies führt dazu, dass General Purpose Processors (GPP) mit ihrer durchschnittlichen Performanz zu viel elektrische Leistung für spezielle Problemstellungen benötigen (S. L. Harris und D. M. Harris 2016, S. 469). Aber auch die reine Betrachtung der Rechenleistung ist ein Grund für heterogenes Rechnen, die durch spezialisierte Prozessoren weiterhin verbessert wird.

Leistungsdichte von Prozessoren

Im Folgenden werden die erprobten Faustregeln das mooresche Gesetz und Dennard Scaling erläutert. Dabei wird der Zusammenhang der beiden Regeln und deren Einfluss auf die Leistungsdichte von Prozessoren dargelegt.

Mooresches Gesetz

Die ständige Verdoppelung der Anzahl an Transistoren eines Chips alle 2 Jahre, wurde von Gordon Moore im Jahre 1965 vorhergesagt (vgl. Gordon Moore 2005). Die Vorhersage ist als mooresches Gesetz bekannt und kann als Faustregel interpretiert werden. Dabei wurde diese seit deren Vorhersage zumindest nachweislich bis zum Jahre 2018 annähernd erfüllt (vgl. Max Roser 2019). Zum einen konnten durch das Erhöhen der Transistordichte zusätzliche Logikbausteine integriert werden. Zum anderen erlaubt die Miniaturisierung der Transistoren das Verwenden höherer Taktfrequenzen. Durch die Erhöhung der Anzahl an Transistoren und deren Taktfrequenz konnten die letzten Jahre immer schnellere Prozessoren hervorgebracht werden. Dabei galt es als selbstverständlich, dass ein geschriebenes Programm bei der nächsten Prozessorgeneration schneller ausgeführt werden kann (vgl. Zahran 2019, S. 2). Doch das mooresche Gesetz stößt an seine Grenzen und es ist bereits zu erkennen, dass dies in Zukunft nicht mehr erfüllt wird. Dies wird nicht zwingend aufgrund physikalischer Gegebenheiten entkräftet, sondern durch wirtschaftliche Beweggründe wie beispielsweise die hohen Investitionskosten bei der Entwicklung von neuen Prozessoren, die lediglich einen moderate Steigerung der Performanz versprechen. (vgl. Merritt 2013). Aber auch die maximale Verlustleistung eines Prozessors ist eine Limitierung, welche die Kosten hinsichtlich des Wärmemanagements erhöht.

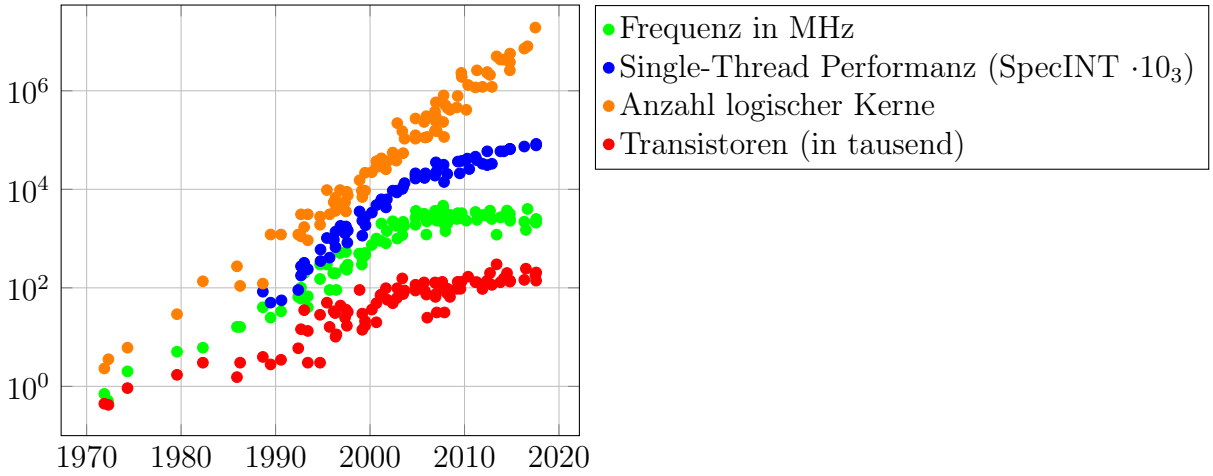


Abbildung 2.2: Trend von unterschiedlichen Mikroprozessoren (Rupp 2020)

Dennard Scaling

Im Zusammenhang mit dem mooreschen Gesetz steht das Dennard Scaling. Das Dennard Scaling aus dem Jahr 1974 besagt, dass bei der Miniaturisierung von Transistoren diese schneller werden und ihre Leistungsdichte konstant bleibt. (vgl. Dennard u. a. 1999) Durch das Erhöhen der Taktfrequenz und Anzahl der Transistoren von integrierten Schaltkreisen konnten somit immer schnellere Prozessoren erstellt werden. Allerdings wurden parasitäre Effekte wie Kriechströme und auch die benötigte Schwellspannung, bei der ein Transistor schaltet, für das Dennard Scaling nicht berücksichtigt. Werden parasitäre Effekte berücksichtigt, so stellt sich heraus, dass die Leistungsdichte nicht konstant ist. Durch die Erhöhung der Anzahl an Transistoren und der Taktfrequenz steigt daher auch die benötigte Leistung. Um das Jahr 2004 wurden erste Prozessoren gefertigt, die eine so große Transistordichte hatten, dass parasitäre Effekte nicht mehr vernachlässigbar waren. Daher gilt das Dennard Scaling seit ungefähr 2004 als entkräftet. (vgl. Zahran 2019, S. 2 und Hennessy 2019, S. 58) Die parasitären Effekte tragen nun einen wesentlichen Teil zur Leistungsaufnahme eines Prozessors bei. Durch Erhöhen der Anzahl an Transistoren erhöht sich nun auch die Leistungsdichte. Da die Leistungsdichte nicht mehr konstant ist müssen nun Aspekte wie die Bauform, Prozessor-Architektur und das Wärmemanagement adaptiert werden. Ansonsten wird die Verlustleistung der Prozessoren im Bezug auf die Hitzeabfuhr untragbar. Exemplarisch dafür veranschaulicht Abbildung 2.2 die exponentielle Zunahme der Anzahl an Transistoren in der schwarzen Datenreihe. Dabei ist in der roten Datenreihe die typische Leistung eines Prozessors zuerkennen. Vielmehr ist ab dem Jahr 2004 ersichtlich, dass

die aufgenommene Leistung der Prozessoren nur minimal zunimmt. Dasselbe gilt für die Taktfrequenz der Prozessoren in der grünen Datenreihe. Dies lässt sich Aufgrund des Dennard Scaling erklären, da nun die parasitären Effekte zu tragen kommen, müssen der Einfluss der Taktfrequenz und die statischen Verluste berücksichtigt werden (siehe Abschnitt 2.4.1). Das Ende des Dennard Scaling und dessen Auswirkungen sind für den Trend zu heterogenen Systemen verantwortlich.

Das Ende des Dennard Scalings und vermutliche Ende des mooreschen Gesetzes

Das Ende des Dennard Scaling und vermutliche Ende des mooreschen Gesetz führt dazu, dass herstellende Unternehmen nicht nur die Miniaturisierung vorantreiben, sondern auch die Leistungsaufnahme der Prozessoren verringern. Dies wird erreicht, indem die ständige Erhöhung der Taktfrequenz der Prozessoren gestoppt wird. Damit trotz dieses Sachverhalts höhere Rechenleistungen erzielt werden, wird die Anzahl an Prozessorkernen erhöht. Dadurch werden herstellende Unternehmen mit neuen Herausforderungen und Forschungsfragen konfrontiert, da größere Dies fehleranfälliger sind. (vgl. Terzo 2019, S. 296) Durch den Multi-Kern-Trend wird die Steigerung der Rechenleistung nicht mehr durch technologische Fortschritte im Sinne von schnelleren Prozessoren erzielt, daher müssen Programme parallelisiert werden, um eine Leistungssteigerung zu erzielen. Auch das nahende Ende des mooreschen Gesetzes führt in weiterer Folge dazu, dass die Anzahl an Kernen in einem Chip begrenzt ist. Leistungssteigerungen werden daher in naher Zukunft vermehrt durch spezialisierte Prozessoren erzielt, die für spezifische Anwendungsgebiete entwickelt werden. (vgl. Terzo 2019, S. xi)

2.2 Arten von Multiprozessorsystemen

Multiprozessorsysteme bestehen aus mehreren Prozessoren, die miteinander verbunden sind und eine gegenseitige Kommunikation ermöglichen (vgl. S. L. Harris und D. M. Harris 2016, S. 468). Abbildung 2.3 veranschaulicht einen Aufbau eines eng gekoppelten Multiprozessorsystems. Multiprozessorsysteme werden hinsichtlich der einzelnen Prozessoren in symmetrische oder asymmetrische Multiprozessorsysteme eingeteilt (siehe Unterabschnitt 2.2.1 und 2.2.2). Aber auch eine Einteilung durch die verwendete Speicherstruktur ist zulässig, dabei wird zwischen eng und lose gekoppelten Systemen unterschieden (siehe Abschnitt 2.3).

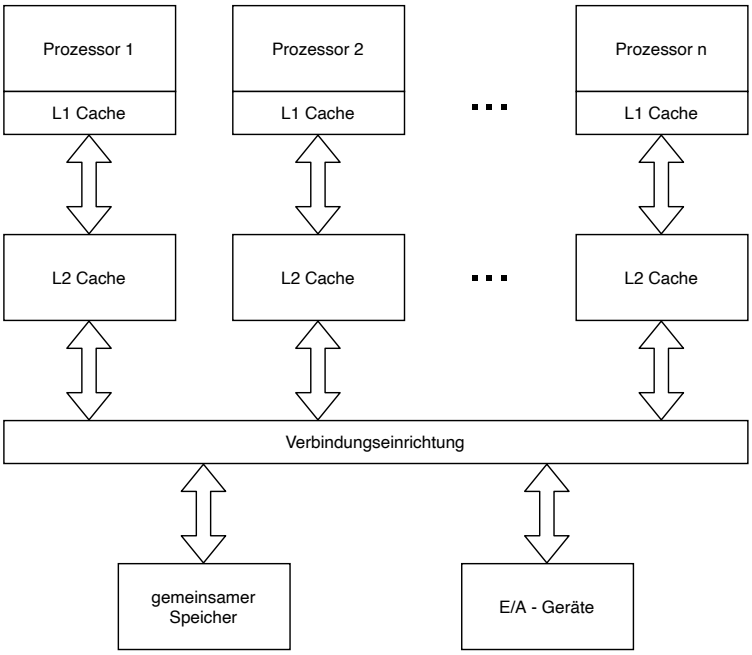


Abbildung 2.3: Eng gekoppeltes Multiprozessorsystem
(vgl. Bengel u. a. 2015, S. 39)

2.2.1 Symmetrische Multiprozessorsysteme

Symmetrische Multiprozessorsysteme (alternative Bezeichnung: homogene Multiprozessorsysteme) bestehen aus zwei oder mehreren Prozessoren desselben Prozessortyps (vgl. Bengel u. a. 2015, S. 51 und S. L. Harris und D. M. Harris 2016, S. 468). Homogene Multiprozessorsysteme ermöglichen es, mehrere Threads gleichzeitig oder einen einzelnen Thread schneller auszuführen. Dabei ist die Beschleunigung eines einzelnen Threads durch das Ausführen auf mehreren Kernen eine Herausforderung, weil der Thread bei der Entwicklung in mehrere Threads aufgeteilt werden muss. Dabei muss beispielsweise die Kommunikation zwischen den einzelnen Kernen sichergestellt werden.
Ein wesentlicher Vorteil von symmetrischen Multiprozessorsysteme ist, dass ein für den Prozessortyp angepasstes Programm auf jedem Kern mit nahezu gleicher Performanz ausgeführt werden kann. Auch das Design eines symmetrischen Multiprozessorsystems ist relativ unkompliziert. Es muss einmal ein Prozessor konstruiert werden und das System kann mit diesem Prozessor nahezu beliebig oft erweitert werden (vgl. S. L. Harris und D. M. Harris 2016, S. 468).
Allerdings wird durch das Hinzufügen von zusätzlichen symmetrischen Prozessoren eine Steigerung der Performanz nicht sichergestellt, weil dazu eine Parallelisierung der Programme durchgeführt werden muss. Die Parallelisierung

kann jedoch nicht beliebig oft durchgeführt werden, da ein Programm nicht beliebig oft in parallelisierbare Einzelteile zerlegbar ist. Des Weiteren sind GPP konstruiert, um mit durchschnittlicher Leistung unterschiedliche Probleme zu lösen. Sie sind kaum für Problemstellungen spezialisiert und können diese somit nicht energieeffizient bearbeiten. Besonders bei eingebetteten Systemen ist die Energieeffizienz ein erheblicher Faktor. (vgl. S. L. Harris und D. M. Harris 2016, S. 469) Zudem muss beachtet werden, dass homogene Multiprozessorsysteme auch heterogene Eigenschaften aufweisen (siehe 2.4.1).

2.2.2 Asymmetrische Multiprozessorsysteme

Asymmetrische Multiprozessorsysteme beinhalten unterschiedliche Prozessoren oder auch Hardware-Beschleuniger, die eine Spezialisierung für bestimmte Operationen und Anwendungen ermöglichen (vgl. S. L. Harris und D. M. Harris 2016, S. 469). Speziell bei eingebetteten Systemen werden oft Spezialisierungen benötigt. Beispielsweise werden bei Audio- und Video-Systemen oft Digitale Signalprozessoren verwendet, um Audio- und Videosignale performant zu verarbeiten (vgl. Bengel u. a. 2015, S. 52 und S. L. Harris und D. M. Harris 2016, S. 469). Aber auch Hardware-Beschleuniger werden bei modernen asymmetrischen Multiprozessorsystemen verwendet, die speziell für eine bestimmte Aufgabe ausgelegt sind. Beispielsweise enthalten moderne Mobiltelefone spezielle Beschleuniger für kryptographische Anwendungen oder auch Grafikprozessoren (vgl. S. L. Harris und D. M. Harris 2016, S. 469). Asymmetrische Multiprozessorsysteme bieten somit unterschiedliche Möglichkeiten, um ein Problem zu lösen. Allerdings wird die Komplexität des Designs und der Programmierung wesentlich erhöht. Nun müssen die unterschiedlichen Eigenheiten der Prozessoren berücksichtigt werden. (vgl. S. L. Harris und D. M. Harris 2016, S. 469 und Bengel u. a. 2015, S. 51 - 52) Beispielsweise kann nicht ohne Weiteres ein Programm, das für einen bestimmten Prozessortyp geschrieben ist, auf einem anderen Prozessortyp ausgeführt werden (vgl. Bengel u. a. 2015, S. 51 - 52).

2.3 Arten der Speicheranbindung von Multiprozessorsystemen

Eine weitere Kategorisierung von heterogenen Systemen kann anhand der Anbindung des Speichers vorgenommen werden. Dabei muss zwischen eng gekoppelten und lose gekoppelten Multiprozessorsystemen unterscheiden werden. Eng gekoppelte Multiprozessorsysteme verwenden einen gemeinsamen Hauptspeicher. Aufgaben können über den gemeinsamen Speicher koordiniert und gesteuert werden (vgl. Bengel u. a. 2015, S. 33). Abbildung 2.1 auf Seite 12 veranschaulicht ein eng gekoppeltes heterogenes System. Im Gegensatz zu eng gekoppelten Multiprozessorsystemen verfügen lose gekoppelte Multiprozessorsysteme

über keinen gemeinsamen Speicher. Ein System von lose gekoppelten Multiprozessoren kann lediglich mittels Nachrichtenaustausch über eine Verbindungseinrichtung koordiniert werden. Lose gekoppelte Multiprozessoren fungieren als selbstsändige Recheneinheiten (vgl. Bengel u. a. 2015, S. 34). Abbildung 2.4 zeigt ein lose gekoppeltes Multiprozessorsystem.

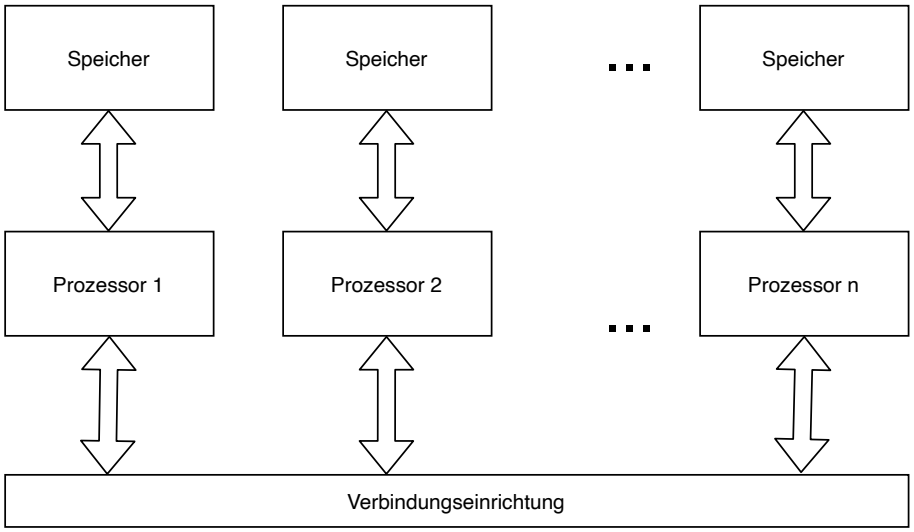


Abbildung 2.4: Lose gekoppeltes Multiprozessorsystem
(Eigene Ausarbeitung in Anlehnung an Bengel u. a. 2015, S. 81)

2.4 Herausforderungen von heterogenem Rechnen

Im Folgenden werden die unterschiedlichen Herausforderungen von heterogenem Rechnen beschrieben, die bei der Verwendung von Multiprozessorystemen auftreten. Dabei gilt es zwischen der kontrollierbaren sowie unkontrollierbaren Heterogenität zu unterscheiden.

2.4.1 Unkontrollierbare Heterogenität

Bei der unkontrollierbaren Heterogenität handelt es sich um die unterschiedlichen Eigenschaften der Komponenten eines Multiprozessorsystems, die das Verhalten des Systems beeinflussen, aber nicht kontrollierbar sind. Die unkontrollierbare Heterogenität muss bei der Entwicklung berücksichtigt werden und kann nicht vermieden werden. Allerdings kann diese indirekt beeinflusst werden. Beispielsweise kann dies durch die Auswahl eines bestimmten Prozessors oder auch durch die Auslastung des jeweiligen Kerns gesteuert werden (siehe 2.4.1).

Technologien

Zur Herstellung von integrierten Schaltkreisen werden hauptsächlich Complementary metal-oxide-semiconductor (CMOS) Halbleiterbauelmente verwendet. Allerdings hat sich die Bauweise von modernen CMOS Feldeffektransistoren (FET) in den letzten Jahren verändert (vgl. Zahran 2019, S. 5 und Terzo 2019, S. 6). Moderne FETs werden nicht mehr traditionell in planar Bauform erstellt, sie werden vermehrt mit speziellen 3D Strukturen gefertigt. Eine Variante dabei ist die FinFET Technologie, die aufgrund der besonderen Bauweise verbesserte Eigenschaften hinsichtlich der Schaltzeiten des Transistors hat. Es werden auch andere Maßnahmen verwendet wie das Verringern der Kapazitäten durch die Silicon-On-Insulator (SOI) Technologie, die mittels spezieller Isolierschicht das Silizium vom Siliziumsubstrat isoliert (vgl. Zahran 2019, S. 5 und Terzo 2019, S. 6). Aufgrund der unterschiedlichen Schaltzeiten und Kapazitäten der verschiedenen Schaltkreise fällt es schwer die Performanz eines Ein-Chip-Systems abzuschätzen, da die technologischen Details meist nicht bekannt sind. Dies führt dazu, dass in Zukunft auch die unterschiedlichen Technologien bei der Entwicklung zu berücksichtigen sind, sofern deren Eigenschaften bekannt sind. Damit die Performanz mehrerer Prozessoren in einem System bewertet werden kann (vgl. Zahran 2019, S. 5).

Energieverwaltung

Ein wesentlicher Aspekt der unkontrollierbaren Heterogenität ist die teilweise unbeeinflussbare Energieverwaltung von Prozessoren.

Dynamic Voltage and Frequency Scaling

Ein Verfahren zur Reduzierung der Verlustleistung von Prozessoren nennt sich *Dynamic Voltage And Frequency Scaling (DVFS)*. Das Verfahren fungiert meist bei jedem einzelnen Kern eigenständig und bereits auf der Hardware-Ebene, aber auch Betriebssysteme nehmen Einfluss auf das DVFS. Dabei wird bei niedriger Auslastung die Versorgungsspannung und/oder Taktfrequenz reduziert. Ziel dabei ist es, die dynamischen sowie statischen Verluste der einzelnen Transistoren eines Prozessors zu reduzieren (vgl. Zahran 2019, S. 5 - 6 und ARM Limited 2011, Kap. 20 S. 7). Die Verlustleistung eines Transistors wird durch die statischen und dynamischen Verluste bestimmt. Die dynamischen Verluste werden durch das Berechnen der Schaltverluste angenähert. Die Gleichung 3.1 veranschaulicht den quadratischen Einfluss der Versorgungsspannung sowie den linearen Einfluss der Taktfrequenz auf die Verlustleistung (vgl. Zahran 2019, S. 5). Dabei sind die Versorgungsspannung und die Frequenz die einzigen Parameter, die eine Beeinflussung nach der Herstellung zulassen. Die Parameter C und N sind

durch das Design des Prozessors vorgegeben.

$$P_{dyn} = C \cdot V_{CC}^2 \cdot f \cdot N \tag{2.1}$$

Dabei gilt: P_{dyn} ... dynamische Verlustleistung
C ... Gate Kapazität des Transistors
V_{CC} ... Versorgungsspannung
f ... Taktfrequenz
N ... Anzahl zu schaltender Bits (Zahran 2019, S. 5)

Die statischen Verluste werden bei der Herstellung und durch die Wahl der Betriebsspannung beeinflusst, sie waren aber bisher lediglich für einen Bruchteil der gesamten Verlustleistung verantwortlich. Durch das Erhöhen der Transistordichte nimmt auch die statische Verlustleistung anteilsmäßig zu, die durch Kriechströme verursacht wird. Dies führt dazu, dass bei Logikbausteinen bereits ein großer Anteil der Verlustleistung während des Leerlaufs dissipiert wird (vgl. Hennessy 2019, S. 28).

$$P_{stat} = I_{stat} \cdot V_{cc} \tag{2.2}$$

Dabei gilt: P_{stat} ... statische Verlustleistung
I_{stat} ... statische Stromaufnahme (Kriechströme)
V_{cc} ... Versorgungsspannung (Hennessy 2019, S. 28)

Unter Berücksichtigung des DVFS muss bedacht werden, dass bei der Verwendung von symmetrischen Multiprozessorsystemen, auch diese sich aufgrund der unterschiedlichen Auslastung der Kerne heterogen verhalten, weil sie aufgrund des DVFS bei unterschiedlichen Arbeitspunkten betrieben werden (vgl. Zahran 2019, S. 5). Aber auch die statischen und dynamischen Verluste eines einzelnen Kerns sind aufgrund der unterschiedlichen Auslastung anders und tragen somit zur Heterogenität bei.

Dark Silicon Durch die ständige Erhöhung der Transistordichte und der Entkräftung des Dennard Scalings werden nun Prozessoren entwickelt, bei denen der gleichzeitige Betrieb aller Transistoren nicht möglich ist, da ansonsten die maximal mögliche Verlustleistung überschritten wird (vgl. Hennessy 2019, S. 28). Dieser Sachverhalt führt dazu, dass bestimmte Bereiche eines Prozessors nicht gleichzeitig verwendet werden können. Die ungenützten Bereiche werden Dark Silicon genannt (vgl. Hennessy 2019, S. 28 und Zahran 2019, S. 3). Die unterschiedlichen Bereiche eines Prozessors werden beispielsweise durch Clock Gating aktiviert, deaktiviert oder deren Leistungsaufnahme beeinflusst (vgl.

Hübner und Becker 2011, S. 233). Durch die jeweilige Auswahl der verwendeten Funktion bzw. Bereiche wird somit auch die Heterogenität beeinflusst.

Speichersystem

Bei Multiprozessorsystemen müssen die unterschiedlichen Zugriffszeiten der einzelnen Kerne auf die jeweiligen Speicher im Gegensatz zu Ein-Prozessor-Systemen zusätzlich berücksichtigt werden (vgl. Zahran 2019, S. 6). Dabei muss zum einen unterschieden werden, ob es sich um einen Uniform-Memory-Access (UMA) oder um einen Non-Uniform-Memory-Access (NUMA) handelt. Zum anderen bedarf es auch der Berücksichtigung der Cache Zugriffe, die auch mit gleicher oder unterschiedlicher Geschwindigkeit erfolgen. In den folgenden Abschnitten wird die jeweilige Art der Speicherzugriffe beschrieben.

Uniform-Memory-Access Wenn die jeweiligen Prozessoren eines eng gekoppelten Multiprozessorsystems auf den gemeinsamen Speicher mit gleicher Geschwindigkeit zugreifen, dann handelt es sich um eine UMA-Architektur. Durch die Gleichheit der Speicherzugriffe dieses Systems kann seine Performanz vorhergesagt werden (vgl. Bengel u. a. 2015, S. 80).

Non-Uniform Memory Access Bei NUMA-Systemen kann ein Prozessor auf den eigenen lokalen Speicher schneller zugreifen, als auf den gemeinsam geteilten Hauptspeicher. Die Geschwindigkeit eines Speicherzugriffs hängt somit von dessen Lokalität ab (vgl. Bengel u. a. 2015, S. 81).

Non-Uniform Cache Access Eine Präzisierung des Begriffs NUMA ist der Begriff Non-Uniform-Cache-Access (NUCA). NUCA ist gegeben, wenn Prozessoren einen geteilten Cache haben und der jeweilige Prozessor auf seinen eigenen Cache schneller Zugreifen kann als die anderen Kerne des Systems. Abbildung 2.5 zeigt ein Beispiel anhand eines Ein-Chip-Systems, bei dem die Prozessoren auf den eigenen geteilten Cache schneller zugreifen, als auf den der anderen Prozessoren, da keine Kommunikation über die Verbindungseinrichtung benötigt wird (vgl. Hennessy 2019, S. 391).

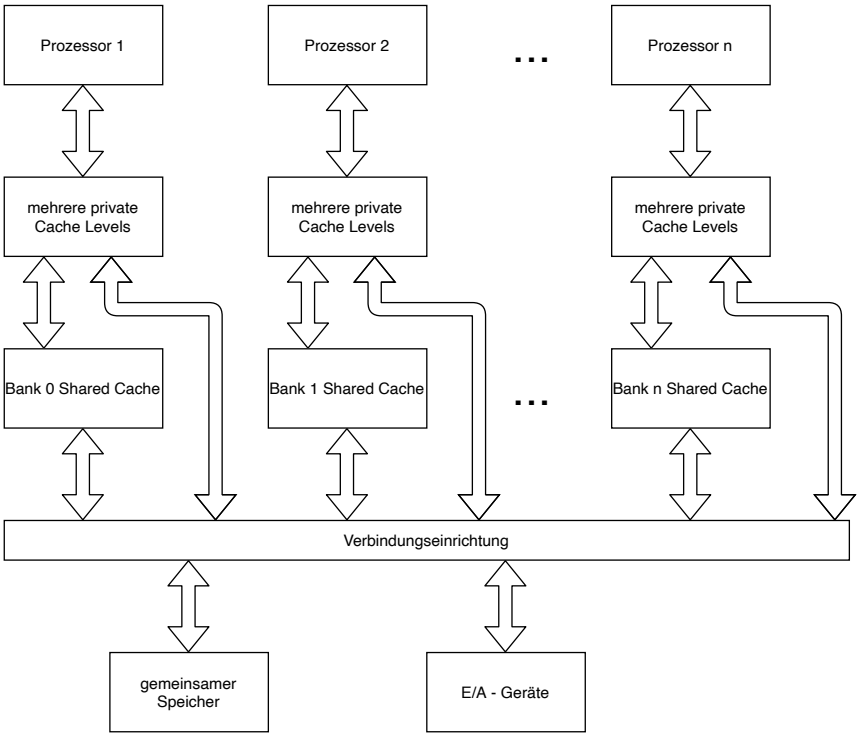


Abbildung 2.5: Ein-Chip-Multicore mit verteiltem Cache und NUCA (Hennessy 2019, S. 391)

Cache-Hits und Misses Ein Cache-Speicher kann nur einen Bruchteil des Hauptspeicher bzw. der Speicherhierachie zwischenspeichern. Damit eine gute Performanz gewährleistet wird, werden die Lokalitätseigenschaften der jeweiligen Programme berücksichtigt. Anhand von diesen wird vorhergesagt und entschieden, welche Bereiche der Speicherhierachie im jeweiligen Cache gepuffert werden. Werden nun Daten benötigt, die nicht im Cache gepuffert sind, kommt es zu einem Cache-Miss und die Daten müssen über die langsameren Speicher geladen werden (vgl. S. L. Harris und D. M. Harris 2016, S. 492 - 493). Ein Cache-Miss verlangsamt somit die gesamte Performanz des Prozessors. Durch die Parallelisierung von Prozessen in einem Multiprozessorsystem und dem daraus resultierenden Nichtdeterminismus kann die Reaktionszeit des Systems nicht vorhergesagt werden. Dabei handelt es sich um eine zeitliche Heterogenität, da es aufgrund der Speicherhierachie und Cache-Hits und Misses zu unterschiedlichen Zugriffs- und Reaktionszeiten kommt (vgl. Zahran 2019, S. 6).

Technologie Wie auch die unterschiedlichen Technologien bei Prozessoren selbst, gibt es auch verschiedene Technologien für die Caches und Hauptspeicher. Dabei sind die gängigen Speicherarten für den Hauptspeicher DRAM und für den Cache SRAM. DRAM hat eine größere Speicherdichte als SRAM, aber auch größere Latenzzeiten. Durch die Zunahme der Kerne in einem System steigt auch der Bedarf an mehr Arbeitsspeicher und Cache. Fortschritte in der Speichertechnologie haben Nachfolger für SRAM und DRAM hervorgebracht, welche die vorteilhaften Eigenschaften wie schnelle Zugriffszeit und hohe Speicherdichte vereinen. Die neuartigen Speichertechnologien werden mit dem Überbegriff Non-Volatile-Memory (NVM) bezeichnet. Durch die unterschiedlichen Arten von Speichertechnologien gibt es auch hier eine Heterogenität, die berücksichtigt werden muss (vgl. Zahran 2019, S. 7).

Byte-Reihenfolge Auch hinsichtlich der Byte-Reihenfolge (engl. Endianess) unterscheiden sich Prozessoren. Für gewöhnlich ist die Endianess eines Prozessors und dessen Befehle nicht änderbar. Allerdings gibt es Prozessoren die dies für den Befehlssatz und/oder Speicherzugriffe ermöglichen. Beispielsweise kann der Prozessor ARM-Cortex A5 Wörter in Little- oder Big-Endian im Speicher ablegen, jedoch werden Befehle des Befehlssatzes immer als Litte-Endian ausgewertet (vgl. ARM Limited 2009, Kap. 3 S. 8). Auch der SHARC+ Prozessor von Analog Devices unterstützt für Operationen über die Serielleschnittstelle Little- und Big-Endian, dabei verwendet der Prozessor selbst allerdings Big-Endian (vgl. Analog Devices Inc. 2019d, Kap. 34 S. 5). Die Byte-Reihenfolge kann somit als unkontrollierbare aber auch kontrollierbare Heterogenität eingestuft werden.

2.4.2 Kontrollierbare Heterogenität

Im Folgenden wird erläutert, wie die Heterogenität eines Multiprozessorsystems beeinflusst werden kann.

Wahl des Prozessors

Durch die unterschiedlichen Prozessoren eines heterogenen Systems wird es ermöglicht, die jeweiligen Prozessoren für bestimmte Aufgaben zu verwenden. Dabei handelt es sich bei der Wahl des Prozessors um eine kontrollierbare Heterogenität, die durch Entwickelnde beeinflusst werden kann. Beispielsweise kann entschieden werden, ob eine FFT auf einem GPP oder einem verfügbaren FFT Hardware-Beschleuniger durchgeführt wird. Möglicherweise wird das Verschieben der Daten zum Hardware-Beschleuniger einen zu großen Mehraufwand benötigen, sodass es zweckmäßiger ist, die Berechnung beim GPP durchzuführen. (vgl. Zahran 2019, S. 10)

Algorithmus

Das vorhergehende FFT Beispiel suggeriert, dass bei paralleler Datenverarbeitung nicht zwingend die Berechnung die zeitaufwändigste Operation ist. Die Kommunikation zwischen den einzelnen Rechenknoten, Konfiguration des Prozessors sowie der Speicherzugriff beanspruchen möglicherweise deutlich mehr Zeit. Wenn die Kommunikation, Konfiguration und Speicherzugriffe den größten Teil des Aufwandes ausmachen, so ist es oft besser, einen weniger geeigneten Prozessor beziehungsweise einen langsameren Algorithmus hinsichtlich der Berechnung zu wählen, wenn diese schnellere Speicherzugriffszeiten aufweisen (vgl. Zahran 2019, S. 9).

Mehrkernprozessor

Wenn das heterogene System einen Mehrkernprozessor inkludiert, so besteht möglicherweise die Option, bestimmte Threads spezifischen Kernen zuzuweisen. Dies ist eine weitere Kontrollmöglichkeit, die möglicherweise durch das Betriebssystem aufgehoben wird, um großen Mehraufwand hinsichtlich der Aufteilung der Threads oder eine zu hohe Auslastung zu verhindern (vgl. Zahran 2019, S. 11).

Zahlenformate

Bei der Programmierung von heterogenen Systemen muss die Wahl der Datenformate hinsichtlich Performanz und Portabilität berücksichtigt werden. Beispielsweise werden DSPs mit Gleitkomma- oder Festkommarithmetik gefertigt (siehe Unterabschnitt 3.4.1). Werden auf einem DSP mit Festkommarithmetik Gleitkommaberechnungen durchgeführt, so werden diese emuliert und benötigen mehrere Taktzyklen, das einen großen Einfluss auf die Performanz hat (vgl. Analog Devices Inc. 2019a, Kap. S. 2-257). Durch die Wahl geeigneter Zahlenformate kann die Heterogenität gesteuert werden.

Datentypen

Auch müssen bei der Verwendung von heterogenen Systemen die jeweiligen Datentypen und deren unterschiedliche Definitionen berücksichtigt werden. Gewöhnlich handelt es sich beispielsweise bei der Verwendung der Programmiersprache C bei dem Datentyp Double um Gleitkommazahlen mit doppelter Genauigkeit (64 Bit). Dies ist allerdings nicht zwingend der Fall. Beispielsweise wird bei Blackfin oder auch SHARC-Prozessoren von Analog Devices standardmäßig der Datentyp Double als 32-Bit Zahl behandelt. Dies kann zwar mittels Compiler-Einstellungen geändert werden, jedoch werden beispielsweise für den Blackfin-Prozessor die Berechnung von Double mit 64 Bit emuliert und somit deutlich langsamer ausgeführt (vgl. Analog Devices Inc. 2019a, Kap. 2 S. 12). Auch die Kompatibilität zu proprietären Bibliotheken muss berücksichtigt

werden, wenn der Datentyp Double anders definiert wurde. Eine einheitliche Definition von Gleitkommazahlen wird durch den IEEE-754 Standard (*754-2008 - IEEE Standard for Floating-Point Arithmetic* 2008) vorgenommen. Es empfiehlt sich Gleitkommazahlen zu verwenden, die beispielsweise nach diesem Standard spezifiziert wurden.

Beliebige Bitbreite

Im Gegensatz zu gewöhnlichen Prozessoren kann bei stark spezialisierten Prozessoren wie Field Programmable Gate Arrays (FPGA) oder Application Specific Integrated Circuits (ASIC) die Bitbreite dieser Prozessoren beliebig entworfen werden. Dies ermöglicht den Entwurf von maßgeschneiderten Lösungen für spezielle Anwendungen, die beliebige Bitbreiten benötigen. Dabei kann durch eine exakte Bitbreite die Bauteilgröße und die benötigte Energie beeinflusst werden. Allerdings kann beispielsweise mit der hardwarenahen Programmiersprache C das Potentzial der beliebigen Bitbreite nicht ausgeschöpft werden, da eine beliebige Bitbreite in der Regel nicht unterstützt wird und die Sprache für diesen Zweck manuell erweitert werden muss. Dadurch kann nicht ohne weiteres C Code für eine spezialisierte Hardware mit beliebiger Bitbreite kompiliert werden (vgl. CC (Conference), Duesterwald und ETAPS (Conference) 2004, S. 250). Seit geraumer Zeit kann mithilfe von High Level Synthesis (HLS) C Code in Hardware Description Language (HDL) umgewandelt und für FPGAs oder ASICs synthetisiert werden. Dabei sind nun gerade die fixen Bitbreiten der Programmiersprachen C eine Limitierung, da durch dies folglich unnötige Logikgatter bei den FPGAs oder ASICs verwendet werden. Compiler wie der LLVM-IR unterstützen mittlerweile beliebige beziehungsweise exakte Bitbreiten und beheben das Problem (vgl. Keane 2020).

Cache-Koheränz

Werden bei Mehrkernprozessoren Daten untereinander geteilt und die Zwischenspeicherung in Caches durchgeführt, so führt dies zu dem Problem der Cache-Kohärenz. Jeder Prozessor puffert den geteilten Speicher mittels eigenem Cache und so besteht das Risiko, dass beide Prozessoren unterschiedliche Werte sehen. Daher muss in einem Mehrkernprozessor die Kohärenz der privaten Caches kontrolliert und gesteuert werden. (vgl. Hennessy 2019, S. 377)

2.4.3 Amdahls Gesetz

Aufgrund dem in Abschnitt 2.1 genannten Ende des Dennard Scalings und dem daraus resultierenden Trend zur Parallelisierung und Spezialisierung von Prozessoren werden Methoden zur Beurteilung der Leistungssteigerung benötigt. Zur Beurteilung der Beschleunigung die durch die Parallelisierung von Aufgaben erzielt werden soll, kann Amdahls Gesetz verwendet werden. Durch

Amdahls Gesetz wird die maximale Beschleunigung ermittelt, die mit symmetrischen Multiprozessoren erzielt werden kann. (vgl. Marowka 2012 und Hennessy 2019, S. 5) Folgende Gleichung beschreibt das Gesetz von Amdahl.

$$Speedup_s = \frac{1}{(1-f) + \frac{f}{c}} \tag{2.3}$$

Dabei gilt: c ... Anzahl Prozessorkerne
f ... parallelisierbarer Anteil eines Programms ($0 \leq f \leq 1$)
(Marowka 2012)

Dabei beschreibt Amdahls Gesetz in Gleichung 2.3, dass die Beschleunigung der Rechenleistung durch den nicht parallelisierbaren Anteil limitiert wird. Da lediglich ein gewisser Anteil parallelisiert werden kann und somit der verbleibende sequentielle Anteil auf einem einzelnen Kern durchgeführt werden muss. (vgl. Hennessy 2019, S. 49) Allerdings handelt es sich bei Amdahls Gesetz um ein theoretisches Modell, das keine weiteren Einschränkungen berücksichtigt. Mögliche Einschränkungen des Modells, die es in der Praxis zu berücksichtigen gibt, sind der Energieverbrauch von Prozessoren und die Verwendung von asymmetrischen Prozessoren mit unterschiedlicher Rechenleistung. (vgl. Marowka 2012) Folgende Gleichung stellt eine Erweiterung von Amdahls Gesetz für ein asymmetrisches Multiprozessorsystem mit CPU und GPU Kernen dar. Dabei wird die Ausführung eines Programms in drei Abschnitte eingeteilt. Ein Abschnitt ist f, bei dem es sich um den sequentiellen Anteil handelt der auf einer CPU ausgeführt wird. Ein weiterer Abschnitt ist α, bei dem das Programm ausschließlich auf der CPU ausgeführt wird. Beim Abschnitt β wird die Berechnung nur auf der GPU durchgeführt.(vgl. Marowka 2012)

$$Speedup_a = \frac{1}{(1-f) + \frac{\alpha f}{c} + \frac{(1-\alpha)f}{g\beta}} \tag{2.4}$$

Dabei gilt: c ... Anzahl CPU Kerne
g ... Anzahl GPU Kerne
f ... gesamter parallelisierbarer Anteil eines Programms ($0 \leq f \leq 1$)
α ... parallelisierbarer Anteil eines Programms CPU ($0 \leq \alpha \leq 1$)
β ... parallelisierbarer Anteil eines Programms GPU ($0 \leq \beta$)
(Marowka 2012)

Gleichung 2.4 berücksichtigt jedoch nicht den Fall der echten Nebenläufigkeit, bei der die CPU und GPU Kerne gleichzeitig rechnen. Für den Fall, dass beim parallelisierten Anteil die CPU Rechenzeit der GPU Rechenzeit entspricht, sind die entsprechenden Terme gleichzusetzen ($\frac{\alpha f}{c} = \frac{(1-\alpha)f}{g \cdot \beta}$). Somit vereinfacht sich die Gleichung Gleichung 2.4 zu folgender Gleichung. (vgl. Marowka 2012)

$$Speedup_{sa} = \frac{1}{(1-f) + \frac{f}{g \cdot \beta + c}} \tag{2.5}$$

Dabei gilt:
- c ... Anzahl CPU Kerne
- g ... Anzahl GPU Kerne
- f ... gesamter parallelisierbarer Anteil eines Programms $(0 \leq f \leq 1)$
- β ... parallelisierbarer Anteil eines Programms GPU $(\beta = 0,5)$(Marowka 2012)

Anhand der Gleichungen 2.4 und 2.5 die durch Marowka 2012 erarbeitet wurden, ist die entstehende Komplexität zu erkennen, die beim Beurteilen der Leistungssteigerung von heterogenen Systemen entsteht. Beispielsweise kann keine der beiden Gleichungen für Systeme verwendet werden, bei denen mehr als zwei unterschiedliche Prozessoren verwendet werden. Auch können unterschiedliche Taktfrequenzen für die gleichen Kerne in den genannten Erweiterungen des amdahlschen Gesetz nicht ohne Weiteres berücksichtigt werden. Wie auch für die Beurteilung der Beschleunigung der Rechenzeit kann auch der Energieverbrauch anhand mathematischer Modellierungen bestimmt werden. Die Bestimmung des Energieverbrauchs von heterogenen Systemen ist ebenso ein schwieriges Vorhaben wie die Bestimmung der Beschleunigung der Rechenzeit. Folgende weiterführende Literatur hinsichtlich des Energieverbrauchs und der Erweiterung des amdahlschen Gesetz ist zu empfehlen: Marowka 2012, Song, Mukhopadhyay und Yalamanchili 2016, Hill und Marty 2008 sowie Woo und Lee 2008

2.5 Programmiermodelle

Infolge der Integrierung von mehreren Prozessoren in ein einzelnes System, bedarf es an Lösungen, um die Prozessoren problemlos programmieren und gemeinsam verwenden zu können. Dabei bedarf es an Kommunikationsmöglichkeiten und Möglichkeiten mit denen Programme ohne Schwierigkeiten parallelisiert werden können. Die meisten bestehenden Frameworks wurden jedoch für Desktop- bzw. Server-Applikationen oder für symmetrische Multiprozessorsysteme entwickelt. In diesem Unterkapitel werden die derzeitigen Möglichkeiten zur Programmierung von eingebetteten asymmetrischen Multiprozessorsystemen beschrieben.

2.5.1 Multicore Association Standards

Aufgrund der Tatsache, dass die Homepage der Multicore Association nicht mehr erreichbar ist und die infolge beschriebenen eingetragenen Marken (MCAPI, MRAPI und MTAPI) dieser Vereinigung annulliert wurden (vgl. WIPO Global Brand Database 2020a, WIPO Global Brand Database 2020b und WIPO Global Brand Database 2020c), muss davon ausgegangen werden, dass die Vereinigung nicht mehr in der einstigen Form existiert. Die Multicore Association war eine Vereinigung von Unternehmen, die ein Forum für herstellende Unternehmen im Bereich der Multiprozessorsysteme zur Verfügung stellte. Das Konsortium hat mittels mehreren Arbeitsgruppen eigene Spezifikationen für die Programmierung von Multiprozessorsystemen und deren Kommunikation definiert (vgl. *The Multicore Association* 2018). Erwähnenswert dabei sind die folgenden Standardisierungen, die von der Multicore Association vorgenommen wurden.

- MCAPI - Multicore Communications API
- MRAPI - Multiore Resource Management API
- MTAPI - Multicore Task Management API
- OpenAMP - Open Asymmetric Multi Processing
- SHIM - Software-Hardware Interface for Multi-Many Core

Der MCAPI Standard ist in diesem Fall besonders erwähnenswert, da dieser die Schnittstellen für die Inter-Core-Communication definiert und von einigen herstellenden Unternehmen und Drittanbietenden implementiert wurde. Dabei handelt es sich bei der MCAPI um einen standardisierten Nachrichtenaustausch zwischen Prozessorkernen, bei der Nachrichten zur Kommunikation und Synchronisation verwendet werden (vgl. Analog Devices Inc. 2015). Beispielsweise unterstützt Analog Devices für die ADSP-Sc58x Produktreihe den MCAPI Standard. Zudem bietet die dritt-anbietende Firma PolyCore Software und deren Poly-Platform MCAPI Unterstützung durch ein Eclipse Plugin für folgende Prozessoren (vgl. PolyCore Software 2020):

- TI DSP C6x
- Tensilica DSP
- ARM
- x86

- Power PC
- Blackfin DSPs

Durch diese Unterstützung können die in Tabelle 4.1 genannten Prozessoren mit MCAPI verwendet werden. Nennenswert ist auch der Open-Source OpenAMP Standard, der nun von Linaro weiterentwickelt wird. OpenAMP wird verwendet um die Interaktion zwischen asymmetrischen Prozessoren und deren Umgebung in eingebetteten Systemen zu standardisieren (vgl. OpenAMP 2019). Texas Instruments hat einen ersten Prototyp für dessen C6x DSPs implementiert (vgl. Texas Instruments 2020b).

2.5.2 OpenCL

OpenCL ist ein offener Standard, der die Parallelisierung von Aufgaben über heterogene Prozessoren ermöglicht. Der Standard wird durch das Khronos Group Konsortium spezifiziert. OpenCL verwendet in einem heterogenen System einen Kern als Host und einen oder mehrere Kerne als OpenCL Geräte. Der Host ist für die Parallelisierung verantwortlich und verteilt die Berechnungen an die OpenCL Geräte. Durch die Standardisierung wird die Programmierung von heterogenen SoCs vereinfacht, sodass durch das Verwenden von Programmierschnittstellen Code und Daten problemlos an die jeweiligen Kerne verteilt werden können (vgl. Texas Instruments 2020c). OpenCL wird im Bereich der eingebetteten Systeme beispielsweise von Texas Insturments, Xilinx, ARM und ST verwendet (vgl. The Khronos Group 2013). Texas Instruments unterstützt OpenCL beispielsweise für die in Tabelle 4.1 genannten Prozessoren der Produktreihe 66AKx. Die Implementierung verwendet einen ARM-Cortex A15 als Host und die DSP Kerne als OpenCL Geräte. Der Vorteil dabei ist in diesem Fall, dass Applikationen ohne Weiteres von einem System mit nur vier DSP Kernen auf ein System mit acht DSP Kernen migriert werden kann, da der Host nun lediglich die Berechnungen auf acht anstatt vier DSP Kerne aufteilen muss (vgl. Texas Instruments 2020c).

2.5.3 OpenMP

OpenMP ist ein Standard für paralleles Rechnen mit eng gekoppelten und homogenen Multiprozessorsystemen, der für die Shared-Memory Programmierung verwendet wird. Des Weiteren kann OpenMP auch für homogene Prozessoren in heterogenen Ein-Chip-Systemen verwendet werden. Der Standard bietet mittels Compiler-Anweisungen und Bibliotheken die Möglichkeit der Parallelisierung von bestehendem Code. Zusätzlich wird durch die Anwendung des Standards eine Skalier- und Portierbarkeit erreicht. Wiederum wurde OpenMP von Texas Instruments bereits für dessen ARM und DSP System implementiert. Texas Instrument hat für die OpenMP Unterstützung den Compiler für die C66x DSPs

angepasst. Für die ARM Kerne wird die GCC Compiler-Suite verwendet, die auch bereits OpenMP unterstützt (vgl. Blinka 2014).

HetroOMP

HetroOMP ist eine vorgeschlagene Erweiterung von OpenMP, die paralleles Rechnen für heterogene Systeme ermöglicht. Durch die Verwendung von HetroOMP ist es möglich, dass der Host auch an der Berechnung teilnimmt und diese nicht nur an den Beschleuniger weiterleitet. Beispielhaft wurde anhand des Keystone II Multiprozessor Ein-Chip-Systems von Texas Instruments ein Konzeptnachweis erbracht, der die in Tabelle 4.1 genannten Prozessoren C66x und ARM-Cortex A15 enthält (vgl. Kumar, Tiwari und Mitra 2019).

3 Heterogene Prozessoren in eingebetteten Systemen

Im Folgenden werden unterschiedliche Arten von Prozessoren und deren Eigenschaften erläutert, die bei eingebetteten Systemen zur Anwendung kommen. Dabei wird zuallererst der Begriff Prozessor definiert.

3.1 Definition eines Prozessors

Ein Prozessor ist eine elektronische Baugruppe oder Einheit, die über ein Rechen- und Steuerwerk verfügt und zum Anwenden von Rechenoperationen auf Datenströme verwendet wird (vgl. Fischer und Hofer 2008, S. 661). Mit dieser Definition soll verdeutlicht werden, dass beispielsweise bereits ein Hardware-Beschleuniger für die schnelle Fourier-Transformation ein spezialisierter Prozessor ist, da dieser Rechenoperationen auf Datenströme durchführt und die eingehenden sowie ausgehenden Daten mittels Steuerwerk steuert. Ein DMA-Controller ist hingegen kein Prozessor, da dieser lediglich als Steuerwerk fungiert und über kein eigenes Rechenwerk verfügt und somit die Daten nicht verändert. In den folgenden Abschnitten wird auf die unterschiedlichen Prozessoren und deren Eigenschaften eingegangen.

3.2 Arten von Prozessoren

In Abschnitt 2.1 wurde erläutert, weshalb eine Leistungssteigerung von Prozessoren oft durch die Parallelisierung und Spezialisierung dieser erzielt wird. Folgende Abbildung zeigt kurz und prägnant den Kompromiss, die bei der Verwendung von Special Purpose Processors (SPP) in gelb gegenüber eines herkömmlichen Prozessors (GPP) in grün eingegangen wird. Je spezialisierter der Prozessor ist, desto besser ist dessen Energieverbrauch und Performanz. Allerdings wird durch die Spezialisierung die Flexibilität in Bezug auf das Lösen von anderen Aufgaben stark reduziert oder sogar eliminiert. Dabei weist der GPP die höchste Flexibilität hinsichtlich dessen Fähigkeiten zur Lösung einer Aufgabe auf, jedoch muss oft eine hohe Energieaufnahme oder eine langsamere Rechengeschwindigkeit in Kauf genommen werden. Eine erste Spezialisierung sind domänenspezifische Prozessoren wie beispielsweise DSPs, die für die digitale Signalverarbeitung speziell im Video- und Audiobereich verwendet werden.

Weitere Spezialisierungen sind FPGAs, die bereits ein sehr gutes Verhältnis im Bezug auf Performanz und Energieverbrauch haben. ASICs bieten gegenüber von Hardware-Beschleunigern noch einen gewissen Teil an Flexibilität, da auch bei einem ASIC die Integration von GPPs möglich ist. Hardware-Beschleuniger sind so stark spezialisiert, dass die Verwendung für andere Aufgaben nicht realisierbar ist, aber dafür den besten Energieverbrauch sowie Performanz aufweisen. (vgl. Uchiyama u. a. 2012, S. 5 - 6)

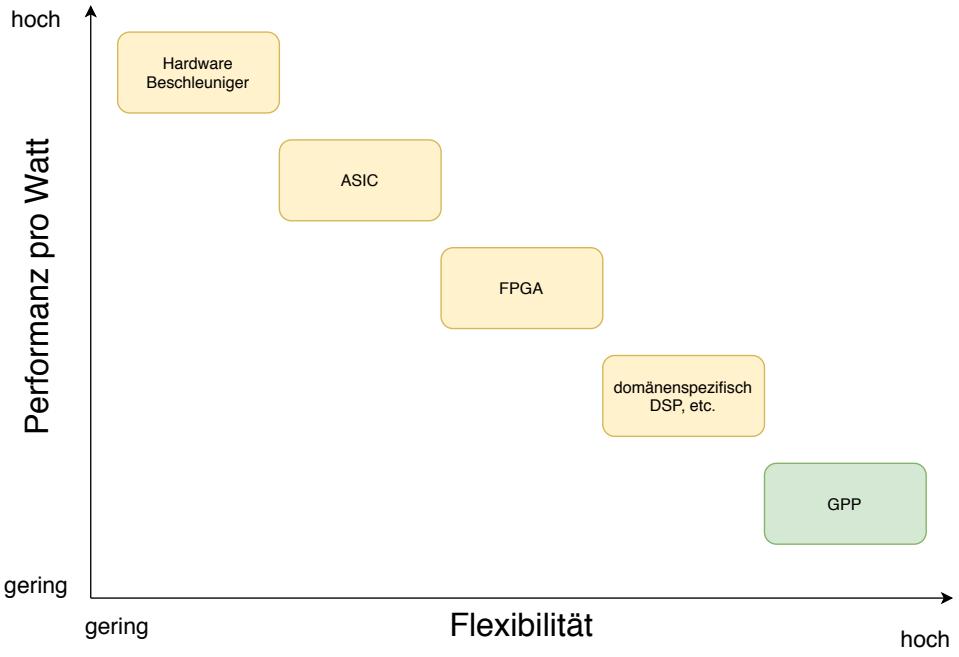


Abbildung 3.1: Flexibilität vs. Performanz unterschiedlicher Prozessoren (In Anlehnung an Uchiyama u. a. 2012, S. 5 und Mansureh, Cho und Choi 2016, S. 6)

In den folgenden Abschnitten werden die jeweiligen Eigenschaften der unterschiedlichen Prozessoren genauer erläutert.

3.3 General Purpose Processor

Grundsätzlich wurden GPPs nicht für einen konkreten Anwendungsfall entwickelt. Sie bieten eine hohe Flexibilität hinsichtlich deren Anwendungsgebiet. Deshalb sind GPPs in Desktop-Computern und Mikrocontrollern zu finden. GPPs verfügen über Gemeinsamkeiten wie unter anderem eine Memory-Management Unit (MMU) sowie einen verallgemeinerten Befehlssatz. GPPs liefern eine durchschnittliche Performanz und die Steigerung dieser ist nur durch

intensive Maßnahmen wie dem Erhöhen der Taktfrequenz, der Anwendung von spezialisierten Befehlssätzen oder durch softwaretechnische Adaptierungen erzielbar (vgl. Nowatzki, Gangadhar und Sankaralingam 2019, S. L. Harris und D. M. Harris 2016, S. 469 und Hennessy 2019, passim).
Im Gegensatz dazu gibt es auch eine andere Betrachtungsweise, die argumentiert, dass es keinen GPP gibt. David Chisnall von der Univeristät Cambridge argumentiert in seinem Fachreferat (siehe Chisnall 2020), dass es keinen GPP gibt, weil heutige GPPs auch stark spezialisiert sind und diese nur aufgrund der geringsten Kosten für generelle Zwecke verwendet werden. Des Weiteren wird dargelegt, dass viele Algorithmen auf die Eigenschaften von GPPs angepasst sind und dieser Trend sich nun umkehre, da nun Grafikprozessoren (Graphic Processing Unit - GPU) hinsichtlich Floating Point Operations Per Second (FLOPS) günstiger als herkömmliche CPUs sind. Deshalb werden nun Algorithmen für GPUs angepasst, obwohl diese keine Datenparallelität aufweisen. (vgl. Chisnall 2020) Dabei handelt es sich um ein valides Argument, jedoch wird in dieser Arbeit trotzdem der Begriff General Purpose Processor verwendet. Zum einen weil dieser in der Literatur Anerkennung findet. Zum anderen weil der aktuelle Stand der Technik es nicht zulässt, für einen weiten Anwendungsbereich andere Prozessoren als herkömmliche CPUs zu verwenden. Der Begriff General Purpose Processor erschließt sich daher nicht aus den oben genannten technischen Details, sondern vor allem durch den aktuellen Entwicklungsstand und durch dessen Gegebenheiten hinsichtlich der praktischen Anwendung. Unter anderem verwenden folgende wissenschaftliche Quellen, die im Rahmen dieser Arbeit verwendet werden den Begriff General Purpose Processor: Hennessy 2019, S. L. Harris und D. M. Harris 2016, Terzo 2019, Zahran 2017, Wolf 2007, Oshana 2012, ARM Limited 2011. Dies soll verdeutlichen, dass es sich bei General Purpose Processor durchaus um einen gängigen Begriff in der Informatik handelt.

3.4 Special Purpose Processor

Dieses Kapitel beschreibt gängige SPP und ihre Eigenschaften in eingebetteten Systemen.

3.4.1 Digitaler Signalprozessor

Das Akronym DSP wird oft bei der Vermarktung von Prozessoren verwendet und ist irreführend, da DSP im Englischen auch als Digital Signal Processing verwendet wird und nicht zwingend digitaler Signalprozessor bedeutet. Daher muss unterschieden werden, ob es sich bei einem beworbenen Prozessor um einen digitalen Signalprozessor, oder einen Prozessor handelt, der lediglich für die digitale Signalverarbeitung verwendet werden kann (vgl. Wolf

2007, Kap. 2 S. 71). Grundsätzlich unterscheiden sich digitale Signalprozessoren von GPPs bereits durch deren Prozessorarchitektur. Bei DSPs wird eine Harvard-Architektur oder eine adaptierte Harvard-Architektur verwendet. Dabei wird gewöhnlich für GPPs eine Von-Neumann Architektur verwendet. Zusätzlich verfügen DSPs über einen eigenen Multiplizierer in deren ALU, da in der Signalverarbeitung oft zwei Faktoren multipliziert werden und anschließend das Produkt zu einem fortlaufenden Summanden addiert wird, verfügen DSPs über einen eigenen Maschinenbefehl für diese Operation (Multiply-Accumulate (MAC) oder Multiply-Add (MAD)) (vgl. Wolf 2007, Kap. 2 S. 71 - 72). Beispielsweise werden MAC Befehle bei der Berechnung des Skalarprodukts (siehe folgende Gleichung), der Faltung oder bei der Auswertung von Polynomen verwendet.

$$\vec{a} \cdot \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 \tag{3.1}$$

Da DSPs oft für Echtzeitanforderungen eingesetzt werden, verfügen DSPs über weitere Hardware-Komponenten, um möglichen Software-Mehraufwand zu reduzieren. Folgenden Hardware-Komponenten werden häufig in DSPs bereitgestellt. (vgl. Oshana 2012, Kap. 3 S. 46 und Microchip Technology Inc. 2020)

- **Zero Overhead Loops** Der Endpunkt einer Schleife wird durch die Hardware ermittelt, daher bedarf es keiner Software, die überprüft ob die Schleife am Endpunkt angekommen ist. (vgl. Microchip Technology Inc. 2020)
- **Barrel-Shifter** Hardware-Komponente, um Bitvektoren zu verschieben oder rotieren. (vgl. Oshana 2012, Kap. 3 S. 46 und Microchip Technology Inc. 2020)
- **spezialisierte ALU** Handhabung von mathematischen Problemen von Fest- und Gleitkommazahlen wie Sättigungsarithmetik, Runden und Überlauf (vgl. Oshana 2012, Kap. 3 S. 46 und Microchip Technology Inc. 2020)

Aufgrund der zusätzlich verfügbaren Hardware-Komponenten eines DSPs bedarf es an Compilern die eine starke Spezialisierung auf den jeweiligen Prozessor und dessen speziellen Befehlssatz aufweisen.

Gleit- und Festkommaarithmetik

Zur Berechnung von reellen Zahlen werden DSPs mit Gleit- oder Festkommaarithmetik gefertigt, selten auch mit beiden Arithmetiken (vgl. Wüst 2011, S. 307). Die Vorteile von Festkommaarithmetik sind ein geringerer Energieverbrauch, schnellere Rechenleistung, geringere Kosten und eine exakte Darstellung einer Zahl über den gesamten Wertebereich. Die Kehrseite dabei ist ein

größerer Aufwand bei der Implementierung von Algorithmen, da es nach einer Rechenoperation einer Skalierung der Zahlen bedarf, dabei kann es zu einer reduzierten Genauigkeit kommen (vgl. Gan und Kuo 2007, S. 252). Algorithmen mit Gleitkommazahlen werden bei DSPs mit Gleitkommarithmetik hingegen leichter implementiert, weil keine Skalierung mehr berücksichtigt werden muss. Bei der Verwendung von Gleitkommazahlen soll beachtet werden, dass die doppelte Genauigkeit nach IEEE (*754-2008 - IEEE Standard for Floating-Point Arithmetic* 2008) in der Regel bei DSPs von eingebetteten Systemen nicht unterstützt wird (siehe auch Abschnitt 2.4.2).

Von-Neumann versus Harvard-Architektur

Prozessoren mit einer Harvard-Architektur verwenden getrennte Programm- und Datenspeicher, die über unterschiedliche Busse angesteuert werden. Entgegengesetzt zu der Harvard-Architektur werden bei Prozessoren mit Von-Neumann-Architektur der Programm- sowie Datenspeicher über einen gemeinsamen Bus adressiert. Eine Erweiterung der Harvard-Architektur ist die Super-Harvard bzw. die adaptierte Harvard-Architektur, die über mehrere Programm- sowie Datenbusse verfügt, aber einen einzelnen, vereinheitlichten Adressraum für die Adressierung der Daten- und Programmspeicher verwendet. Dabei kann der Datenbus (DM) lediglich Daten und der Bus für den Programmspeicher (PM) Daten und Befehle handhaben. Dies ermöglicht einen doppelten Datenzugriff innerhalb eines Taktzyklus. Durch dies kann mit einem Taktzyklus der nächste Befehl sowie zwei Speicheroperanden gelesen werden (vgl. Analog Devices Inc. 2019d, Kap. 7 S. 2 und Wüst 2011, S. 350).

3.4.2 Field Programmable Gate Array

Field Programmable Gate Arrays (krz. FPGA) sind keine Prozessoren im herkömmlichen Sinne. FPGAs sind elektronische Bausteine, die eine frei konfigurierbare Verschaltung von zahlreichen Logikgattern ermöglichen (Fischer und Hofer 2008, S. 319 und Marwedel 2008, S. 127). Durch die hohe Transistordichte die in Halbleiterbausteinen erzielt werden kann, werden FPGAs mit Millionen von Logikgattern gefertigt. Dies ermöglicht die Realisierung von komplexer Logik wie zum Beispiel Prozessoren oder Algorithmen für die Signalverarbeitung. Die Verschaltung der einzelnen Logikgattern wird durch Hardware Beschreibungssprachen wie VHDL und Verilog oder einen Schaltplan vorgegeben. FPGAs bieten gegenüber ASICs (siehe Unterabschnitt 3.4.4) den Vorteil, dass sie problemlos rekonfiguriert und aufgrund der geringen Kosten hinsichtlich deren Anschaffung auch für Kleinstserien oder einzelne Projekte verwendet werden.

3.4.3 Hardware-Beschleuniger

Wie bereits aus Abbildung 3.1 zu entnehmen ist, sind Hardware-Beschleuniger Prozessoren, die für eine bestimmte Problemstellung entwickelt wurden, um diese möglichst effizient und/oder schnell zu lösen. Hardware-Beschleuniger bieten keine Flexibilität und werden nicht für generelle Aufgaben verwendet. Hardware-Beschleuniger werden beispielsweise bei ARM und DSP SoCs von Analog Devices und Texas Instruments für folgende Anwendungsgebiet bereitgestellt.

· **Signalverarbeitung** FFT, Filter mit Impulsantwort, Filter mit unendlicher Impulsantwort, Harmonic Analysis Engine, Sinus-Kardinal Filter

· **Kryptographie** Public Key Accelerator, Security Crypto Engine (vgl. Analog Devices Inc. 2020a und Texas Instruments 2020a)

3.4.4 Application Specific Integrated Circuit

Anwendungsspezifische integrierte Schaltkreise (engl. Application Specific Integrated Circuit (ASIC)) werden hergestellt, wenn eine möglichst hohe Performanz hinsichtlich Energieeffizienz benötigt wird und die Herstellung finanziell auch tragbar ist (vgl. Marwedel 2008, S. 109). Dabei werden bei ASICs Logikbausteine, Prozessoren und Analoge-Komponenten integriert. ASICs bieten gegenüber Hardware-Beschleunigern eine mögliche Flexibilität, wenn beispielsweise auf dem Integrated Circuit (IC) herkömmliche Prozessoren integriert sind. Wenn jedoch keine Flexibilität durch einen integrierten GPP oder domänenspezifische Prozessoren gegeben ist, sind ASICs in Abbildung 3.1 mit Hardware-Beschleuniger gleichzusetzen.

4 ARM und DSP System-On-Chip

Die heterogenen Eigenschaften von GPP und DSP ermöglichen eine zweckmäßige Aufteilung der Aufgaben in einem System. Dabei werden DSPs für rechenintensive Aufgaben für die Signalverarbeitung und GPPs beispielsweise für Steueraufgaben verwendet. Durch die Integration von DSPs und GPPs in einem Die kann eine verbesserte Performanz und ein optimierter Energieverbrauch erzielt werden. Überdies kommen noch zusätzliche Effekte wie die Kostenreduzierung oder Platzersparnis zu tragen, da es nun nur noch an einem Chip bedarf. Allerdings wird durch die Integrierung der beiden Komponenten auch die Komplexität bezüglich deren Programmierung erhöht. Im Folgenden wird ein Überblick über die derzeitigen Ein-Chip-Systeme verschafft, die für GPP Technologien von ARM und für die Signalverarbeitung DSP Kerne verwenden. Dabei wird in Abschnitt 4.1 zuerst erläutert, warum herstellende Unternehmen auf die Technologien von ARM setzen.

4.1 ARM

Die Firma ARM Limited ist eine Firma die Technologien und geistiges Eigentum zur Herstellung von Prozessoren im Bereich der Mikroprozessortechnik anbietet. ARM lizenziert seine Produkte an Unternehmen, die Prozessorkerne von ARM mit zusätzlicher Peripherie ausstatten und herstellen. ARM selbst produziert keine Prozessoren. Bevor ARM mit seinen unterschiedlichen Prozessorarchitekturen und Befehlssätzen kommerziellen Erfolg feierte, war der Mikroprozessormarkt sehr heterogen. Viele Unternehmen produzierten Mikrocontroller mit eigenen Architekturen und Befehlssätzen, die eigene Compiler, Debug Interfaces, Betriebssysteme und Entwicklungsumgebungen benötigten. Dies führte dazu, dass für Entwickelnde ein Umstieg auf einen neuen Prozessor ein zeitaufwändiges und kostspieliges Vorhaben war. Durch das Verwenden eines Lizenzierungsmodells für die Technologien von ARM wurde erreicht, dass vielerlei Mikrocontroller von untereinander konkurrierenden Unternehmen dieselbe Grundarchitkektur verwenden. Somit kann bei der Entwicklung aus einer großen Auswahl an Mikrocontroller gewählt werden, die dieselbe Architektur und Befehlssätze verwenden. Durch das Lizenzierungsmodell wurde somit

am Markt eine gewisse Standardisierung erreicht, die zu einer starken Vereinfachung der Entwicklungsprozesse führte. ARM versucht, weitere Teile von Mikrocontrollern wie die Zugriffe auf Peripherie-Bausteine zu standardisieren. Für dies wurde der Cortex-Microcontroller-Software-Interface-Standard (CMSIS) entworfen, der Zugriffe auf Peripheriegeräte spezifiziert. (vgl. Asche 2016, S. 21 - 23) Aufgrund der guten Akzeptanz und Verbreitung von ARM Prozessoren bei einfachen Mikrocontrollern und der daraus in gewisser Weise resultierenden Standardisierung werden diese auch vermehrt in Multiprozessor-Ein-Chip-Systemen mit spezialisierten Prozessoren kombiniert. Durch dies wird es Anwendenden ermöglicht, generelle Aufgaben in einem ARM Kern zu implementieren und für die spezialisierten Aufgaben den geeigneten SPP zu wählen, wie diese bereits zuvor beschrieben wurde.

4.2 RISC-V - Alternative zu proprietären Befehlssatzarchitekturen

Aufgrund der starken Marktdurchdringung der ARM Befehlssatzarchitekturen sowie der x86 Befehlssatzarchitekturen der Firma Intel, bilden diese Unternehmen mit ihren Technologien ein weltweites Duopol hinsichtlich der Verbreitung von Prozessoren und ihren Befehlssatzarchitekturen. (vgl. Greengard 2020) Die proprietären Mikroprozessoren von ARM und Intel werden in den unterschiedlichsten Applikationen verwendet und sind nahezu unumgänglich. Nichtsdestotrotz ersehnen sich Forschende und Entwickelnde seit geraumer Zeit freie Befehlssätze, die auf ihr Bedürfnisse angepasst werden können. Infolgedessen wurde an der Universität von Kalifornien, Berkley die offene Befehlssatzarchitektur RISC-V entwickelt, die eine hohe Flexibilität bei geringen Kosten bietet. Dadurch ist es Entwickelnden möglich eigene Chips herzustellen, die für ihre Anwendungen geeignet sind. (vgl. Greengard 2020) RISC-V bietet einen Softwarestack, der Compiler, Betriebssystem und einen Simulator beinhaltet. Außerdem stehen bereits mehrere Implementierungen für FPGAs und benutzerdefinierte Chips frei zur Verfügung. (vgl. Hennessy 2019, S. 12) Durch das modulare Design des Befehlssatzes, kann je nach Bedarf ein Paket zur Erweiterung der Einsatzmöglichkeiten verwendet werden, sodass durch dies die Performanz und der Energieverbrauch beeinflusst werden kann. Aus diesem Grund wird RISC-V zunehmend für herstellende Unternehmen interessant, da aufgrund der offenen Befehlssatzarchitektur diese eigene Prozessoren herstellen können. Dabei sind es zum einen wirtschaftliche Gründe, da hinsichtlich der Verwendung geringere Kosten anfallen, zum anderen ermöglicht RISC-V den Unternehmen tiefere Einblicke in die Funktionsweise des Prozessors und erlaubt es, eigene Features zu implementieren. Dementsprechend gibt es keine Limitierungen in Bezug auf das Anwendungsgebiet, sodass ein Prozessor beispielsweise für die Berechnung von

neuronalen Netzwerken optimiert werden kann. Dies führt dazu, dass Produkte nicht mehr an die bestehenden Chips angepasst werden. Im Gegenteil, nun werden für neue Produkte geeignete Chips entworfen. (vgl. Greengard 2020) Aufgrund dieser Tatsachen sind bereits 60 Unternehmen der RISC-V Stiftung beigetreten, darunter sind bekannte Technologiekonzerne wie AMD, Google, Microsoft, Western Digital, NVIDIA, Qualcomm und Samsung zu finden (vgl. Hennessy 2019, S. 12). Die letzteren vier der aufgezählten Unternehmen haben bereits angekündigt, dass diese in Zukunft RISC-V in ihren Produkten einsetzen werden. RISC-V übt durch seinen zunehmenden Erfolg Druck auf Unternehmen wie ARM und Intel aus, da RISC-V bereits erste Marktanteile bei Mikroprozessoren für sich beansprucht. Ebendarum ist es nicht verwunderlich, dass ARM ein großer Kritiker von RISC-V ist und es für ARM zu einem Interessenskonflikt kommt. Infolgedessen war ARM bereits im Zugzwang und hat zuerst eine Anti-RISC-V Homepage für einige Tage online gestellt. Anschließend wurde die Strategie von ARM geändert und es wurde die Öffnung der Cortex M Befehlssatzarchitektur angekündigt, sodass Kunden diese adaptieren können. Im Gegensatz dazu ist Intel einer der großen Geldgeber von RISC-V, obwohl RISC-V in direkter Konkurrenz zu deren Technologie steht. (vgl. Greengard 2020) Kritiker argumentieren, dass RISC-V eine Fragmentierung der Industrie und deren verwendeten Befehlssatzarchitekuren verursacht. Ferner drohen dem Standard mögliche Kompatibilätsprobleme bei bestehenden Applikationen, die für ARM Befehlssätze entworfen wurden. (vgl. Greengard 2020) Nichtsdestotrotz überwiegen die Vorteile und es werden dem offenen Standard bis 2025 bereits ein Marktanteil von 6 % bei CPU-Kernen vorhergesagt. (vgl. *Semico Forecasts Strong Growth for RISC-V* 2019 und Greengard 2020) Aufgrund der großen Unterstützung der Technologiekonzerne, scheint es, dass diese das Potenzial von offenen Standards und Open-Hardware erkannt haben. Google bietet beispielsweise in Kooperation mit Efabless die gratis Herstellung von Chips für Open-Hardware Projekte (vgl. Wagner 2020), dies verdeutlicht wie bedeutsam Open-Hardware Projekte auch für große Technologiekonzerne sind.
Nennenswert hinsichtlich des aktuellen Forschungsstands für heterogene Ein-Chip-Systeme ist die Heterogeneous Embedded Research Platform (HERO) (siehe Kurth u. a. 2017). Bei der Veröffentlichung von Kurth u. a. 2017 wird darauf hingewiesen, dass führende Unternehmen ihre heterogenen Ein-Chip-Systeme verbessern, jedoch hinkt die Wissenschaft in der Forschung in diesem Bereich hinterher. HERO ist eine Forschungsplatform, die aus Clustern aus RISC-V Softcores besteht und zur Ansteuerung ein ARM Cortex A Multiprozessor Chip verwendet. Dabei ist ein Softwarestack verfügbar, der eine heterogene Cross-Compilation Toolchain, Laufzeitbibliotheken, Linux Treiber und OpenMP Unterstützung zur Verfügung stellt. Durch das Bereitstellen von Lauf-

zeitbibliotheken und der effizienten Programmierung mittels OpenMP und der Cross-Compilation Toolchain kann Forschung auf der Systemebene betrieben werden, sodass standardisierte Benchmarks und Applikationen für heterogene Systeme problemlos portiert werden können. HERO wurde bereits als Open-Source Projekt veröffentlicht. (vgl. Kurth u. a. 2017 und *HERO: The Open Heterogeneous Research Platform* 2020)

4.3 Überblick von verfügbaren ARM und DSP SoCs

Ein-Chip-Systeme, die ARM sowie DSPs auf einem Die integrieren, werden unter anderem von den Herstellenden Analog Devices (ADI), NXP Semiconductors (NXP) und Texas Instruments (TI) gefertigt. Die Tabelle 4.1 gibt einen ausgewählten Überblick über verfügbare Produkte von Analog Devices, NXP Semiconductors und Texas Instruments. Mithilfe der Maßzahlen Dryhstone MIPS (DMIPS), Giga Floating Point Operation Per Second (GFLOPS) und Giga Multiply-Accumulate Operation Per Second (GMAC) kann die Leistungsfähigkeit der jeweiligen Prozessoren verglichen werden. Dabei bezieht sich das Maß DMIPS auf die ARM und die Maßzahlen GFLOPS und GMACS auf die DSP Kerne. Fehlende Angaben der Hersteller wurden nachträglich berechnet und in der Tabelle berücksichtigt. Die unterschiedlich verfügbaren Hardware-Beschleuniger der jeweiligen Prozessoren wurden in diesem Falle nicht berücksichtigt, da die jeweiligen Prozessoren nicht dieselben Beschleuniger zur Verfügung stellen und daher ein Vergleich nicht sinnvoll ist. Die Tabelle 4.1 soll zudem einen groben Überblick über die aktuelle Marktlage liefern. Eine Empfehlung für einen konkreten Prozessor kann nicht ohne Weiteres getroffen werden, da diese stark von der Applikation und deren Anforderungen abhängt. Die SoCs von Analog Devices wurden beispielsweise für eingebettete Systeme entwickelt, die Gleitkommaoperationen und einen sehr geringen Energieverbrauch von < 2 W aufweisen (vgl. Analog Devices Inc. 2020a). Das Äquivalent dazu ist von Texas Instruments der OMAP-L138, der jedoch nur in einer Ausführung bereitgestellt wird und einen bereits älteren ARM Prozessor verwendet. Die 66AK2x Serie von Texas Instruments wurde hingegen für hochperformante Anwendungsbereiche entwickelt, die heterogene sowie homogene Programmierung mittels OpenMP und OpenCL ermöglicht (vgl. Texas Instruments 2020a). Der i.MX RT600 Prozessor von NXP ist ein SoC, der es ermöglicht, als sprach-unterstützender Endknoten zu fungieren und einen optimierten Energieverbrauch aufweist (vgl. NXP Seminconductors 2020). Für beide Kerne wird jeweils geistiges Eigentum von den linzenzgebenden Unternehmen ARM und Cadence verwendet.
Die gemeinsame Eigenschaft dieser Ein-Chip-Systeme ist die Schwierigkeit der

Parallelisierung von Aufgaben sowie die Kommunikation zwischen den unterschiedlichen Prozessoren. In Abschnitt 2.5 werden Frameworks beschrieben, welche die Kommunikation und Parallelisierung standardisieren und bei Multiprozessorsystemen verwendet werden.

Analog Devices					
Typ	Arm Kerne	DSP Kerne	DMIPS	GFLOPS	GMACS
ADSP-SC582	1x Cortex-A5 500 MHz	1x SHARC+ 500 MHz	800	3	1
ADSP-SC583	1x Cortex-A5 300 MHz	2x SHARC+ 300 MHz	480	1,8	0,6
ADSP-SC583	1x Cortex-A5 500 MHz	2x SHARC+ 500 MHz	800	3	1
ADSP-SC584	1x Cortex-A5 300 MHz	2x SHARC+ 300 MHz	480	1,8	0,6
ADSP-SC584	1x Cortex-A5 500 MHz	2x SHARC+ 500 MHz	800	3	1
ADSP-SC587	1x Cortex-A5 500 MHz	2x SHARC+ 500 MHz	800	3	1
ADSP-SC589	1x Cortex-A5 500 MHz	2x SHARC+ 500 MHz	800	3	1
Texas Instruments					
Typ	Arm Kerne	DSP Kerne	DMIPS	GFLOPS	GMACS
OMAP-L138	1x Arm9 456 MHz	1x C674x 456 MHz	456	2,75	3,65
66AK2G12	1x Cortex-A15 1 GHz	1x C66x 1 GHz	3500	24	32
66AK2E05	4x Cortex-A15 1,4 GHz	1x C66x 1,4 GHz	19600	67,2	44,8
66AK2L06	2x Cortex-A15 1,2 GHz	4x C66x 1,2 GHz	8400	69,0	153,6
66AK2H14	4x Cortex-A15 1,4 GHz	8x C66x 1,2 GHz	19600	198,4	307,2
NXP					
Typ	Arm Kerne	DSP Kerne	DMIPS	GFLOPS	GMACS
i.MX RT600	1x Cortex-M33 300 MHz	1x Cadence Xtensa HiFi4 Audio 600 MHz	450	2,4	2,4

Tabelle 4.1: Ausgewählte ARM und DSP Multiprozessor-Ein-Chip-Systeme (Quelle in Anlehnung an ARM Limited 2011, Kap. 2 S. 9, Analog Devices Inc. 2020a, Texas Instruments 2020a, ARM Limited 2020 und *Tensilica HiFi Audio/Voice DSP IP* 2020)

5 Heterogenes Ein-Chip-System ADSP-SC582

In diesem Kapitel wird das zu untersuchende System und dessen Komponenten auf ihre Heterogenität analysiert.

5.1 Überblick

Basierend auf dem in Abbildung 1.1 gezeigten heterogenen System wird festgestellt, dass es sich laut der in Kapitel 2 erarbeiteten Theorie über heterogenes Rechnen um ein asymmetrisches Multiprozessorsystem handelt. Dies wird aufgrund der unterschiedlichen Prozessoren (ARM, SHARC+, FPGA und Hardware-Beschleuniger) begründet. Außerdem wird das System als eng gekoppeltes Multiprozessorsystem identifiziert, da die Prozessoren Zugriff auf mehrere gemeinsam geteilte Speicher (System L2 Memory und externer Speicher) haben.

5.1.1 ARM Cortex A5

Der ARM Cortex A5 ist der Prozessor mit den geringsten Stückkosten sowie Energieverbrauch der ARMv7 Prozessoren (vgl. Analog Devices Inc. 2017, Kap. 2 S. 1). Der im ADSP-SC58x integrierte A5 Prozessor enthält unter anderem einen eigenen Generic Interrupt-Controller (GIC), eine MMU und einen eigenen L1- sowie L2-Cache (vgl. Analog Devices Inc. 2017, Kap. 2 S. 1). Ein Zugriff auf den L2 Cache des ARM Kerns ist für den SHARC+ Kern möglich, hingegen dazu ist der L1 Cache des ARM Kerns für andere Prozessoren nicht zugänglich. Nennenswert dabei ist, dass keine Sicherheit hinsichtlich Datenkoheränz zwischen ARM und SHARC+ Kern besteht. (vgl. Analog Devices Inc. 2017, Kap. 2 S. 5) Bei den L2 Cache Zugriffen handelt es sich um NUCA, da der ARM Kern direkt auf diesen zugreift und dem SHARC+ Kern der Zugriff auf den L2 des ARM Kerns nur über eine Verbindungseinrichtung möglich ist.

5.1.2 SHARC+

Der SHARC+ ist ein Single-Instruction-Multiple-Data (SIMD) DSP von Analog Devices mit Super Harvard Architektur, der 32-bit/40-bit und 64-bit Gleitkommaoperationen unterstützt. Der Prozessor verfügt gegenüber seinen Vor-

gängern Verbesserungen hinsichtlich Cache Enhancement, Branch Prediction und weiteren Anpassungen des Befehlssatzes, ist aber trotzdem noch mit bisherigen Befehlssätzen kompatibel. (vgl. Analog Devices Inc. 2017, Kap. 1 S. 2) Im Kontrast zum ARM Kern verfügt der SHARC+ Kern über keinen eigenen L2 Cache. Allerdings verfügt er über einen L1 Cache, auf welchen andere Prozessoren und Controller zugreifen können. Bei Zugriffen auf den L1 Cache durch andere externe Prozessoren und Controller muss bei der Adressierung der Multiprocessor Offset berücksichtigt werden (siehe Abschnitt 6.3.2). Daraus resultiert, dass der SHARC+ Kern schneller auf seinen eigenen L1 Kern zugreifen kann, als andere Prozessoren und Controller, da diese wiederum über eine Verbindungseinrichtung auf den L1 des SHARC+ Kerns zugreifen müssen. Daher handelt es sich auch bei Zugriffen auf den Cache des SHARC+ Kerns um NUCA.

Dual Memory Support Keywords PM DM

Nennenswert für den Compiler des SHARC+ Kerns sind die zwei Schlüsselwörter PM und DM. Diese Spracherweiterung ermöglicht es, statische und globale Variablen für den Programm- oder Datenspeicher zu deklarieren. Dabei ist dies für automatische Variablen nicht möglich, da diese auf dem Stack hinterlegt werden, der sich immer im Datenspeicher befindet. Zudem werden String-Literale immer dem Datenspeicher zugeordnet. (vgl. Analog Devices Inc. 2019b, Kap. 2 S. 135) Durch diese Schlüsselwörter können die Vorteile der Harvard-Architektur genutzt werden. Beispielsweise kann so bei der Berechnung des Betrags einer komplexen Zahl der Realteil im Datenspeicher und der Imaginärteil im Programmspeicher gespeichert werden. Somit kann wie in Abschnitt 3.4.1 beschrieben, innerhalb eines Taktzyklus die komplexe Zahl gelesen und eine Operation darauf angewendet werden. Die Verwendung der Schlüsselwörter kann somit zur Optimierung von Programmen verwendet werden. Allerdings muss angemerkt werden, dass das Verwenden der Schlüsselwörter speziell bei der Verwendung von Zeigern weitere Schwierigkeiten einführt, die zu beachten sind (vgl. Analog Devices Inc. 2019b, Kap. 2 S. 137).

5.2 Programmiermodelle

Hinsichtlich der in Abschnitt 2.5 beschriebenen Programmiermodelle für Multiprozessorsysteme muss festgestellt werden, dass die ADSP-SC58x Produktreihe derzeit lediglich den MCAPI Standard unterstützt. Eine Parallelisierung von Aufgaben mittels Frameworks, wie dies beispielsweise durch OpenCL, OpenMP und HetroOMP für andere DSPs möglich ist, wird aktuell nicht unterstützt, obwohl das ADSP-S58x Ein-Chip-System aufgrund seiner Architektur dafür geeignet ist. Allerdings ist dies für den ADSP-SC852 nur teilweise der Fall,

da dieser nur heterogene Prozessoren enthält. Beispielsweise kann der L2 System Memory für Shared-Memory Programmierung verwendet werden, wie dies bereits bei der MCAPI Implementierung der Fall ist. Speziell für den ADSP-SC582 ist eine Parallelisierung der Aufgaben auf beide Prozessoren (ARM und SHARC) erstrebenswert, wie dies beispielsweise für andere Multiprozessor Ein-Chip-Systeme durch HetroOMP ermöglicht wird. Für die nächst größeren Prozessoren der Produktreihe, die einen zweiten SHARC+ Kern aufweisen, ist dies sogar für drei Prozessoren möglich.

Inter-Core-Communication

Letztendlich ist die bereitgestellte MCAPI Bibliothek ideal für die Kommunikation zwischen den einzelnen Prozessoren geeignet. Dabei bietet die MCAPI blockierende Funktionen sowie das zyklische Abfragen von Variablen (Polling), jedoch werden keine Rückruffunktionen unterstützt. Durch die blockierenden Funktionen beziehungsweise das Polling werden Taktzyklen benötigt, die ansonsten anderweitig eingesetzt werden können. Daher muss bedacht werden, dass durch die Verwendung der MCAPI eine wesentliche Reduzierung der Performanz in Kauf genommen werden muss. Eine Alternative zu MCAPI ist Memory-To-Memory-DMA (MDMA), der die schnellste Methode für die Inter-Core-Communication (ICC) ist. (vgl. Analog Devices Inc. 2015)

6 Fallbeispiel UART

Aufgrund der beschriebenen Methodik in Kapitel 1 wird in diesem Kapitel die UART-Schnittstelle als Fallbeispiel untersucht. Zum einen wurde die UART-Schnittstelle gewählt, damit zumindest ein Peripheriegerät des ADSP-SC582 betrachtet wird. Zum anderen bietet sich das Senden und Empfangen von Daten an, um einen Serialisierer und Deseralisierer zu verwenden. Durch die Serialisierung und Deserialisierung sowie die Übertragung der Daten mittels UART an ein anderes System kann somit überprüft werden, inwiefern eine kontrollierbare Heterogenität hinsichtlich der Zahlenformate und Datentypen besteht, wie dies in Unterabschnitt 2.4.2 thematisiert wird. Zudem wurde die UART-Schnittstelle gewählt, weil ein Datenaustausch zwischen eingebetteten Prozessoren wie Mikrocontrollern unentbehrlich ist. Im Bereich der eingebetteten Systeme werden die Daten in der Regel seriell übertragen, da so nur wenige Leitungen benötigt werden (vgl. Wüst 2011, S. 270). Dementsprechend sind viele Mikrocontroller mit einer seriellen Schnittstelle ausgestattet (vgl. Wüst 2011, S. 270) und falls dies nicht der Fall ist, kann mit zwei digitalen Ausgängen auch eine UART-Schnittstelle softwaremäßig implementiert werden. Ferner wird UART oft während der Entwicklung als Hilfestellung bei der Fehlersuche oder zum Bereitstellen von zusätzlichen Informationen eines Systems verwendet. (vgl. Wüst 2011, S. 270) Zudem lässt sich eine serielle Schnittstelle mit geringem Aufwand in Betrieb nehmen. Da beim ADSP-SC582 zwei Prozessoren die verfügbaren UART-Peripherien und auch unterschiedlichen Speicher verwenden können, soll im Zuge dieser Arbeit untersucht werden, welche Eigenheiten und Problematik bei der Verwendung einer UART-Schnittstelle mit dem ARM und SHARC+ Prozessor zu berücksichtigen sind.

6.1 Beschreibung des Fallbeispiels

Damit in Zukunft das in Abschnitt 1.1 beschriebene Prüfgerät über die UART-Peripherie ferngesteuert werden kann, soll im Zuge dieser Arbeit die Verwendung der UART-Schnittstelle untersucht werden. Dabei sollen die Daten in serialisierter Form gesendet und empfangen werden. Für dieses Fallbeispiel wird die UART-Schnittstelle verwendet, um explorativ auf neue Erkenntnisse zu stoßen, die es bei der Verwendung eines Multiprozessor Ein-Chip-Systems zu berücksichtigen gibt. Für diesen Zweck wurde für den ARM sowie SHARC+ Kern

ein UART-Treiber implementiert. Anschließend wurden beide Treiber getestet und zum Senden und Empfangen von serialisierten Daten verwendet. Für die Serialisierung und Deserialisierung der Daten wurde das MessagePack Format verwendet. MessagePack ist ein weitverbreitetes, schnelles und platzsparendes Serialisierungsformat (vgl. Furuhashi 2020a), das für Implementierungen im Bereich von eingebetteten Systemen geeignet ist. Im Rahmen dieser Arbeit wird eine bereits bestehende Implementierung (siehe Gunyon 2020) verwendet, die nur minimal adaptiert wurde. Die Spezifikation des MessagePack Formats und dessen Beschreibung kann aus folgender Quelle entnommen werden: Furuhashi 2020b. Weiterführend wurde der bestehende Code des ARM Kerns erweitert, sodass dieser nun über die `UART0` Schnittstelle gesteuert werden kann. Dabei werden an den ARM Kern serialisierte Daten gesendet und von diesem deseralisiert und ausgewertet. Beispielsweise können nun so mittels PC neue Zielwerte direkt an den ARM Kern gesendet werden, der wiederum diese an den FPGA weiterleitet. Bei der Verwendung der MessagePack Implementierung von Gunyon 2020 und der Serialisierung der Daten wurden weitere Erkenntnisse hinsichtlich der unterschiedlichen Datentypen und deren Darstellung erarbeitet, die es bei der Programmierung des Ein-Chip-Systems zu berücksichtigen gibt (siehe Abschnitt 6.3). Weitere Erkenntnisse wurden durch die Implementierung der UART-Treiber und der Analyse der grundsätzlichen Verwendung der UART-Peripherie (siehe Abschnitt 6.2) erbracht, die weiterführend in Abschnitt 6.3.2 thematisiert werden.

6.2 Grundsätzliche Verwendung der UART-Peripherie

Der ADSP-SC582 bietet drei gewöhnliche DMA-fähige UART-Schnittstellen. Diese verfügen über jeweils zwei separate DMA-Kanäle zum Senden und Empfangen von seriellen Daten. Alternativ können die UART-Schnittstellen auch mittels Interrupts oder Polling verwendet werden. Im Zuge dieser Arbeit wurde für die UART-Schnittstelle `UART0` die Verwendung von Polling, Interrupts und DMA implementiert, um mögliche Problemstellungen zu identifizieren, die für heterogenes Rechnen von Relevanz sind. Infolgedessen wird die jeweilige Verwendung der UART-Peripherie beschrieben und auf mögliche Problemstellungen hingewiesen.

Polling

Im Polling Betrieb muss vor dem Senden der Daten das `THRE` Bit des `UART0_STAT` Register überprüft werden. Ist dieses gesetzt, können in das `UART0_THR` Register die Daten zum Senden geschrieben werden. Für das Lesen von Daten muss zuerst das `DR` Bit des `UART0_STAT` Register überprüft werden. Wenn dieses gesetzt

ist, können die Daten aus dem `UART0_RBR` Register gelesen werden. (vgl. Analog Devices Inc. 2017, Kap. 17 S. 23) Dabei ist dies eine gebräuchliche Umsetzung, die für den ARM und SHARC+ Kern in gleicherweise verwendet werden kann.

DMA

Damit beispielsweise Messdaten der Verstärkerplatine effizient versendet werden können, wurde zum Senden von großen Datenmengen eine DMA Funktion implementiert, die einen DMA Controller konfiguriert, sodass dieser einmalig alle Daten eines Puffers an das `THR` Register der `UART0` Peripherie weiterleitet. Dadurch wird nur ein minimaler Aufwand zum Aufsetzen des DMAs benötigt und somit die CPU entlastet. So können große Datenmengen effizient versendet werden. Beispielhaft zeigt folgender Code des SHARC+ Kerns, die Konfiguration des DMA Controllers für die UART0 Peripherie zum Senden von Daten mittels DMA. Zu Beginn wird der DMA für den richtigen Modus, die Pufferlänge, sowie die Größe der zu übertragenden Elemente des Puffers konfiguriert. Erwähnenswert ist die IF-Anweisung, welche die Adresse des Puffers auf dessen Speicherbereich prüft. Liegt der Puffer im L1 Speicher des SHARC+ Kerns wird anhand dieser IF-Anweisung ein Versatz des Adressbereichs berücksichtigt. Dies muss aufgrund einer Problemstellung durchgeführt werden, die in Unterabschnitt 6.3.2 detailliert beschrieben wird.

```
1  void Uart0_Tx_DMA_StopMode(uint64_t *data, uint32_t len){
2    *pREG_DMA20_CFG = (ENUM_DMA_CFG_READ | ENUM_DMA_CFG_MSIZE01 |
       ENUM_DMA_CFG_PSIZE01 | ENUM_DMA_CFG_STOP );
3    *pREG_DMA20_XCNT = len;
4    *pREG_DMA20_XMOD = 1;
5    if(data >=  L1_BLOCK_0_START  && data < L1_BLOCK_3_END){
6      *pREG_DMA20_ADDRSTART = ADD_MULTIPROCESSOR_OFFSET_PORT1(data);
7    }
8    else{
9      *pREG_DMA20_ADDRSTART = data;
10   }
11   *pREG_DMA20_CFG = ENUM_DMA_CFG_EN | ENUM_DMA_CFG_XCNT_INT ;
12 }
```

Abschließend wird der DMA und dessen Interrupt aktiviert, der ausgelöst wird, wenn die Übertragung abgeschlossen ist.

Interrupts

Für das Einlesen der eingehenden Daten wurde ein Interrupt-Handler implementiert. Da eingehende Daten wie beispielsweise Befehle an die Verstärkerplatine über die UART-Schnittstelle asynchron stattfinden, ist für dies die Verwendung von Interrupts ideal geeignet. (vgl. Wüst 2011, S. 112) Folgender Codeausschnitt zeigt die die Interrupt Funktion die zum Empfangen der UART Eingangsdaten verwendet wird. Bei der Funktion `UART0_RX_Handler()` handelt es sich um den Interrupt Handler, der zuerst das Interrupt-Flag zurücksetzt

und anschließend die Eingangsdaten des `UART0_RBR` Register in einem Ringspeicher speichert. Je nach Prozessor und Adresse des Puffers (`uartBuf`) muss der Multiprozessor Offset berücksichtigt werden. Daher muss die Implementierung des Interrupt-Handlers gegebenenfalls angepasst werden. Der Multiprozessor Offset wird in Unterabschnitt 6.3.2 erläutert.

```
1 void UART0_RX_Handler() {
2   *pREG_DMA21_STAT |= ENUM_DMA_STAT_IRQDONE;
3   circBuf_push(&uartBuf, *pREG_UART0_RBR);
4 }
```

6.3 Erkenntnisse hinsichtlich des SHARC+ Kerns

Im Folgenden werden die Erkenntnisse erläutert, die bei der Umsetzung der UART-Schnittstelle in Kombination mit der MessagePack Serialisierung für den SHARC+ Prozessor erarbeitet wurden.

6.3.1 Datentypen

Während der Portierung und Verwendung des MessagePack-Formats wurde festgestellt, dass für die vollständige Unterstützung der in MessagePack definierten Datentypen die bereits in Abschnitt 2.4.2 thematisierte Problematik hinsichtlich des Datentyps Double von Relevanz ist. Aber auch die Größe eines Characters oder die unterschiedliche Darstellung des Datentyps Float muss thematisiert werden.

Einfache und doppelte Genauigkeit von Gleitkommazahlen

Die Definition des Datentyps Double mittels Compileroption in einfacher Genauigkeit hat die Auswirkung, dass die verwendete MessagePack Implementierung das MessagePack-Format nicht vollständig implementiert. Da der Datentyp Double mit doppelter Genauigkeit nicht unterstützt wird. Daher wurde mittels Compileroption der Datentyp Double mit doppelter Genauigkeit definiert. Dies resultiert allerdings in einem Linker-Fehler, wenn der FFT Hardware-Beschleuniger ebenfalls verwendet wird. Der Grund dafür ist, dass die proprietäre Bibliothek des FFTAs mit der einfachen Genauigkeit des Datentyps Double kompiliert wurde. Damit dieses Problem behoben werden kann, bedarf es einer zusätzlichen Compileroption: `-double-size-any`. Dadurch können Dateien, die mit dieser Option kompiliert wurden, mit anderen Dateien verknüpft werden, die mit `-double-size-32` oder `-double-size-64` kompiliert wurden. Allerdings muss beachtet werden, dass so die Typprüfung hinsichtlich des Wertebereichs für den Datentyp Double aufgehoben wird. Wird eine Konformität mit ISO/IEC-9899: 1990 C standard, ISO/IEC 9899: 1999 C standard oder mit ISO/IEC 14882: 2003 C++ benötigt, so muss die Compileroption für die

doppelte Genauigkeit des Datentyps Double ausgewählt werden. (vgl. Analog Devices Inc. 2019b, Kap. 2 - Data Type and Data Type Sizes)

Unterschiedliche Größen des Datentyps Character

Darüber hinaus kann mit der Compileroption `-char-size[-8|-32]` die Größe eines Characters definiert werden, sofern der Prozessor die Byte-Addressierung unterstützt. Dadurch wird für die Option `-char-size-8` die Größe eines Characters als acht Bit und die Größe des Datentyps Short als 16 Bit definiert. Somit wird bei der Option `-char-size-32` ein Character mit einer Größe von 32 Bit definiert. Zum einen kann mit Charactern in der Größe von 32 Bit die Kompatibilität zu existierenden SHARC Applikationen sichergestellt werden, die für früher Generationen von SHARC Prozessoren entwickelt wurden und keine Unterstützung der Byte-Adressierung hatten. Zum anderen bietet die `-char-size-8` Option Kompatibilität, zu Applikationen die in C/C++ entworfen wurden, sofern ein Character bei diesen Applikationen acht Bit entsprechen. Dabei wird standardmäßig für den ADSP-SC58x die Byte-Adressierung verwendet. Allerdings impliziert die Änderung der Größe eines Characters mit der Option `-char-size[-8|-32]` noch weitgehende Änderungen, die aus folgender Auflistung zu entnehmen sind.

- **Byte-Reihenfolge**
  In Kombination mit der Compileroption `-double-size-64` wird die Byte-Reihenfolge der Datentypen Long Long, Unsigned Long Long, Long Double und Double geändert, wenn die Größe eines Character mithilfe der `-char-size[-8|-32]` Option geändert wird.
  Für den Fall, dass die `-char-size-32` Option ausgewählt ist, wird für die genannten Datentypen das Big-Endian Format verwendet. Im Falle der `-char-size-8` Option werden die Datentypen in Little-Endian gespeichert. Beispielhaft ist in Tabelle 6.1 eine 64 Bit Variable für die zwei unterschiedlichen Byte-Reihenfolgen dargestellt. Dies wird aufgrund der Kompatibilität des SHARC+ Prozessors zu anderen Prozessoren und Geräten so gehandhabt.

- **Zeiger**
  Im Falle der `char-size-8` Option werden alle Zeiger dem Byte-Addressraum zugeordnet. Dies gilt auch für den Stack- und Frame-Pointer sowie für alle Adressen von globalen Variablen und Datenpuffer.

- **Symbolnamen**
  Da die primitiven Datentypen und ihre Zeiger sich im Code je nach Adressierungsart in ihrer Größe und Darstellung unterscheiden, muss ein

Long Long Variable	-char-size-8 Adresse: Wert	-char-size-32 Adresse: Wert
0x1122334455667788LL;	0x2c0000: 0x88 0x2c0001: 0x77 0x2c0002: 0x66 0x2c0003: 0x55 0x2c0004: 0x44 0x2c0005: 0x33 0x2c0006: 0x22 0x2c0007: 0x11	0xb0000: 0x11223344 0xb0001: 0x55667788

Tabelle 6.1: Byte-Reihenfolge einer 64-Bit Variable für die zwei Compileroptionen `-char-size[-8|-32]` (Analog Devices Inc. 2019b, Kap.2 S.69)

Kreuzen der unterschiedlichen Addressierungsmethoden (Byte- vs. Word-Addressed) im Code unterbunden werden. Dies wird erreicht, indem für Code, der mit der Compileroption `char-size-8` kompiliert wurde, ein Unterstrich als Präfix dem Symbolname hinzugefügt wird. Für den Fall der `char-size-32` Option wird ein Punkt als Postfix angehängt.

Applikationen mit 8 Bit und 32 Bit Character

Angesichts der unterschiedlichen Byte-Reihenfolgen ist es möglich, mittels Compiler-Anweisungen die Größe eines Characters für unterschiedliche Abschnitte eines Programms oder für einen gesamten Quellcode zu bestimmen. Mit den Pragmas `#pragma word_addressed` und `#pragma byte_addressed` können Symbole im Quellcode markiert werden, um zu kennzeichnen, mit welcher Einstellung diese kompiliert wurden. Eine weitere Möglichkeit ist es, Regionen mit Push und Pop zu definieren, um zu kennzeichnen, dass diese Region mit der anderen Character Größe kompiliert werden soll. Beispielsweise ist dies im Folgenden Code-Ausschnitt zu erkennen, für den Fall, dass dieser mit der Compileroption `-char-size-8` kompiliert wird.

```
1 #pragma word_addressed(push)
2 // function prototype for function built with -char-size-32
3 extern int my_char_size_32_func(void);
4 // function prototype for another function built with -char-size-32
5 extern void another_char_size_32_func(int a);
6 #pragma word_addressed(pop)
7 // function prototype for function built with same char-size as
8 // current compilation
9 int my_char_size_8_func(void);
```

Demzufolge interpretiert der Compiler den mit Push und Pop gekennzeichneten Abschnitt, in der markierten Character Größe, die für diesen Fall 32 Bit entspricht. Aufgrund dessen ändern sich die Byte-Reihenfolge, Größe der Datentypen und Symbole für diesen Abschnitt, wie in der vorhergehenden Auflistung beschrieben. (vgl. Analog Devices Inc. 2019b, Kap. 2 - S. 59)

Dezimalzahlen

Im Folgenden wird auf die Unterstützung von Gleit- und Festkommazahlen des ADSP-SC582 eingegangen.

Gleitkommazahlen Abgesehen von den Implikationen des SHARC+ hinsichtlich der Darstellung von Gleitkommazahlen in Abschnitt 6.3.1, werden auch 16 Bit Gleitkommazahlen unterstützt. Durch dies können doppelt so viele Gleitkommazahlen gespeichert werden. Der Prozessor unterstützt die Konvertierung von 32 Bit Gleitkommazahlen auf 16 Bit Gleitkommazahlen und umgekehrt. Dabei werden für beide Konvertierungen eigene Befehle unterstützt, die innerhalb eines Taktzyklus durchgeführt werden. (vgl. Analog Devices Inc. 2019a, Kap. 3 S. 19 und Analog Devices Inc. 2019b, Kap. 3. S. 13) 16 Bit Gleitkommazahlen werden von MessagePack standardmäßig nicht unterstützt (vgl. Furuhashi 2020b).

Festkommazahlen Zusätzlich zu den Gleitkommazahlen unterstützt der SHARC+ mittels dedizierter Hardware 32 Bit Festkommazahlen. Allerdings kann die Semantik der arithmetischen Operationen zur Verwendung der verfügbaren Hardware in der Programmiersprache C nicht leicht abgebildet werden. Daher unterstützt der Compiler für den SHARC+ Prozessor primitive Datentypen die das Verwenden von Festkommazahlen erleichtert, ohne intrinsische Funktionen des Compilers oder integrierter Assemblercode zu benötigen. Zudem wird ein Teil des Extensions to support embedded processors ISO/IEC Technical Report 18037 (*ISO/IEC TR 18037:2008(en), Programming languages — C — Extensions to support embedded processors* 2020) unterstützt, der unter anderem die Handhabung von Festkommazahlen in der Sprache C standardisiert. (vgl. Analog Devices Inc. 2019c, Kap. 2 S. 85)

6.3.2 Multiprocessor Space und O□set

Im Folgenden wird beschrieben, welche Aspekte bei einem Speicherzugriff auf den internen Speicherbereich des SHARC+ Kerns beachtet werden muss. In Tabelle 9.1 sind die einzelnen Speicherabschnitte zu erkennen. Dabei sind die L1 Blöcke im SHARC+ Space nur für den SHARC+ Kern selbst über den angegebenen Adressbereich zugänglich. Andere Prozessoren, Peripherie-Geräte

und Controller können auf den L1 Speicher des jeweiligen SHARC+ Kerns über den Multiprocessor Byte Address Space zugreifen (siehe folgende Abbildung).

	Memory Block	Byte Address Space for Cortex-A5 and SHARC+	Normal Word Address Space for SHARC+
Address via Slave Port 1	Block 0	0x28240000-0x2826FFFF	0x0A090000-0x0A09BFFF
	Block 1	0x282C0000-0x282EFFFF	0x0A0B0000-0x0A0BBFFF
	Block 2	0x28300000-0x2831FFFF	0x0A0C0000-0x0A0C7FFF
	Block 3	0x28380000-0x2839FFFF	0x0A0E0000-0x0A0E7FFF
Address via Slave Port 2	Block 0	0x28640000-0x2866FFFF	0x0A190000-0x0A19BFFF
	Block 1	0x286C0000-0x286EFFFF	0x0A1B0000-0x0A1BBFFF
	Block 2	0x28780000-0x2871FFFF	0x0A1C0000-0x0A1C7FFF
	Block 3	0x28780000-0x2879FFFF	0x0A1E0000-0x0A1E7FFF

Tabelle 6.2: Unterschiedliche Addressierungsbereiche für den L1 des SHARC+ Kerns 1
(Analog Devices Inc. 2020b)

Tabelle 6.2 beschreibt zwei Möglichkeiten zur Adressierung des L1 Speichers, weitere Adressierungsarten sind für den SHARC+ Kern möglich. Dabei ist beim Vergleich der Adressen des Byte Address-Space aus Tabelle 6.2 mit dem SHARC+ Space aus Tabelle 9.1 zu erkennen, dass die Adressen lediglich einen Versatz zueinander aufweisen. Damit auf den L1 Speicher eines SHARC+ Kerns andere Prozessoren, Controller und Peripherie-Bausteine zugreifen können, muss daher ein Multiprozessor Offset zu der privaten Adresse des SHARC+ Kerns hinzugefügt werden. Für die Produktreihe des ADSP-SC58x sind für den ersten SHARC+ Kern folgende Offsets definiert.

- SHARC1 Slave Port 1: 0x28000000
- SHARC1 Slave Port 2: 0x28400000

Dabei stehen zur Adressierung des L1 jeweils zwei Slave Ports zur Verfügung. Beispielsweise kann auf die Adresse 0x00240000 über den Slave Port 1, durch hinzufügen des Offsets 0x28000000 mittels Oder-Verknüpfung, zugegriffen werden. Gleichung 6.1 zeigt beispielhaft das Hinzufügen eines Offsets. (vgl. Analog Devices Inc. 2020b)

$$(0x28000000|0x00240000) = 0x28240000 \tag{6.1}$$

Den Multiprozessor Offset müssen Entwicklende bewusst bei der Entwicklung berücksichtigen, wenn die Daten im L1 des SHARC+ Kerns gespeichert sind und von externen Komponenten verwendet werden, da eine Nichtberücksichtigung zu ungültigen Speicherzugriffen führt. Eine Möglichkeit ist, mithilfe von einer IF-Anweisung den Adressbereich zu prüfen und gegebenenfalls den Offset hinzuzufügen, wie dies in folgendem Beispiel in Zeile 14 und 15 gezeigt wird. Das Beispiel zeigt die Konfiguration eines DMA Auto Buffer für die UART0 Peripherie, der die empfangen Daten in einem Puffer speichert, dieser wird durch den DMA als Ringspeicher verwendet.

```
1  /* Macros and defines for Multiprocessor Offset */
2  #define OFFSET_PORT1                  0x28000000
3  #define L1_BLOCK_0_START              (void*)0x00240000
4  #define L1_BLOCK_3_END                (void*)0x00400000
5  #define ADD_MULTIPROCESSOR_OFFSET_PORT1(addr)(OFFSET_PORT1+(void*)addr)
6
7  void URA_Uart0_DMA_Rx_Auto_Buffer(uint64_t *data, uint32_t len) {
8    // Configure RX DMA
9    *pREG_DMA21_CFG = (ENUM_DMA_CFG_WRITE | ENUM_DMA_CFG_MSIZE01 |
       ENUM_DMA_CFG_PSIZE01 | ENUM_DMA_CFG_SYNC | ENUM_DMA_CFG_AUTO);
10   *pREG_DMA21_XCNT = len;
11   *pREG_DMA21_XMOD = 1;
12   if(data >=  L1_BLOCK_0_START  && data < L1_BLOCK_3_END){
13     *pREG_DMA20_ADDRSTART = ADD_MULTIPROCESSOR_OFFSET_PORT1(data);
14   }
15   else{
16     *pREG_DMA20_ADDRSTART = (void*) data;
17   }
18 }
```

Dabei gilt zu beachten, dass beim Debugging mit der Eclipse basierten Entwicklungsumgebung CrossCore® Embedded Studio von Analog Devices, der Memory Browser beide Speicherbereiche (SHARC+ und MULTIPROCESSOR Space) darstellen kann (siehe Abbildung 6.1). Dies kann zu Missverständnissen beim Debugging führen, da je nach Art des Speicherzugriffs die jeweilige Darstellung im Memory Browser verwendet werden muss. Eine Alternative zum jeweiligen Hinzufügen des Multiprozessor Offset im Einzelfall, ist ein ständiges Hinzufügen des Offsets, unabhängig davon, ob auf die L1 Daten des SHARC+ Kerns von intern oder extern zugegriffen wird. Allerdings hat das einen starken Einfluss auf die Performanz (siehe Abschnitt 6.3.2). Auch muss beachtet werden, dass Analog Devices bei anderen Komponenten wie beispielsweise dem FFT Hardware-Beschleuniger beim Verwenden der proprietären Bibliotheken die Adressen der Funktionsparameter prüft und gegebenenfalls den Multiprozessor Offset hinzufügt. Zudem muss für den ARM Kern kein Multiprozessor Offset berücksichtigt werden, dies wird durch die Memory Management Unit (MMU) gehandhabt (siehe Abschnitt 6.4).

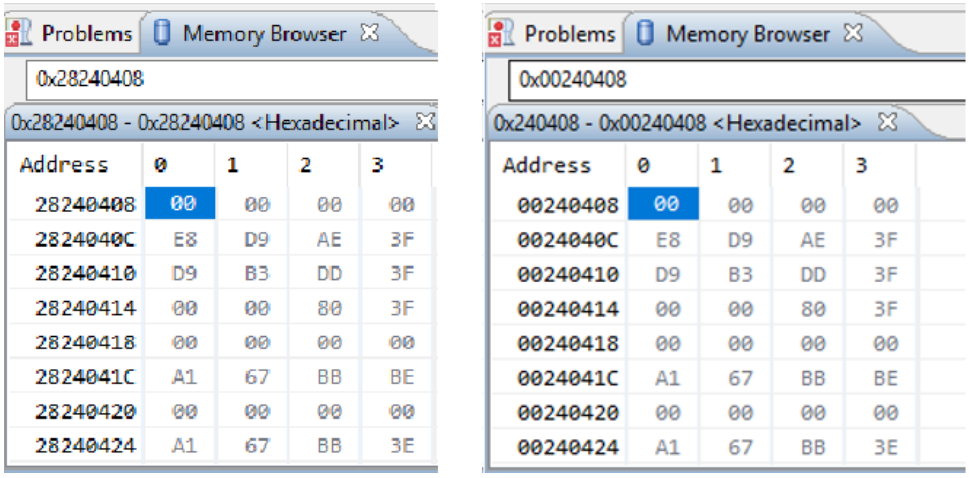


Abbildung 6.1: CrossCore® Embedded Studio Screenshot zweier Memory Browser Ansichten von unterschiedlichen Address Spaces

Vergleich Speicherzugri System Space vs. Multiprocessor Space

In folgender Tabelle 6.3 ist der oben beschriebene Einfluss auf die Performanz des SHARC+ Kerns zu erkennen, falls dieser über die Verbindungseinrichtung auf seinen eigenen Speicher zugreift. Ein ständiges Hinzufügen des Multiprozessor Offsets ist daher nicht zu empfehlen.

Speicherbereich	Taktzyklen für Lesezugriff	Taktzyklen für Schreibzugriff
SHARC Space	8	6
Multiprocessor Space	48	19

Tabelle 6.3: Speicherzugriff Multiprocessor Space versus SHARC Space des SHARC-Kerns

6.3.3 Linker-Description File

Für die SHARC Prozessoren wird es ermöglicht, Daten oder Code einem bestimmten Speicherbereich zuzuweisen. Dazu muss das Linker Description File (LDF) konfiguriert werden, sodass dieses das verwendete Speicherlayout abbildet. Anschließend kann mittels Kennzeichner eine Funktion oder Variable einem Speicherbereich zugewiesen werden. (vgl. Analog Devices Inc. 2015) Im Zuge dieser Arbeit wurde das Linker Description File des ADSP-SC582 angepasst, sodass dieses das in Tabelle 9.1 gezeigte Speicherlayout abbildet. Dies wird im Zuge dieser Arbeit benötigt, um die Ein- und Ausgangspuffer der UART-

Schnittstelle aber auch die Ein- und Ausgangsdaten der FFT (siehe Kapitel 7) in bestimmten Speicherbereichen zu hinterlegen.

6.4 Erkenntnisse hinsichtlich des ARM Kerns

Auch der ARM Kern des ADSP-SC58x Systems verfügt über einen eigenen Adressbereich. Möchte dieser beispielsweise auf den L1 Speicher des SHARC+ Kerns zugreifen, so muss die MMU des ARM Kerns richtig konfiguriert werden. Dies erfolgt durch adaptieren der `abstract_page_table.c` Quelldatei, die bei Projektvorlagen des ADSP-SC58x enthalten ist. (vgl. Analog Devices Inc. 2020b) Der Vorteil dabei ist, dass die Entwickelnden keinen Offset für die unterschiedlichen Adressbereiche berücksichtigen müssen. Somit kann festgestellt werden, dass hinsichtlich der Adressierung und Konfiguration der Speicherbereiche eine Heterogenität herrscht, welche die Problematik der Programmierung dieses Systems erhöht.

7 Fallbeispiel FFT

Aufgrund der in Kapitel 1 definierten Zielsetzung wird das bestehende System auf dessen Performanz hinsichtlich des verwendeten Speichers und der FFT Berechnung untersucht. In Abschnitt 7.1 wird das Fallbeispiel und das grundlegende Prinzip der Implementierungen beschrieben. Anschließend werden der Hardware-Beschleuniger und die verfügbaren Bibliotheken zur Berechnung der FFT erläutert. Infolgedessen werden die Zugriffszeiten des SHARC+ Kerns auf die einzelnen Speicher (siehe Unterabschnitt 7.3.1) analysiert. Danach werden die Performanz der jeweiligen FFT Bibliotheken und dem FFT Hardware-Beschleuniger analysiert (siehe Unterabschnitt 7.3.2).

7.1 Beschreibung des Fallbeispiels

Im Fallbeispiel FFT werden unterschiedliche Funktionen zur Berechnung der FFT analysiert, die auf dem SHARC+ Kern des ADSP-SC582 ausgeführt werden. Dabei wird die Performanz der Implementierungen und die des Hardware-Beschleunigers des ADSP-SC582 gegenübergestellt. Die unterschiedlichen FFT Funktionen sowie der Hardware-Beschleuniger sind in Abschnitt 7.2 beschrieben. Damit die Performanz der Implementierung bewertet werden kann, wird für die jeweilige FFT Berechnung die benötigte Anzahl an Taktzyklen gemessen. Die angegebenen Taktzyklen in Abschnitt 7.3 beziehen sich immer auf die des SHARC+ Kerns. Da die Implementierungen und der Hardware-Beschleuniger für unterschiedliche Anwendungsfälle optimiert sind, werden die FFT Berechnungen für verschiedene FFT Größen durchgeführt. Da die Performanz auch durch den verwendeten Speicher beeinflusst wird, ist dies im Zuge dieser Untersuchung zu berücksichtigen. Somit werden die Experimente jeweils mit den unterschiedlichen Speichern (L1, L2 System Memory und L3) durchgeführt. Die Ein- und Ausgangsdaten der FFT sowie zusätzlich benötigte Daten werden mithilfe von Bezeichner dem jeweiligen Speicher zugewiesen. Daher werden im Zuge dieses Fallbeispiels in Abschnitt 7.3 zuerst die Speicherzugriffszeiten des SHARC+ Kerns untersucht. Damit wird eine erste Beurteilung der Speicherhierachie hinsichtlich deren Zugriffszeiten ermöglicht. Für jedes FFT Experiment wird das gleiche zeitdiskrete Grundsignal als Eingangsvektor verwendet, das in Abbildung 7.1 dargestellt ist.

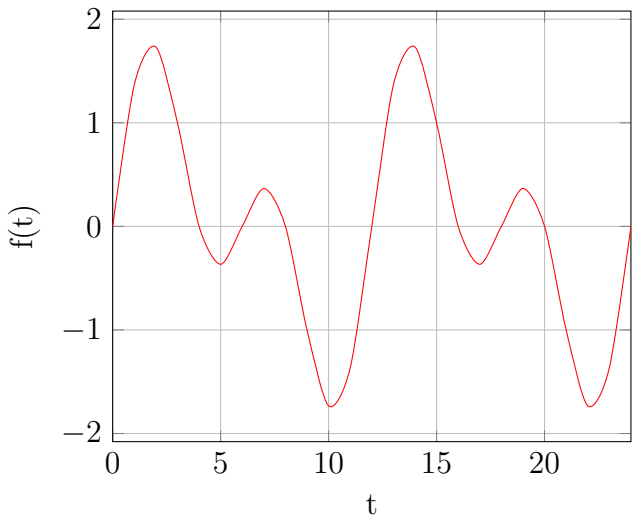


Abbildung 7.1: FFT Testsignal

Je nach Größe der FFT eines Experiments wird das Grundsignal in Abbildung 7.1 dementsprechend oft aneinandergehängt und so der benötigte Eingangsvektor erzeugt. Beispielhaft wird in folgendem vereinfachten Codeausschnitt die grundlegende Funktionsweise der Implementierungen dargestellt. Dabei ist am Beginn zu erkennen, dass mittels Bezeichner die Ein- und Ausgangsdaten sowie eine weitere Variable zur Berechnung des Mittelwerts dem L1 Speicher zugewiesen werden. Die `measure_TransFFT_Performance64()` führt die Berechnung der FFT für eine Größe von 64 Abtastwerten durch. Dabei wird die `rfft64(...)` Funktion der `trans.h` Bibliothek verwendet, die in Abschnitt 7.2 beschrieben wird. Vor dem Funktionsaufruf der `rfft64(...)`, werden die Eingangsdaten der FFT initalisiert und anschließend wird die Messung der Taktzyklen gestartet. Nach der Berechnung der FFT wir die Messung der Taktzyklen gestoppt.

```
#define __data_L1__          __attribute__ ((section("seg_int_data")))
__data_L1__ float fftInput[MAX_FFT_SIZE];
__data_L1__ float realOutput[MAX_FFT_SIZE];
__data_L1__ float imagOutput[MAX_FFT_SIZE];
__data_L1__ volatile double calc_avg;

void measure_TransFFT_Performance64(){
    for(int j = 0; j < TEST_LENGTH; j ++){
      init_FFTData(fftInput, baseCurve, FFT_SIZE_64);
      START_CYCLE_COUNT(...);
      rfft64(fftInput, realOutput, imagOutput);
      STOP_CYCLE_COUNT(...);
    }
    calc_avg /= TEST_LENGTH;
  }
}
```

Die Messung der Performanz wird im Zuge dieser Arbeit zehnmal wiederholt und aus den Resultaten der arithmetische Mittelwert gebildet, wie dies auch in Abschnitt 7.3 beschrieben wird.

7.2 FFT Hardware-Beschleuniger und FFT Bibliotheken

Im Folgenden werden der Hardware-Beschleuniger und die verfügbaren Bibliotheken von Analog Devices erläutert, die zur Berechnung der FFT und Analyse der Performanz verwendet werden. Zusätzlich zu den verfügbaren Bibliotheken von Analog Devices wird noch eine quelloffene Bibliothek (siehe Unterabschnitt 7.2.3) zum Vergleich verwendet, damit die Vorteile von optimierten Bibliotheken und Hardware-Beschleunigern veranschaulicht werden.

7.2.1 FFT Hardware-Beschleuniger

Der FFT Hardware-Beschleuniger (FFTA) des ADSP-SC58x erlaubt das selbständige Durchführen einer FFT beziehungsweise einer inversen FFT für Gleitkommazahlen in einfacher Genauigkeit nach IEEE-754/854, sodass es an keinem zusätzlichen Aufwand durch andere Prozessoren bedarf. Die Anzahl der Abtastwerte für die FFT muss einer 2er Potenz im Bereich von 64 bis 4194304 Werten entsprechen. Mittels dem FFTA können kleine und große FFTs durchgeführt werden. Dabei kann der Hardware-Beschleuniger für kleine FFTs im Bereich von 64 bis 2048 Abtastwerten sowie für große FFTs im Bereich von 4096 bis 4194304 Abtastwerten verwendet werden. Allerdings werden große FFTs nur eingeschränkt unterstützt. Aufgrund des geringen internen Speichers des Hardware-Beschleunigers kann dieser lediglich die FFT für 2048 Abtastwerte durchführen. Für größere FFTs muss diese in mehreren Phasen durchgeführt werden, dabei kommt ein Teile-und-Herrsche-Verfahren zur Anwendung. Zudem kann der Hardware-Beschleuniger automatisch Eingangswerte skalieren. Schnelles Laden und Ablegen von Daten mit einer Breite von 64-Bit mittels High-Speed DMA von internem wie externem Speicher und unterschiedliche Taktfrequenzen zur Reduzierung des Energieverbrauchs werden unterstützt (vgl. Analog Devices Inc. 2017, Kap. 50). Die Taktfrequenz des FFTA kann maximal dem SYSCLK entsprechen, der mit 250 MHz begrenzt ist und nicht größer als die Core Clock Frequency (f_{CCLK}) sein darf. Eine Programmierung auf Registerebene wird für den Hardware-Beschleuniger nicht unterstützt. Bei der Verwendung des FFTAs sind Entwickelnde auf eine proprietäre Bibliothek von Analog Devices angewiesen, die durch das CrossCore® Embedded Studio bereitgestellt wird (vgl. Analog Devices Inc. 2018, passim und Analog Devices Inc. 2017, Kap. 50). Folgende Betriebsarten können mit dem FFTA bearbeitet werden.

- Single-Shot FFT
- Pipelined FFT

Dabei wird für die Single-Shot FFT eine blockierende Funktion von Analog Devices aufgerufen. Diese verwendet den Hardware-Beschleuniger zur einmaligen Durchführung einer Transformation der Eingangsdaten. Im Unterschied dazu wird die Pipelined FFT verwendet, wenn dieselbe Transformation auf mehrere Datensätze angewendet wird. Der Vorteil dabei ist, dass nur einmal der FFT Hardware-Beschleuniger konfiguriert werden muss und anschließend kann die Transformation für mehrere Datensätze durchgeführt werden, so wird der Mehraufwand der Konfiguration reduziert. Allerdings muss die Pipe zu Beginn mit Dummy-Eingangswerten konfiguriert werden, falls noch keine Daten zur Verfügung stehen. Durch das Verwenden von Triggern kann die Pipe auch kontinuierlich verwendet werden, sodass es keiner Interaktion eines anderen Prozessors bedarf (vgl. Analog Devices Inc. 2017, passim und Analog Devices Inc. 2019c, Kap. 3 S. 13)

7.2.2 Analog Devices Bibliotheken

Folgende Bibliotheken von Analog Devices stehen zur Berechnung der FFT in Assemblersprache zur Verfügung. Nennenswert dabei ist, dass die Header-Dateien `filter.h` sowie `trans.h` teilweise dieselben Funktionsnamen deklarieren. Allerdings sind die Implementierungen in der Header-Datei `filter.h` für SIMD optimiert, dies ist für die `trans.h` Bibliothek nicht der Fall. (vgl. Analog Devices Inc. 2019c, Kap 3. S. 6)

filter.h

Die filter.h Header-Datei stellt Funktionen für die Signalverarbeitung zur Verfügung. Die Bibliothek enthält Funktionen für Filter, aber auch Funktionen zur Berechnung der FFT. Die Datei enthält drei unterschiedliche Gruppen an Funktionen, diese können wie folgt unterteilt werden.

Gruppe 1 Funktionen für SHARC+ Prozessoren mit SIMD (bspw. rfftN)

Gruppe 2 Funktionen für alle ADSP-21xxx Prozessoren (bspw. rftt)

Gruppe 3 Funktionen für ADSP-21xxx SIMD Plattformen (bspw. rfftf_2)

Gruppe 1 stellt Funktionen für SHARC+ Prozessoren bereit, die den geringsten bedarf an Speicherplatz haben, jedoch wird dies durch eine reduzierte Flexibilität und Performanz erzielt. Dabei wird für jede Zweierpotenz im Bereich von 16 bis 65536 eine separate FFT Funktion bereitgestellt. Beispielsweise muss für

eine FFT mit 64 Abtastwerten die Funktion `rfft64(...)` verwendet werden. Infolgedessen werden bei Applikationen mit unterschiedlichen FFT Größen auch unterschiedliche Funktionen benötigt, die für die jeweilige Abtastraten geeignet sind. Die Reduzierung des Speicherplatzes erfolgt durch das Wiederverwenden des Speicherplatzes der Eingangsdaten als temporär Speicher für die Berechnungen.

Gruppe 2 bietet Funktionen für alle ADSP-21xxx Prozessoren und somit keine SIMD Unterstützung. Im Gegensatz zu den Funktionen in Gruppe 1 kann bei den Funktionen in Gruppe 2 die Größe der FFT als Funktionsparameter übergeben werden. Der Vorteil dabei ist eine geringere Code-Größe sowie bessere Performanz für unterschiedliche Abtastraten, jedoch bedarf es an mehr Speicherplatz, da der hinterlegte Algorithmus zusätzlichen Speicher für die temporären Berechnungsdaten benötigt.

Gruppe 3 stellt stark optimierte Funktionen bereit, die nur für die ADSP-21xxx SIMD Plattformen verfügbar sind. (vgl. Analog Devices Inc. 2019c, Kap. 3 S. 5 - 6) Dies gilt beispielsweise für die in dieser Arbeit relevante `rfftf_2` Funktion. Die Funktion berechnet mit einem Funktionsaufruf zwei unabhängige FFTs unter Anwendung eines speziellen Algorithmus (Decimation-in-Frequency (DIF) Radix-2 FFT). Allerdings gilt zu erwähnen, dass aufgrund der Spezialisierung und starken Optimierung die `rfftf_2` Funktion nur auf Daten im internen Speicher angewendet werden kann. Zusätzlich sind weiter Details für eine optimale Performanz zu berücksichtigen. Zum einen ist eine Aufteilung der Eingangsdaten auf unterschiedliche Speichersegmente zu empfehlen. Zum anderen wird eine spezielle Speicherausrichtung gefordert. Des Weiteren werden aufgrund der Speicherausrichtung falsche Ergebnisse erzielt, wenn die Daten auf dem Stack gespeichert sind. (vgl. Analog Devices Inc. 2019c, Kap. 3 S. 286)

trans.h

Durch die `trans.h` Header-Datei wird eine Sammlung an FFT Funktionen deklariert, die keine SIMD Unterstützung haben. Die Header-Datei definiert dieselben Funktionsnamen für die jeweiligen FFT Funktionen wie für die zuvor beschriebene Gruppe 1 in `filter.h`. Folglich müssen auch in diesem Fall für die jeweiligen FFT Größen unterschiedliche Funktionen inkludiert werden. Aufgrund der selben Signatur, können die Funktionen der `trans.h` mit denen der `filter.h` nicht gleichzeitig in einer Quelldatei inkludiert werden. (vgl. Analog Devices Inc. 2019c, Kap. 3 S. 10)

Komplexe Zahlen

Ursächlich für die unterschiedlichen Signaturen der Funktionsprototpyen in `filter.h` und `trans.h` ist die Abbildung von komplexen Zahlen. `trans.h` verwendet für die komplexen Zahlen zwei getrennte Float-Arrays für den Real- und Imaginärteil, die als Referenz beim Funktionsaufruf übergeben werden. Wobei in `filter.h` die folgende Datenstruktur für komplexe Zahlen verwendet wird, die in `complex.h` definiert ist. Dabei gilt anzumerken, dass beide Varianten nicht den komplexen Zahlen entsprechen, die in C99 definiert sind.

```
1  typedef struct {
2     float re;
3     float im;
4  } complex_float ;
```

7.2.3 General Purpose FFT Package

Als zusätzlichen Vergleichsalgorithmus wird die General Purpose FFT Package von Takuya OOURA verwendet. Die Bibliothek kann zur Berechnung der diskreten Fourier-, Cosinus- und Sinus-Transformation verwendet werden. Dabei können die Transformationen auf eindimensionale Eingangsvektoren angewendet werden, wenn deren Länge einer Zweierpotenz entspricht. (vgl. OOURA 2006) Die Vorteile der General Purpose FFT sind zum einen die freizügige Lizenz, die eine uneingeschränkte Verwendung der Software für nicht kommerzielle sowie kommerzielle Zwecke einräumt. Zum anderen kann die Implementierung ohne Weiteres auf andere Prozessoren portiert werden, da diese in reinem C implementiert wurde. Außerdem steht der Quellcode zur Verfügung, der ein Adaptieren und Analysieren des Codes erlaubt.

7.3 Performanzanalyse

Im Folgenden werden zuerst die Speicherzugriffszeiten des SHARC+ Kerns (siehe Unterabschnitt 7.3.1) untersucht. Anschließend wird die Messung der unterschiedlichen Performanz der FFT Bibliotheken und des FFT Hardware-Beschleunigers analysiert (siehe Unterabschnitt 7.3.2). Für die Messung der Taktzyklen in diesen Untersuchungen wurde in allen Fällen die `cycle_count.h` Bibliothek von Analog Devices verwendet. Die Bibliothek muss mittels Compiler Option `-DDO_CYCLE_COUNTS` aktiviert werden. Alle Performanzanalysen wurden zehnmal wiederholt. Anschließend wurde aus den zehn Resultaten der arithmetische Mittelwert gebildet. Für alle Versuche wurde das Caching deaktiviert.

7.3.1 Analyse der Speicherhierachie

Abbildung 7.2 zeigt die Zugriffszeiten des SHARC+ Kerns auf die jeweiligen Speicherbereiche. Für die Messung der Taktzyklen wurden alle Caches deakti-

viert. Des Weiteren wurde der ARM Kern deaktiviert, sodass dieser die Messergebnisse mit Zugriffen auf die Speicher nicht beeinflussen kann.

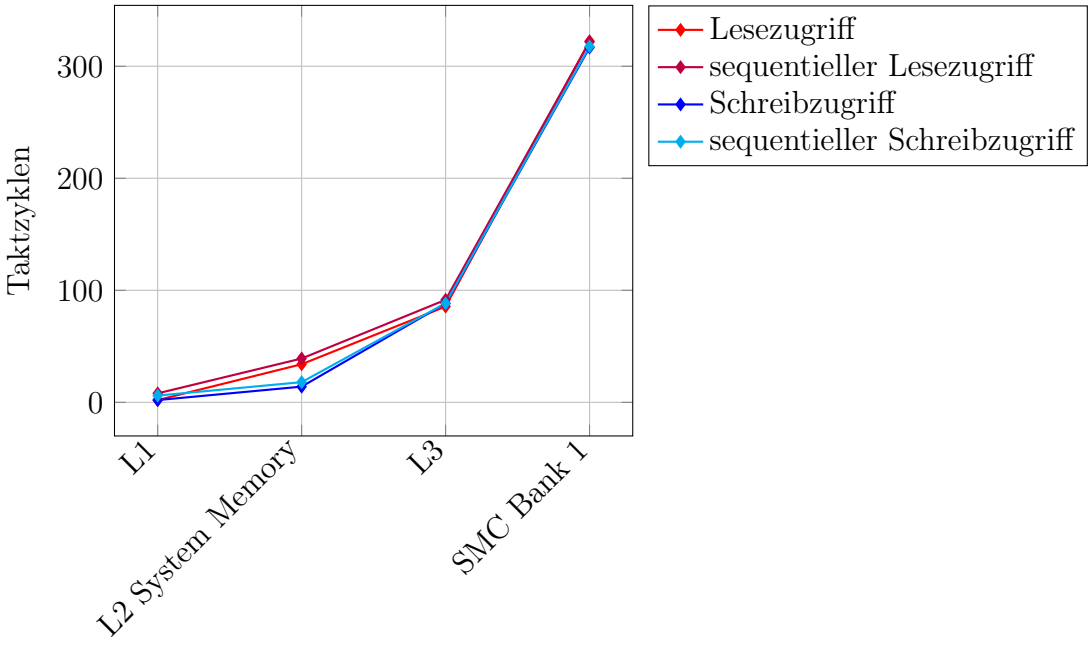


Abbildung 7.2: Gemessene Speicherzugriffszeiten des SHARC+ Kerns

Beim Test des Schreibzugriffs wird eine im L1 gespeicherte `uint32_t` Variable einem Element eines `uint32_t` Array zugewiesen, das je nach Experiment in dem jeweiligen Speicher (L1, L2 System Memory, L3 oder SMC Bank 1) liegt. Über den Static Memory Controller (SMC) Bank 1 ist der externe Speicher angebunden (siehe Tabelle 9.1 und Abbildung 1.1). Für den Lesezugriff wird ein Element aus einem `uint32_t` Array gelesen und einer Variable im L1 zugewiesen. Bei der Analyse wird einmal unterschieden, ob es sich um einen sequentiellen oder gleichbleibenden Speicher- bzw. Lesezugriff handelt. Anhand des in Abbildung 7.2 gezeigten Resultats, ist zu erkennen, dass sich die Anzahl der Taktzyklen für die Schreib- und Lesezugriffe erhöht, wenn auf niedrigere Ebenen der Speicherhierachie zugegriffen wird. Dabei erfolgt im Falle des L2 System Memorys der Schreibzugriff deutlich schneller als der Lesezugriff.

7.3.2 Analyse FFTA versus Programmbibliotheken

Im Folgendem werden die Performanz des FFT Hardware-Beschleunigers gegenüber den Implementierungen verglichen. Dabei werden alle FFT Berechnungen mittels Software Implementierungen und die Ansteuerung des Hardware-Beschleunigers mittels SHARC+ Kern durchgeführt.

Anforderungen an die FFT

In der Absicht, eine FFT auf ausgewählte Messkanäle durchzuführen, bedarf es einer Definition der Anforderungen. Aufgrund der Anwendung der Verstärkerplatine in der Energietechnik, sind die gängigen Netzfrequenzen und die Spezifikation der Verstärkerplatine für die FFT zu berücksichtigen. Die in der Energietechnik gängigen Netze der elektrischen Energieversorgung haben eine nominale Netzfrequenz von 50 Hz oder 60 Hz. Des Weiteren ist im Bahnbereich die Frequenz 16,7 Hz sowie in speziellen Bereichen wie der Luftfahrt 400 Hz gängig. Angesichts dieses Sachverhalts wird ein Frequenzbereich von zumindest 15 Hz bis 400 Hz bei einer Abtastrate von 10 kHz in Betracht gezogen. Damit eine möglichst genau Messung der Netzfrequenz möglich ist, bedarf es einer geeigneten Auflösung. Die Auflösung d_F der FFT berechnet sich wie folgt:

$$d_F = \frac{f_s}{N} \tag{7.1}$$

Dabei gilt: f_s ... Abtastrate in Hertz
N ... Anzahl Abtastwerte

Mit einer Anzahl von 2048 Abtastwerten kann bei einer Abtastrate von 10 kHz bereits eine Auflösung von ±4,88 Hz erreicht werden. Für eine Auflösung von ±2,5 Hz bedarf es bei einer Abtastrate von 10 kHz an 4092 Abtastwerten. Wird eine Auflösung von ±1 Hz benötigt, bedarf es bereits an 16384 Abtastwerten. Aufgrund der hohen Datenmenge bedarf es einer Analyse des Gesamtsystems, um die geeigneten Komponenten zum Rechnen der FFT zu identifizieren. Obwohl der ADSP-SC582 einen eigenen Hardware-Beschleuniger für die FFT bereitstellt und dieser ideal zur Auslagerung der FFT geeignet ist, bedarf es einer Analyse des Systems, da die die Zugriffszeiten auf die unterschiedlichen Speicher zu berücksichtigen sind. Außerdem ist die Rechenleistung der Analog Devices Bibliotheken und die des Hardware-Beschleunigers für die verwendeten Taktfrequenz nicht ausreichend dokumentiert. Die verwendete Konfiguration der unterschiedlichen Taktfrequenzen des Systems kann aus dem Anhang entnommen werden.

Beschreibung der Performanzanalyse

Folgende Auflistung beschreibt die Weise wie die Performanzanalyse für die einzelnen Bibliotheken und den Hardware-Beschleuniger durchgeführt wurde. Dabei wurden drei unterschiedliche Fälle für den Hardware-Beschleuniger betrachtet.

1. FFTA Blocking
   Diesbezüglich werden die von Analog Devices bereitgestellten Funktionen für den blockierenden Betrieb verwendet. Dabei wird die gesamte

Anzahl an benötigten Taktzyklen gemessen, bis der Funktionsaufruf abgeschlossen ist. Für die Abtastwerte von 64 bis 2048 wird die Funktion `accel_rfft_small` und für größere Werte die `accel_rfft_large` verwendet.

2. FFTA Pipe gesamt
   Für diesen Fall wird einmalig die Pipe des Hardware-Beschleunigers konfiguriert. Dabei wird die Zeit der Konfiguration beim Mitteln der Taktzyklen inkludiert, da beim Konfigurieren und Aufsetzen der Pipe bereits eine FFT berechnet wird. Daher benötigt die erste FFT Berechnung länger als die restlichen neun. Letztendlich wird wieder die gesamte Anzahl an benötigten Taktzyklen gemessen, bis die FFT Berechnung abgeschlossen ist.

3. FFTA Pipe
   In diesem Fall werden nur die Taktzyklen gemessen, die zum Konfigurieren sowie dem erneuten Starten der Pipe benötigt werden. Da für die Zeit der FFT Berechnung der Prozessor, der für die Ansteuerung des FFTAs verantwortlich ist, andere Aufgaben durchführen kann.

4. General Purpose FFT - OOURA Messung der Dauer des Funktionsaufrufs.

5. filter.h und filter.h mit SIMD
   In der Absicht, die mögliche Leistungssteigerung durch SIMD zu veranschaulichen, werden in diesem Fall die abwärtskompatiblen Implementierungen der filter.h ohne SIMD verwendet. Damit ein Vergleich der Funktionen mit und ohne SIMD durchgeführt werden kann. Dabei wird auch die Dauer des Funktionsaufrufs gemessen.

6. trans.h Messung der Dauer des Funktionsaufrufs.

7. rfftf_2 Ebenfalls wird in diesem Falle die Dauer des Funktionsaufrufs gemessen. Dabei muss beachtet werden, dass die `rfftf_2` Funktion pro Funktionsaufruf zwei FFT Berechnungen durchführt. Für den Versuch wurde dasselbe Signal für beide Eingangsdaten verwendet. Die in den Analysen angegebenen Taktzyklen für `rfftf_2` beziehen sich immer auf die Berechnung von zwei FFTs.

L1 als Speicher der Ein- und Ausgangsdaten der FFT

Zuallererst wurde die FFT Performanz der unterschiedlichen Funktionen der Bibliotheken und die des Hardware-Beschleunigers für den Fall untersucht, dass die Ein- und Ausgangsdaten im L1 Speicher gespeichert sind. Zunächst kann

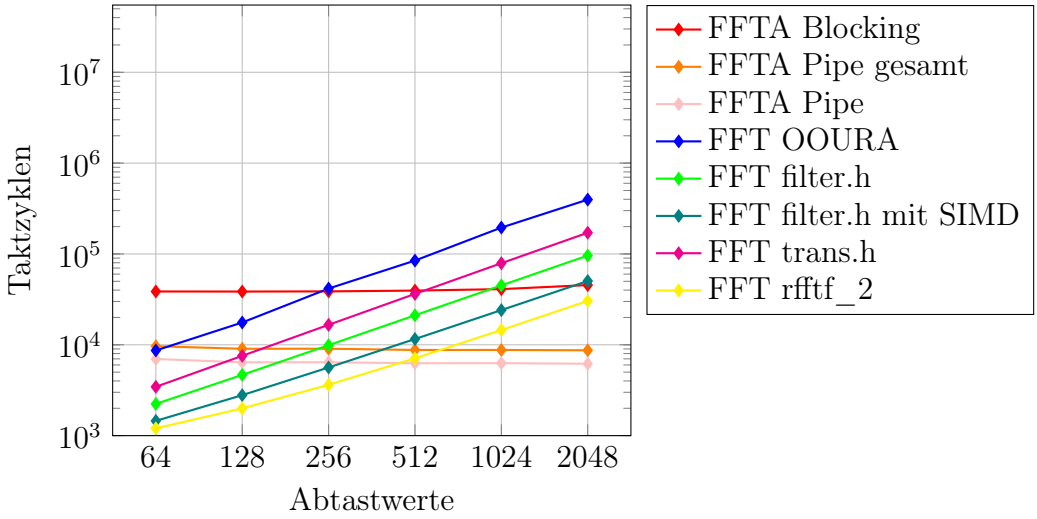


Abbildung 7.3: FFTA versus FFT Bibliotheken - L1

in Abbildung 7.3 erkannt werden, dass der FFTA für den blockierenden Modus nahezu gleich viele Taktzyklen für die FFT Berechnungen der unterschiedlichen FFT Größen benötigt. Dasselbe gilt für die „FFTA Pipe“ und „FFTA Pipe gesamt“ Ergebnisse. Dies ist durch den benötigten Mehraufwand zum Aufsetzen und Aktivieren der DMAs zu erklären, der gegenüber den kleinen FFT Berechnungen den Großteil der benötigten Rechenzeit ausmacht. Nicht verwunderlich ist die schlechte Performanz der General Purpose FFT von OOURA gegenüber den anderen Bibliotheken, da diese nicht für den ADSP-SC582 optimiert ist. Anhand der `filter.h` Bibliothek ist sehr gut die Leistungssteigerung zu erkennen, wenn für diese die SIMD optimierten Funktionen verwendet werden. Auch ohne die SIMD Optimierung kann mittels `filter.h` gegenüber der `trans.h` eine besser Performanz erzielt werden. Dessen ungeachtet konnte mit der `rfftf_2` Funktion das beste Ergebnis einer Software-Implementierung hinsichtlich der Rechengeschwindigkeit erzielt werden, obwohl durch diese Funktion sogar zwei FFTs gleichzeitig durchgeführt werden und diese Eigenschaft in diesem Benchmark nicht ausgenutzt wurde. Im Vergleich zu den Software-Implementierungen ist zu erkennen, dass der Mehraufwand des FFTAs erst ab einer Anzahl von 512 Abtastwerten vorteilhaft ist, sofern der Hardware-Beschleuniger im Pipe Betriebsmodus verwendet wird. Die Differenz zwischen „FFTA Pipe gesamt“ und „FFTA Pipe“ beträgt durchschnittlich rund 2570 Taktzyklen. Diese Differenz entspricht der Zeit, die der SHARC+ Prozessor für andere Aufgaben aufwenden kann.

L2 als Speicher der Ein- und Ausgangsdaten der FFT

Für den L2 Speicher wurde die Performanzanalyse im Speicherbereich `SHARC0` (siehe Tabelle 9.1 im Anhang) durchgeführt, der für gewöhnlich als Cache verwendet wird. Dabei ist in Abbildung 7.4 zu erkennen, dass die Performanz aufgrund der Speicherung der Daten im L2 System Memory vermindert wird. Lediglich die General Purpose FFT von OOURA ist nahezu deckungsgleich, zu den vorhergehenden Resultaten aus Abschnitt 7.3.2. Dabei sind die vorhergehenden Resultate in der Abbildung 7.4 strichliert dargestellt. In Anbetracht des FFT Hardware-Beschleunigers ist festzustellen, dass der Mehraufwand für dessen Verwendung bei 512 Abtastwerten immer noch rentabel ist.

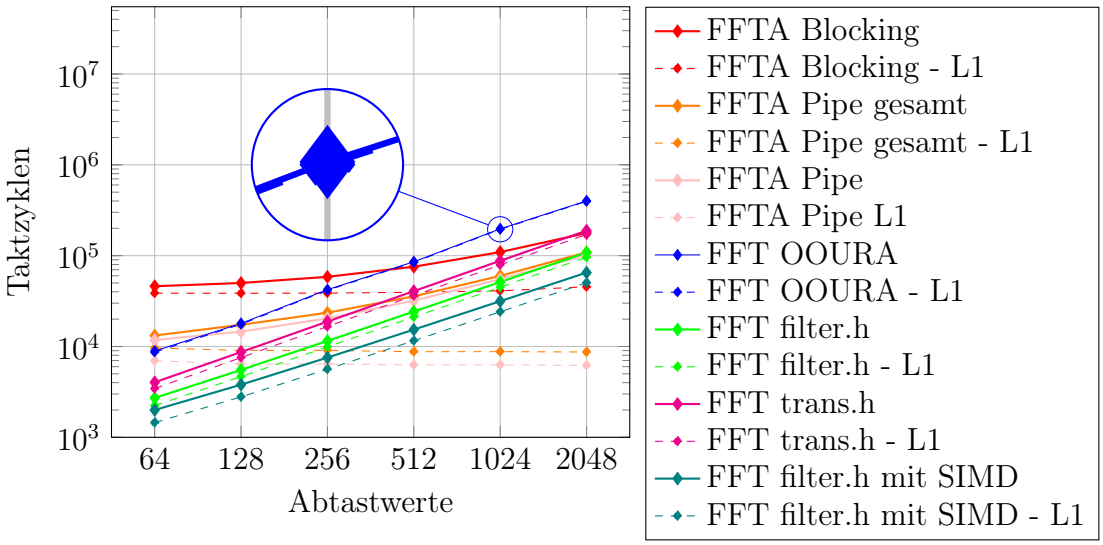


Abbildung 7.4: FFTA versus FFT Bibliotheken - L2

L3 als Speicher der Ein- und Ausgangsdaten der FFT

Abschließend wurde Untersucht, wie sich die Performanz auf den FFT Hardware-Beschleuniger und die verwendeten Bibliotheken auswirkt, wenn die Ein- und Ausgangsdaten im L3 Speicher gespeichert sind. Bemerkenswert dabei ist die in Abbildung 7.5 gezeigte Performanz der General Purpose FFT von OOURA, die nun besser als der Hardware-Beschleuniger im blockierenden Betrieb abschneidet. Weiterhin kann die General Purpose FFT von OOURA ab 8192 Abtastwerten auch besser Ergebnisse als die `filter.h` und `trans.h` erzielen. Im Falle des FFTAs lässt sich die nicht lineare Steigerung der benötigten Taktzyklen aufgrund der in Abschnitt 7.2 beschriebenen Limitierung erklären. Da der FFTA ab einer Anzahl von 4096 Abtastwerten die Berechnung der FFT aufgrund des zu geringen Speichers in mehreren Phasen durchführen muss, wird

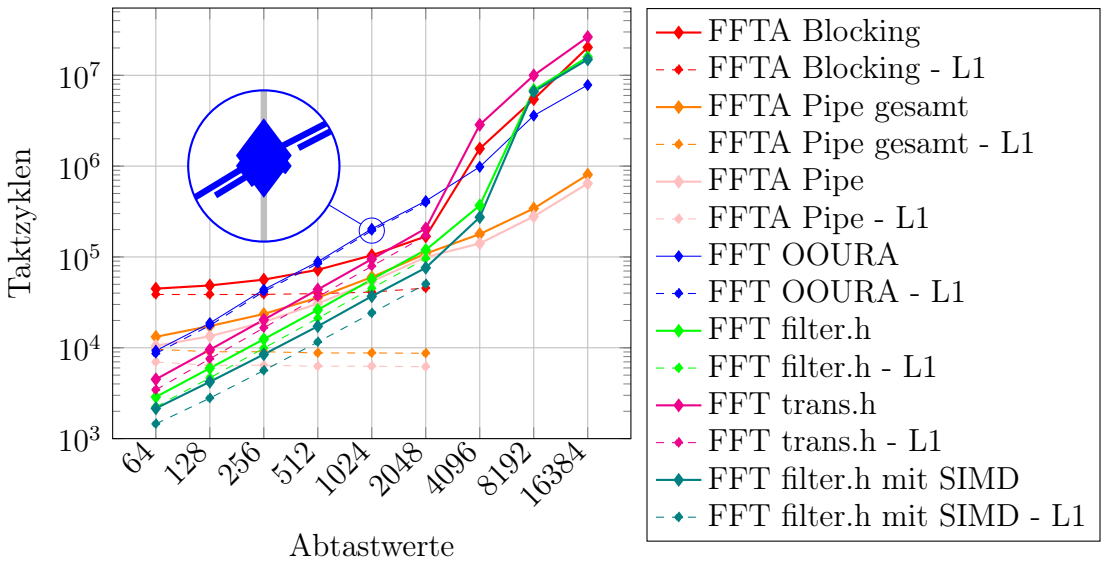


Abbildung 7.5: FFTA versus FFT Bibliotheken - L3

nun zusätzlicher Mehraufwand zum Aufsetzen der DMAs benötigt. Die Reduktion der Performanz der Implementierungen `filter.h` und `trans.h` von Analog Devices ist aufgrund der zusätzlich benötigten trigonometrischen Werte zu erklären, die von den Funktionen für die Berechnung der FFT benötigt und von diesen inkludiert werden. Allerdings gilt dies auch für die Abtastraten von 64 bis 2048, daher war für die Performanz der größeren FFTs auch eine lineare Steigerung zu erwarten.

8 Verknüpfung der Fallbeispiele

Mithilfe der Fallbeispiele in den Kapiteln 6 und 7 wurden die bedeutsamen Eigenheiten hinsichtlich der Programmierung des heterogenen Multiprozessor-Ein-Chip-Systems ADSP-SC582 erarbeitet, damit das System in Zukunft zur Signalverarbeitung von Messsignalen der Verstärkerplatine verwendet werden kann. In diesem Kapitel wird ein Konzeptnachweis erbracht, der anhand einer Implementierung die Möglichkeiten des heterogenen Systems veranschaulicht. Beispielhaft wird gezeigt, wie der ARM und SHARC+ Kern sowie der FFT Hardware-Beschleuniger gemeinsam in einem Anwendungbeispiel verwendet werden. Das Ziel des Anwendungsbeispiels ist es, über die `UART0`-Schnittstelle einen serialisierten Befehl zu empfangen und auszuwerten, um anschließend einen Ausgangskanal der Verstärkerplatine zu steuern und die Frequenzanteile des ausgegebenen Wechselspannungssignals mittels FFT zu bestimmen. Anschließend werden die Daten der FFT aufbereitet und über die `UART0`-Schnittstelle versendet. Dabei wird der ARM Kern für den Empfang des serialisierten Befehls über die `UART0`-Schnittstelle und zur Steuerung der Verstärkerplatine verwendet. Der FFT Hardware-Beschleuniger wird zur Berechnung der FFT verwendet. Der SHARC+ Kern wird zur Steuerung des FFT Hardware-Beschleunigers, Berechnung des Absolutbetrags der FFT Resultate, Serialisierung der FFT Ausgangsdaten und zum Versenden der Daten über die `UART0`-Schnittstelle verwendet. Im Folgenden wird die Implementierung beschrieben.

Implementierung

Abbildung 8.1 zeigt ein Sequenzdiagramm der Implementierung, die in der folgenden Auflistung beschrieben wird.

Computer Für den Konzeptnachtweis wurde ein Computer verwendet, um Befehle an den ARM Kern zu senden. Der Computer sendet die Befehle im MessagePack-Format über ein USB/TTL-Konverter und die `UART0`-Peripherie an den ARM Kern. Durch das Versenden des jeweiligen Befehls kann die Spannung und Frequenz des Wechselspannungsausgangs der Verstärkerplatine eingestellt werden. Wie in Abbildung 8.1 zu erkennen ist, wird durch die Nachricht des Computers eine Messung gestartet. Ebenfalls kann die Messung durch einen Befehl gestoppt werden.

ARM Bevor durch den Computer eine Messung gestartet werden kann, muss durch den ARM Kern der SHARC+ aktiviert werden, wie dies in Abbildung 8.1 ersichtlich ist. Direkt nach der Aktivierung des SHARC+ Kerns sendet der ARM Kern mithilfe der MCAPI Implementierung von Analog Devices eine Nachricht an den SHARC+ Kern. Dabei werden die Adressen von bestimmten Variablen mit dem SHARC+ Kern geteilt (siehe dazu Analog Devices Inc. 2015). Durch Zuweisung der Adressen an Adresszeiger kann der SHARC+ Kern die Variablen in gleicherweise verwenden wie der ARM Kern. Die geteilten Variablen dienen dazu, dass der SHARC+ über den aktuellen Status des ARM Kerns und dessen Zustandsmaschine informiert bleibt. Die Variablen liegen im geteilten L2 System Memory in einem Bereich, der nicht zwischengespeichert wird. Somit kann der SHARC+ Kern die Variablen ohne Weiteres verwenden. Eine der geteilten Variablen ist die boolesche Variable `isMeasurementActive`, die in der Implementierung durch den SHARC+ Kern kontinuierlich überprüft wird. Wird nun durch den Computer eine Messung gestartet, so empfängt der ARM Kern den Befehl über die `UART0`-Schnittstelle und deserialisiert die eingehenden Daten. Nach der Verarbeitung des Befehls gibt der ARM Kern dem FPGA einen neuen Zielwert vor, dazu wird der neue Zielwert in einen Bereich des externen Speichers geschrieben, der kontinuierlich durch den FPGA überprüft wird. Anschließend wird durch den ARM Kern die boolesche Variable auf `true` gesetzt. Wird ein Befehl empfangen, der ein Stoppen der Messung verlangt, so wird dem FPGA ein neuer Zielwert vorgegeben und die boolesche Variable auf `false` gesetzt.

FPGA Der FPGA befolgt den Zielwert, der durch den ARM Kern vorgegeben wird, dieser wird durch den FPGA kontinuierlich überprüft. Anhand des Zielwerts steuert der FPGA die Leistungselektronik. Zur Wahrung der Übersichtlichkeit wurde die Leistungselektronik im Sequenzdiagramm nicht dargestellt.

SHARC+ Der SHARC+ Kern wird durch den ARM Kern aktiviert und empfängt durch den Nachrichtenaustausch mittels MCAPI Adressen von Variablen, die den aktuellen Status der Zustandsmaschine des ARM Kerns abbilden. Anschließend überprüft der SHARC+ Kern kontinuierlich den Status der Zustandsmaschine und wartet, bis eine aktive Messung erkannt wird. Wenn eine Messung aktiv ist, wird durch den SHARC+ Kern der Synchronisation GPIO Interrupt abonniert (siehe Abbildung 1.1), der durch den FPGA im 10 kHz Takt ausgelöst wird. Wenn der Interrupt ausgelöst wird, lädt der SHARC+ Kern den Momentanwert des Ausgangsignals, der vom FPGA in einem Bereich des externen Speicher gespeichert

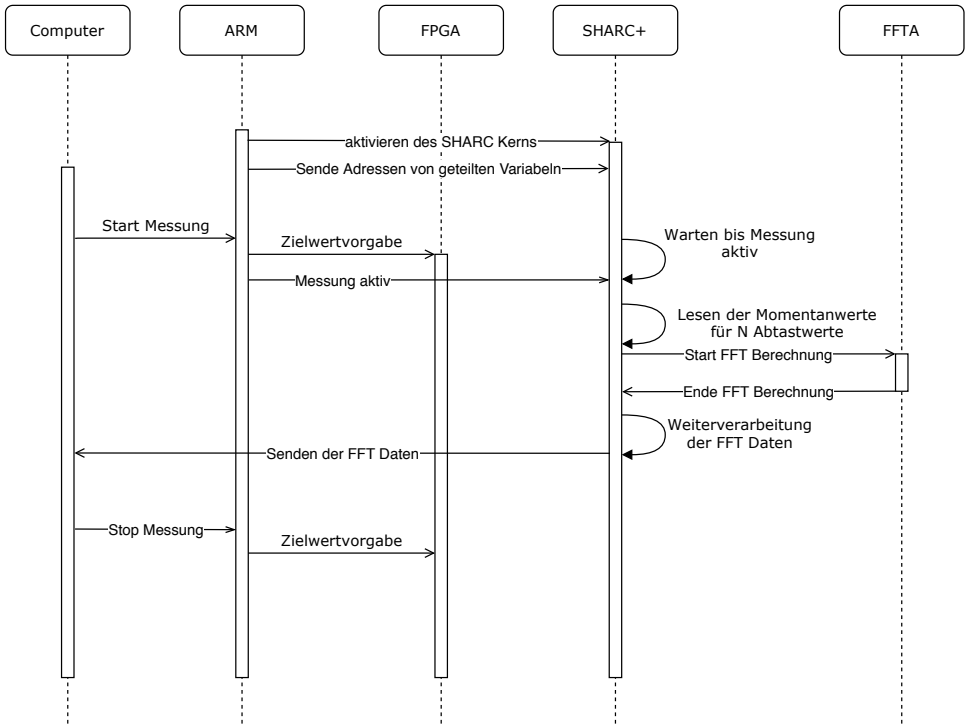


Abbildung 8.1: Ablauf des Anwendungsbeispiels

wurde. Dies wird so oft wiederholt, bis genügend Abtastwerte zur Berechnung der FFT zur Verfügung stehen. Wenn genügend Abtastwerte vorhanden sind, startet der SHARC+ Kern die FFT Berechnung durch den Aufruf der `accel_rfftN()` Funktion des FFT Hardware-Beschleunigers. Der SHARC+ Kern wartet anschließend, bis die FFT Berechnung durch den Hardware-Beschleuniger abgeschlossen ist und berechnet danach den Betrag für jeden Abtastwert des FFT Resultats. Anschließend werden die Daten im MessagPack-Format serialisiert und mittels `UART0`-Peripherie an den Computer gesendet.

FFTA Für dieses Anwendungsbeispiel wird der FFTA im blockierenden Modus betrieben. Der FFT Hardware-Beschleuniger berechnet die FFT der reellen Eingangsdaten und liefert komplexe Ausgangsdaten zurück.

Analyse der Implementierung Zusammenfassend kann anhand des Anwendungsbeispiel festgestellt werden, dass durch die Kombination der zwei Fallbeispiel und der Verwendung der bestehenden Hardware ein praktisches Beispiel gezeigt werden kann, das die grundsätzlichen Möglichkeiten des heteroge-

nen Systems veranschaulicht. Durch denn Nachrichtenaustausch mittels MCAPI können Speicheradressen problemlos zwischen den zwei Prozessoren ausgetauscht werden. Dabei bietet dies den Vorteil, dass die MCAPI-Schnittstelle nur während der Initialisierung des Systems benötigt wird. Anschließend kann der Speicherinhalt durch beide Prozessoren verwendet werden. Außerdem bietet die Verwendung von Interrupts die Möglichkeit, auf Ereignisse zu reagieren, wie dies durch den 10 kHz Interrupt realisiert wird, wenn neue Messwerte der Verstärkerplatine verfügbar sind. Allerdings kann die aktuelle Implementierung verbessert werden. Zum einen wird zur Berechnung der FFT der Hardware-Beschleuniger im blockierenden Modus betrieben, sodass der SHARC+ Prozessor für den Moment der FFT Berechnung ebenfalls blockiert ist. Zum anderen, muss durch den SHARC+ Prozessor ständig geprüft werden, ob eine Messung durch den ARM Kern gestartet wurde.

9 Fazit

In dieser Masterarbeit wurde ein existierendes System auf dessen Heterogenität analysiert, welches ein ADSP-SC582 Multiprozessor Ein-Chip-System sowie einen FPGA enthält. Zuallererst wurden die theoretischen Grundlagen und der aktuelle Stand der Technik erörtert. Dabei wurden die Gründe für heterogenes Rechnen und die daraus resultierende Trendwende in der Informatik erläutert. Ferner wurden die unterschiedlichen Arten von Multiprozessorsystemen, deren Speicheranbindung und die Herausforderungen von heterogenem Rechnen thematisiert.
Infolgedessen wurden Programmiermodelle wie bspw. OpenCL, OpenMP oder HetroOMP vorgestellt, welche die Programmierung von heterogenen Systemen vereinfachen. Weiterführend wurden die unterschiedlichen Arten von gängigen Prozessoren in eingebetteten Systemen erklärt. Anhand der Theorie wurde gezeigt, dass es sich bei dem ADSP-SC582 um ein eng gekoppeltes, asymmetrisches Multiprozessor System handelt, welches über eine NUCA Architektur verfügt. Anschließend wurde nach dem aktuellen Stand der Technik die Limitierung des ADSP-SC582 hinsichtlich dessen Programmierung dargelegt, da das Ein-Chip-System derzeit lediglich den MCAPI Standard unterstützt. Dabei haben Recherchen im Rahmen dieser Arbeit gezeigt, dass ARM und DSP Ein-Chip-Systeme von Texas Instruments bereits unterschiedliche Programmiermodelle zur Parallelisierung von Aufgaben unterstützen. Angesichts der Annulation der Markennamen der Multicore Association Standards sowie der fragwürdigen Existenz der Multicore Association ist davon auszugehen, dass eine weitere Entwicklung beziehungsweise Standardisierung des MCAPI Standards nicht mehr durch diese stattfindet. Lediglich der OpenAMP Standard wird derzeit durch Linaro weitergeführt. Dies sollte für zukünftige Entwicklungen mit der ADSP-SC58x Produktreihe bedacht werden. Darüber hinaus wurde die offene und standardisierte RISC-V Befehlssatzarchitektur erwähnt, die proprietäre Technologien von ARM und anderen Unternehmen ersetzen könnte. Dies wäre auch für die im ADSP-SC582 eingesetzten proprietären Kerne möglich. Die aktuellen Entwicklungen zeigen, dass RISC-V durch seine offene Befehlssatzarchitektur bereits richtungsweisend ist. Dabei geraten anbietende Unternehmen von geistigem Eigentum wie beispielsweise ARM zunehmend unter Zugzwang und veröffentlichen Teile ihrer Technologie.
Nennenswert hinsichtlich heterogenem Rechnen ist dabei die Heterogeneous

Embedded Research Plattform und der dazu betriebenen Forschung (siehe Abschnitt 2.5), welche den Forschungsrückstand der Wissenschaft gegenüber den führenden Unternehmen und ihren Entwicklungen minimiert.

Mithilfe der Untersuchungen sollte eine Antwort auf die folgende Forschungsfrage gefunden werden: *Was muss bei der Programmierung von ARM und DSP Ein-Chip-Systemen hinsichtlich der heterogenen Prozessoren und Peripheriebausteinen beachtet werden?"*. Zur Beantwortung der Forschungsfrage wurden zwei Fallbeispiele gewählt, sodass mit deren Hilfe bedeutsame Eigenheiten zur Programmierung und Verwendung des Ein-Chip-Systems für zukünftige Entwicklungen erörtert wurden. Anhand des Fallbeispiels der UART-Schnittstelle konnten für die Programmierung des ARM und SHARC+ Prozessors einige Erkenntnisse gewonnen werden. Zum einen muss bei der Programmierung der Multiprozessor Offset beachtet werden, falls auf den L1 Speicher des SHARC+ Kerns mittels Peripherie oder Hardware-Beschleuniger zugegriffen wird. Für den Fall, dass der ARM Kern auf den L1 des SHARC+ Kerns zugreifen soll, muss die MMU dementsprechend konfiguriert werden. Zum anderen konnten weitere Erkenntnisse hinsichtlich der unterschiedlichen Definitionen von Datentypen bei der Programmierung des SHARC+ Kerns festgestellt werden. Auch wurde dargelegt, dass die Änderung der Definition eines Datentyps mittels Compileroption weitgehende Folgen auf das Gesamtsystem hat.
Mithilfe des Fallbeispiels der FFT Berechnung konnte gezeigt werden, dass sich die von Analog Devices bereitgestellten Bibliotheken stark in deren Anwendung, Performanz, Kompatibilität und Speicherplatzbedarfs voneinander unterscheiden. Zum Vergleich der FFT Bibliotheken wurde zusätzlich die General Purpose FFT von OOURA gewählt, welche nicht für den SHARC+ Prozessor optimiert ist. Nennenswert dabei ist, dass die General Purpose FFT von OOURA für größere FFTs doch konkurrenzfähig zu den von Analog Devices bereitgestellten Implementierungen ist, sofern die Ein- und Ausgangsdaten der FFT im L3 Speicher hinterlegt sind. Zudem wurde gezeigt, dass die Verwendung des Hardware-Beschleunigers erst ab einer gewissen Anzahl an Abtastwerten rentabel ist. Für die Programmierung des Hardware-Beschleunigers wurde gezeigt, dass dieser nicht auf Registerebene programmiert werden kann. Ferner muss der Multiprozessor Offset für den Fall, dass die Ein- und Ausgangsdaten im L1 Speicher des SHARC+ Prozessors liegen nicht berücksichtigt werden, da dieser automatisch durch die proprietären Funktionen von Analog Devices überprüft und gegebenenfalls hinzugefügt wird. Der Multiprozessor Offset sowie die Compileroptionen bezüglich der Datentypen und Byte-Reihenfolge sind für alle anderen Peripherien und Hardware-Beschleuniger der ADSP-SC58x Produktreihe gültig.

Anhand des Fallbeispiels FFT wurden weitere heterogenen Aspekte gezeigt, die bei der Programmierung von heterogenen Ein-Chip-Systemen zu berücksichtigen sind. Erstens wurden durch die Analyse der Speicherhierachie die unterschiedlichen Speicherzugriffszeiten der Hierachieebnen veranschaulicht, welche einen starken Einfluss auf die Performanz eines Algorithmus haben. Zweitens wurde durch die Verwendung unterschiedlicher Bibliotheken gezeigt, dass diese unterschiedliche Vor- und Nachteile haben. Beispielsweise funktioniert die rfftf_2 Funktion aufgrund ihrer starken Optimierung nur für den internen Speicher des SHARC+ Kerns. Dafür können gleichzeitig zwei FFTs durchgeführt werden. Weitere Unterschiede der FFT Bibliotheken sind die verschiedenartige Abbildung von komplexen Zahlen, SIMD-Unterstützung und Optimierung des Speicherplatzes durch das Wiederverwenden des Eingangspuffers als temporärer Speicher. Auch der Hardware-Beschleuniger hat unterschiedliche Eigenschaften wie die Betriebsmodi die berücksichtigt werden müssen. Die wesentliche Einschränkung des Hardware-Beschleunigers ist der geringe Speicherplatz, daher müssen größere FFTs in mehreren Phasen durchgeführt werden. In dieser Arbeit wurde das kontinuierliche Betreiben der Pipe des FFT-Beschleunigers für die Experimente nicht betrachtet. Je nach Anwendungsgebiet und Größe der FFT kann der kontinuierliche Betriebsmodus zur FFT Berechnung geeignet sein, da keine Interaktion mit anderen Prozessoren benötigt wird. Allerdings wird in diesem Fall eine konstante Anzahl der Abtastwerte vorausgesetzt. Für Applikationen mit unterschiedlichen FFT Größen muss evaluiert werden, ob sich der Mehraufwand hinsichtlich des Betreibens des Hardware-Beschleunigers auszahlt. Aus Praktibilitätsgründen konnte in dieser Arbeit der FFT Hardware-Beschleuniger nicht vollständig analysiert werden. Zum einen ist dies auf die proprietären Bibliotheken des Hardware-Beschleunigers zurückzuführen. Zum anderen gibt es noch weitere Untersuchungen die benötigt werden, um ein Gesamtbild des Hardware-Beschleunigers zu erhalten. Beispielsweise wurde in der gesamten Arbeit der Energieverbrauch der jeweiligen Prozessoren nicht betrachtet. Eine Untersuchung der Rechenleistung und des Energieverbrauchs des Hardware-Beschleunigers bei unterschiedlichen Taktfrequenzen würde Erkenntnisse erbringen, die speziell für eingebettete System von Relevanz sind, die mit einer limitierten Energieversorgung betrieben werden. Zudem wurde der Hardware-Beschleuniger immer gegenüber dem SHARC+ Kern verglichen, der derzeit mit einer höheren Taktfrequenz als der ARM Kern betrieben wird. Schlussendlich muss für das FFT Fallbeispiel immer der jeweilige Anwendungsfall und dessen Anforderungen betrachtet werden, um eine geeignet Variante zur Berechnung der FFT zu bestimmen. Die in Kapitel 7 beschriebenen Anforderungen an die FFT für die Verstärkerplatine sind ein gutes Beispiel dafür. Werden beispielsweise für eine Anwendung eine Frequenzauflösung der FFT

von 2,5 Hz benötigt, so genügt es, 4096 Messwerte bei einer Abtastrate von 10 kHz zu verwenden. Wird jedoch eine Auflösung von 1 Hz benötigt, so bedarf es bereits an 16384 Abtastwerten. Für beide Fälle ist in Abbildung 7.5 zu erkennen, dass das Verwenden des Hardware-Beschleunigers bereits eine bessere Performanz als die der Software-Bibliotheken aufweist, jedoch wird aufgrund der Limitierung des Speichers zusätzlicher Implementierungsaufwand benötigt. Durch die geeignete Auswahl der Fallbeispiele konnte die Forschungsfrage für wichtige Teile des Ein-Chip-Systems beantwortet werden. Abschließend wurde als Konzeptnachweis eine Verknüpfung der Fallbeispiele durchgeführt, sodass die grundsätzlichen Möglichkeiten des heterogenen Rechnens des ADSP-SC58x veranschaulicht werden. Dadurch konnte gezeigt werden, wie das bestehende System für die Signalverarbeitung erweitert werden kann und mehrere unterschiedliche Prozessoren zum gemeinsamen Lösen einer Aufgabe verwendet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADI Analog Devices

ALU Arithmetic Logic Unit

ASIC Application Specific Integrated Circuit

CMOS Complementary Metal-Oxide-Semiconductor

CMSIS Cortex-Microcontroller-Software-Interface-Standard

CPU Central Processing Unit

DFT Diskrete Fourier Transformation

DMA Direct Memory Access

DRAM Dynamic Random Access Memory

DSP Digitaler Signalprozessor

DVFS Dynamic Frequency And Voltage Scaling

FET Feldeffekttransistor

FFT Fast Fourier Transformation

FFTA Fast Fourier Transformation Accelerator

FLOPS Floating Point Operation Per Second

FPGA Field Programmable Gate Array

GIC Generic Interrupt-Controller

GPP General Purpose Processor

GPU Graphic Processing Unit

HDL Hardware Description Language

HERO Heterogeneous Embedded Research Platform

HLS High Level Synthesis

IC Integrated Circuit

ICC Inter-Core Communications

LLVM Low Level Virtual Machine

LLVM-IR Low Level Virtual Machine Intermediate Representation

MAD Multiply-Add

MAC Multiply-Accumulate

MDMA Memory To Memory Direct Memory Access

NUCA Non-Uniform Cache Access

NUMA Non-Uniform Memory Access

SCB System Crossbar

SHARC Super Harvard Architecture

SIMD Single Instruction Multiple Data

SMC Static Memory Controller

SoC System-on-a-Chip

SOI Silicon-On-Insulator

SPP Special Purpose Processor

SRAM Static Random Access Memory

TI Texas Instruments

UART Universal Asynchronous Receiver Transmitter

UMA Uniform Memory Access

Literatur

754-2008 - IEEE Standard for Floating-Point Arithmetic (2008). OCLC: 1112398168. Place of publication not identified: IEEE. ISBN: 978-0-7381-5752-8. URL: https://ieeexplore.ieee.org/servlet/opac?punumber=4610933 (besucht am 15. 05. 2020).

Analog Devices Inc. (2018). *ADSP-SC582/SC583/SC584/SC587/SC589/ADSP-21583/21584/21587 Data Sheet*. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADSP-SC582_583_584_587_589_ADSP-21583_584_587.pdf (besucht am 15. 05. 2020).

– (2020a). *ADSP-SC58x and ADSP-2158x Series | Analog Devices*. URL: https://www.analog.com/en/products/landing-pages/001/adsp-sc58x-adsp-2158x-series.html (besucht am 24. 05. 2020).

– (2017). *ADSP-SC58x/ADSP-2158x SHARC+ Processor Hardware Reference*. URL: https://www.analog.com/media/en/dsp-documentation/processor-manuals/SC58x-2158x-hrm.pdf (besucht am 11. 05. 2020).

– (2019a). *CCES 2.9.0 C/C++ Compiler and Library Manual for Blackfin Processors*. URL: https://www.analog.com/media/en/dsp-documentation/software-manuals/cces-BlackfinCompiler-library-manual.pdf (besucht am 15. 05. 2020).

– (2019b). *CCES 2.9.0 C/C++ Compiler Manual for SHARC Processors*. URL: https://www.analog.com/media/en/dsp-documentation/software-manuals/cces-SharcCompiler-manual.pdf (besucht am 15. 05. 2020).

– (2019c). *CCES 2.9.0 C/C++ Library Manual for SHARC Processors*. URL: https://www.analog.com/media/en/dsp-documentation/software-manuals/cces-SharcLibrary-manual.pdf (besucht am 15. 05. 2020).

– (2020b). *FAQ: What is Multiprocessor O set in ADSPSC58x/ADSP-215xx processors?? - Documents - ADSP-SC5xx/ADSP-215xx - EngineerZone*. URL: https://ez.analog.com/dsp/sharc-processors/adsp-sc5xxadsp-215xx/w/documents/5041/faq-what-is-multiprocessor-offset-in-adspsc58x-adsp-215xx-processors (besucht am 06. 07. 2020).

Analog Devices Inc. (2019d). *SHARC+ Core Programming Reference*. URL: https://www.analog.com/media/en/dsp-documentation/processor-manuals/SC58x-2158x-prm.pdf (besucht am 23. 05. 2020).

– (11. Juni 2015). „Using MCAPI/MDMA for ADSP-SC58x Dual-SHARC Audio Talkthrough (EE-377)“. In: Unter Mitarb. von Eric Gregori.

ARM Limited (2011). *ARM Cortex-A Series Programmer's Guide*. URL: https://static.docs.arm.com/den0013/d/DEN0013D_cortex_a_series_PG.pdf (besucht am 23. 05. 2020).

– (2009). *Cortex-A5 Technical Reference Manual*. URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0433c/DDI0433C_cortex_a5_trm.pdf (besucht am 23. 05. 2020).

– (2020). *Cortex-M33*. Arm Developer. Library Catalog: developer.arm.com. URL: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m33 (besucht am 15. 06. 2020).

Asche, Rüdiger R. (2016). *Embedded Controller: Grundlagen und praktische Umsetzung für industrielle Anwendungen*. OCLC: 958467189. Wiesbaden: Springer Vieweg. 293 S. ISBN: 978-3-658-14849-2 978-3-658-14850-8.

Bengel, Günther u. a., Hrsg. (2015). *Masterkurs parallele und verteilte Systeme: Grundlagen und Programmierung von Multicore-Prozessoren, Multiprozessoren, Cluster, Grid und Cloud*. 2., erw. und aktualisierte Aufl. Lehrbuch. OCLC: 913050632. Wiesbaden: Springer Vieweg. 495 S. ISBN: 978-3-8348-2151-5 978-3-8348-1671-9.

Blinka, Ellen (2014). „Programming TI KeyStone II-based ARM + DSP Devices using Industry Standard Tools“. In: S. 2.

CC (Conference), Evelyn Duesterwald und ETAPS (Conference) (2004). *Compiler construction: 13th international conference, CC 2004, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29-April 2, 2004 : proceedings*. Berlin; New York: Springer. ISBN: 978-3-540-24723-4.

Chisnall, David (2020). *There's No Such Thing as a General-purpose Processor - And the belief in such a device is harmful - ACM Queue*. URL: https://queue.acm.org/detail.cfm?id=2687011 (besucht am 25. 07. 2020).

Dennard, Robert H u. a. (1999). „Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions“. In: *PROCEEDINGS OF THE IEEE* 87.4, S. 11.

Fischer, Peter und Peter Hofer (2008). *Lexikon der Informatik*. 14., überarb. Aufl. OCLC: 198923430. Berlin: Springer. 966 S. ISBN: 978-3-540-72549-7 978-3-540-72550-3.

Furuhashi, Sadayuki (2020a). *MessagePack: It's like JSON. but fast and small.* URL: https://msgpack.org/#crosslang (besucht am 25. 07. 2020).

– (2020b). *msgpack/msgpack - Specification*. GitHub. Library Catalog: github.com. URL: https://github.com/msgpack/msgpack (besucht am 25. 07. 2020).

Gan, Woon-Seng und Sen-Maw Kuo (2007). *Embedded signal processing with the micro signal architecture*. OCLC: 637272593. Hoboken, NJ: Wiley-Interscience [u.a.] 486 S. ISBN: 978-0-471-73841-1.

Gordon Moore (2005). *Cramming more componentsonto integrated circuits*. URL: https://web.archive.org/web/20160308091431/https://web.eng.fiu.edu/npala/eee6397ex/gordon_moore_1965_article.pdf (besucht am 24. 04. 2020).

Greengard, Samuel (20. Apr. 2020). „Will RISC-V revolutionize computing?" In: *Communications of the ACM* 63.5, S. 30–32. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3386377. URL: https://dl.acm.org/doi/10.1145/3386377 (besucht am 24. 07. 2020).

Gunyon, Charlie (26. Juni 2020). *camgunz/cmp*. original-date: 2014-04-25T19:51:21Z. URL: https://github.com/camgunz/cmp (besucht am 25. 07. 2020).

Harris, Sarah L. und David Money Harris (2016). *Digital design and computer architecture*. ARM® Edition. OCLC: ocn900028206. Amsterdam: Elsevier/-Morgan Kaufmann. 559 S. ISBN: 978-0-12-800056-4.

Hennessy, John L. (2019). *Computer architecture: a quantitative approach*. Sixth edition. Cambridge, MA: Morgan Kaufmann Publishers. 1 S. ISBN: 978-0-12-811905-1.

HERO: The Open Heterogeneous Research Platform (2020). URL: https://pulp-platform.org/hero.html (besucht am 26. 07. 2020).

Hill, M. D. und M. R. Marty (Juli 2008). „Amdahl's Law in the Multicore Era". In: *Computer* 41.7, S. 33–38. ISSN: 1558-0814. DOI: 10.1109/MC.2008.209.

Hübner, Michael und Jürgen Becker, Hrsg. (2011). *Multiprocessor system-on-chip: hardware design and tool integration*. OCLC: 706881597. New York, NY: Springer. 270 S. ISBN: 978-1-4419-6459-5 978-1-4419-6460-1.

ISO/IEC TR 18037:2008(en), Programming languages — C — Extensions to support embedded processors (2020). URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:18037:ed-2:v1:en (besucht am 25. 07. 2020).

Keane, Erich (21. Apr. 2020). *The New Clang _ExtInt Feature Provides Exact Bitwidth Integer Types*. Library Catalog: blog.llvm.org. URL: http://blog.llvm.org/2020/04/the-new-clang-extint-feature-provides.html (besucht am 22. 04. 2020).

Kumar, Vivek, Abhiprayah Tiwari und Gaurav Mitra (2019). „HetroOMP: OpenMP for Hybrid Load Balancing Across Heterogeneous Processors“. In: *OpenMP: Conquering the Full Hardware Spectrum*. Hrsg. von Xing Fan u. a. Bd. 11718. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, S. 63–77. ISBN: 978-3-030-28595-1 978-3-030-28596-8. DOI: 10.1007/978-3-030-28596-8_5. URL: http://link.springer.com/10.1007/978-3-030-28596-8_5 (besucht am 16. 07. 2020).

Kurth, Andreas u. a. (Okt. 2017). „HERO: Heterogeneous Embedded Research Platform for Exploring RISC-V Manycore Accelerators on FPGA“. In: *Proceedings of Computer Architecture Research with RISC-V Workshop (CARRV' 17)*. First Workshop on Computer Architecture Research with RISC-V (CARRV 2017). Accepted: 2017-12-11T08:38:34Z. DOI: 10.3929/ethz-b-000219249. URL: https://www.research-collection.ethz.ch/handle/20.500.11850/219249 (besucht am 26. 07. 2020).

Mansureh, Moghaddam Shahraki, Jae-Min Cho und Kiyoung Choi (2016). „Reconfigurable Architectures“. In: *Handbook of Hardware/Software Codesign*. Hrsg. von Soonhoi Ha und Jürgen Teich. Dordrecht: Springer Netherlands, S. 1–42. ISBN: 978-94-017-7358-4. DOI: 10.1007/978-94-017-7358-4_12-1. URL: http://link.springer.com/10.1007/978-94-017-7358-4_12-1 (besucht am 23. 05. 2020).

Marowka, A. (2012). „Extending Amdahl's Law for Heterogeneous Computing“. In: *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, S. 309–316.

Marwedel, Peter (2008). *Eingebettete Systeme =: Embedded system design*. Übers. von Lars Wehmeyer. Korrigierter Nachdruck 2008. eXamen.press. OCLC: 552488933. Berlin Heidelberg: Springer. 264 S. ISBN: 978-3-540-34048-5 978-3-540-34049-2.

Max Roser (2019). *Transistor-Count-over-time-to-2018.png (PNG Image, 6529 4716 pixels) - Scaled (20%)*. URL: https://ourworldindata.org/uploads/2019/05/Transistor-Count-over-time-to-2018.png (besucht am 23. 04. 2020).

Merritt, Rick (27. Aug. 2013). *EETimes - Moore's Law Dead by 2022, Expert Says -*. EETimes. Library Catalog: www.eetimes.com Section: News Analysis. URL: https://www.eetimes.com/moores-law-dead-by-2022-expert-says/ (besucht am 23. 04. 2020).

Microchip Technology Inc. (2020). *DSP Features of the Microchip dsPIC® DSC - Developer Help*. URL: https://microchipdeveloper.com/dsp0201:start (besucht am 01. 06. 2020).

Nowatzki, Tony, Vinay Gangadhar und Karthikeyan Sankaralingam (21. Mai 2019). „Heterogeneous Von Neumann/dataflow microprocessors". In: *Communications of the ACM* 62.6, S. 83–91. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3323923. URL: https://dl.acm.org/doi/10.1145/3323923 (besucht am 04. 07. 2020).

NXP Seminconductors (2020). *i.MX RT600 | NXP*. URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt600-crossover-mcu-with-arm-cortex-m33-and-dsp-cores:i.MX-RT600 (besucht am 16. 06. 2020).

OOURA, Takuya (2006). *General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package*. URL: http://www.kurims.kyoto-u.ac.jp/~ooura/fft.zip (besucht am 18. 07. 2020).

OpenAMP (23. Sep. 2019). *OpenAMP project joins the Linaro Community Projects division*. OpenAMP Project. Library Catalog: www.openampproject.org. URL: https://www.openampproject.org/news/openamp-project-joins-the-linaro-community-projects-division/ (besucht am 15. 07. 2020).

Oshana, Robert (2012). *Expert Guide DSP for Embedded and Real-Time Systems.* OCLC: 812415056. San Diego: Newnes [Imprint] Elsevier Science & Technology Books. ISBN: 978-0-12-386535-9.

PolyCore Software (2020). *PRODUCTS | PolyCore Software, Inc.* Library Catalog: polycoresoftware.com. URL: http://polycoresoftware.com/products (besucht am 15. 07. 2020).

Rupp, Karl (16. Juli 2020). *karlrupp/microprocessor-trend-data*. original-date: 2018-02-15T05:10:17Z. URL: https://github.com/karlrupp/microprocessor-trend-data (besucht am 16. 07. 2020).

Semico Forecasts Strong Growth for RISC-V (25. Nov. 2019). RISC-V International. Library Catalog: riscv.org Section: RISC-V Foundation News. URL: https://riscv.org/2019/11/9679/ (besucht am 25. 07. 2020).

Song, W. J., S. Mukhopadhyay und S. Yalamanchili (März 2016). „Amdahl's law for lifetime reliability scaling in heterogeneous multicore processors“. In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. ISSN: 2378-203X, S. 594–605. DOI: 10.1109/HPCA.2016.7446097.

Tensilica HiFi Audio/Voice DSP IP (2020). URL: https://ip.cadence.com/ipportfolio/tensilica-ip/audio (besucht am 15. 06. 2020).

Terzo, Olivier, Hrsg. (2019). *Heterogeneous computing architectures: challenges and vision*. Boca Raton: Taylor & Francis, a CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa, plc. ISBN: 978-0-367-02344-7.

Texas Instruments (2020a). *C6000 DSP + Arm | Overview | DSP | TI.com*. URL: http://www.ti.com/processors/digital-signal-processors/c6000-dsp-arm/overview.html (besucht am 24. 05. 2020).

– (2020b). *OpenAMP Prototype - Texas Instruments Wiki*. URL: https://processors.wiki.ti.com/index.php/OpenAMP_Prototype (besucht am 16. 07. 2020).

– (2020c). *OpenCL™ Libraries | Texas Instruments*. URL: https://www.ti.com/processors/digital-signal-processors/libraries/OpenCL.html (besucht am 16. 07. 2020).

The Khronos Group (21. Juli 2013). *OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems*. The Khronos Group. Library Catalog: www.khronos.org Section: API. URL: https://www.khronos.org// (besucht am 16. 07. 2020).

The Multicore Association (25. Dez. 2018). URL: https://web.archive.org/web/20181225075936/https://www.multicore-association.org/index.php (besucht am 15. 07. 2020).

Uchiyama, Kunio u. a., Hrsg. (2012). *Heterogeneous multicore processor technologies for embedded systems*. OCLC: 751753295. New York, NY: Springer. 224 S. ISBN: 978-1-4614-0283-1 978-1-4614-0284-8.

Wagner, Philipp (2020). *Produce your own physical chips. For free. In the Open.* FOSSI Foundation. URL: https://fossi-foundation.org/2020/06/30/skywater-pdk (besucht am 25. 07. 2020).

WIPO Global Brand Database (2020a). *85409910 - MTAPI*. URL: https://www3.wipo.int/branddb/en/showData.jsp?ID=USTM.85409910 (besucht am 15. 07. 2020).

– (2020b). *85409915 - MRAPI*. URL: https://www3.wipo.int/branddb/en/showData.jsp?ID=USTM.85409915 (besucht am 15. 07. 2020).

– (2020c). *85409918 - MCAPI*. URL: https://www3.wipo.int/branddb/en/showData.jsp?ID=USTM.85409918 (besucht am 15. 07. 2020).

Wolf, Marilyn (2007). *High-performance embedded computing: architectures, applications, and methodologies*. OCLC: 315412258. Amsterdam: Elsevier/Morgan Kaufmann. 521 S. ISBN: 978-0-12-369485-0.

Woo, Dong Hyuk und Hsien-Hsin S. Lee (Dez. 2008). „Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era". In: *Computer* 41.12, S. 24–31. ISSN: 0018-9162, 1558-0814. DOI: 10.1109/MC.2008.494. URL: https://ieeexplore.ieee.org/document/4712496/ (besucht am 27. 08. 2020).

Wüst, Klaus (2011). *Mikroprozessortechnik: Grundlagen, Architekturen, Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern ; mit 44 Tabellen*. 4., aktualisierte und erweiterte Auflage. Aus dem Programm Elektronik. OCLC: 688619003. Wiesbaden: Vieweg + Teubner. 336 S. ISBN: 978-3-8348-0906-3.

Zahran, Mohamed (2019). *Heterogeneous computing: hardware and software perspectives*. OCLC: 1105289148. ISBN: 978-1-4503-6233-7 978-1-4503-6097-5.

– (21. Feb. 2017). „Heterogeneous computing: here to stay". In: *Communications of the ACM* 60.3, S. 42–45. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3024918. URL: https://dl.acm.org/doi/10.1145/3024918 (besucht am 29. 04. 2020).

Verwendete Entwicklungsumgebung und Emulator

Für diese Arbeit wurde die Entwicklungsumgebung CrossCore® Embedded Studio von Analog Devices mit folgender Version verwendet.

- Product version 2.9.2.0 (w2002252)
- IDE version 2.9.60.202002200746

Als JTAG Emulator wurde der ICE-1000 JTAG-Emulator von Analog Devices verwendet.

Compiler, Assembler und Linker Einstellungen

Folgende Einstellungen wurden in der Entwicklungsumgebung für den Compiler, Assembler und Linker für den jeweiligen Kern ausgewählt.

ARM Kern

- CrossCore ARM Bare Metal Assembler - arm-non-eabi-gcc

```
-c -x assembler-with-cpp -mproc=ADSP-SC582 -msi-revision=1.0 -g -gdwarf-2
    -DCORE0 -D__ADI_FREERTOS -D_DEBUG -DADI_MCAPI @includes-17
   c096d22a83026fc339a9e592b71c36.txt
```

- CrossCore ARM Bare Metal C Compiler - arm-non-eabi-gcc

```
-g -gdwarf-2 -ffunction-sections -fdata-sections -DCORE0 -D__ADI_FREERTOS
    -D_DEBUG -DADI_DEBUG -DADI_MCAPI @includes-
   b822ff0e4fefe547c1573a8ca77aa61b.txt -Wall -c -mproc=ADSP-SC582 -msi-
   revision=1.0
```

- CrossCore ARM Bare Metal C++ Compiler - arm-non-eabi-g++

```
-g -gdwarf-2 -ffunction-sections -fdata-sections -DCORE0 -D__ADI_FREERTOS
    -D_DEBUG -DADI_MCAPI -DADI_DEBUG @includes-17
   c096d22a83026fc339a9e592b71c36.txt -Wall -c -mproc=ADSP-SC582 -msi-
   revision=1.0
```

- CrossCore ARM Bare Metal C++ Linker - arm-non-eabi-g++

```
-mproc=ADSP-SC582 -msi-revision=1.0 -D__ADI_FREERTOS -Wl,--gc-sections -
   Wl,-M=Linker.map -mdebug-libs -lm
```

SHARC-Kern 1

- CrossCore SHARC Assembler - easm21k.exe

```
-file-attr ProjectName="${ProjName}" -proc ADSP-SC582 -si-revision 1.0 -
   D_ADI_THREADS -g -DCORE1 -DADI_MCAPI -DDEBUG @includes-387
   a2c1d3eccc8c505b434fc2ff2b377.txt -swc -char-size-8
```

- CrossCore SHARC C/C++ Compile cc21k.exe

```
-c -file-attr ProjectName="${ProjName}" -proc ADSP-SC582 -flags-compiler
    --no_wrap_diagnostics -si-revision 1.0 -g -D_DEBUG -DCORE1 @includes-
    ebf3b187ff022d70fdbbde7c89f58357.txt -structs-do-not-overlap -no-
    const-strings -no-multiline -warn-protos -threads -double-size-32 -
    char-size-8 -swc -no-simd -DDO_CYCLE_COUNTS
```

- CrossCore SHARC Linker - cc21k.exe

```
-proc ADSP-SC582 -si-revision 1.0 -TC:/Users/User/source/repos/MATE/
    Prj_Core1/system/startup_ldf/app.ldf -no-mem -map Prj_AppB_Core1.map.
    xml -LC:/Users/User/source/repos/MATE/Prj_Core1/system/startup_ldf -
    add-debug-libpaths -threads -flags-link -MDNO_LIBDRV
```

Konfiguration des Systems

Damit die Ergebnisse dieser Arbeit reproduziert und verifiziert werden können, bedarf es einiger Angaben zur Konfiguration des Systems.

Speicherlayout

Tabelle 9.1 zeigt die verwendet Speicherzuordnung.

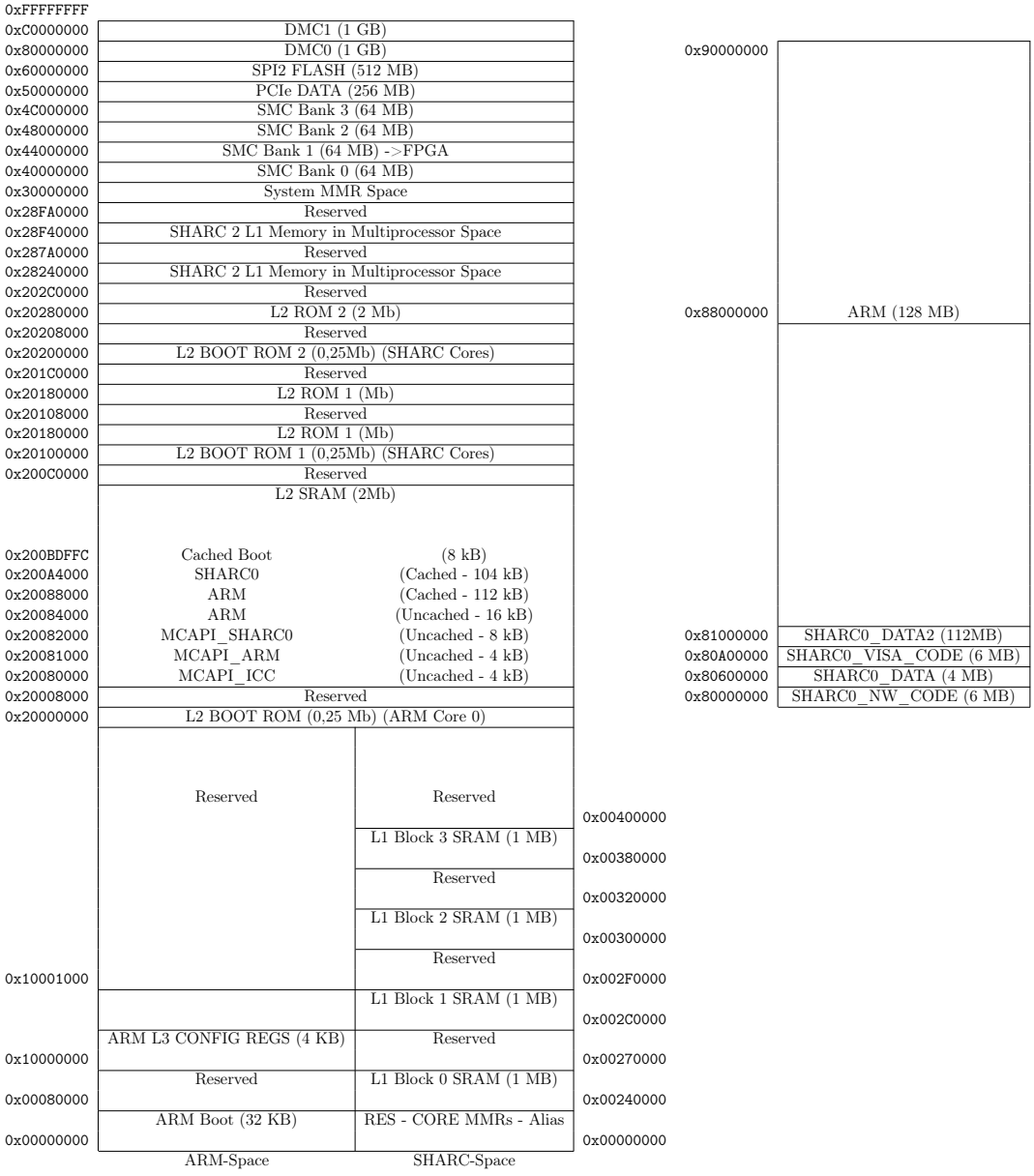


Tabelle 9.1: Verwendetes Speicherlayout des ADSP-SC582

ARM System Configuration - CrossCore® Embedded Studio

Folgende Änderungen wurden bei der Konfiguration des CrossCore® Embedded Studio Projekts vorgenommen.

Heap Konfiguration

- Custom System Heap Size 32 Mega Bytes
- Custom Heap Alignment 4 Bytes

SHARC System Configuration - CrossCore® Embedded Studio

Folgende Änderungen wurden bei der Konfiguration des CrossCore® Embedded Studio Projekts vorgenommen.

Cache Konfiguration

- Instruction Cache 16 Kilo Byte
- PM Cache 16 Kilo Byte
- DM Cache 16 Kilo Byte

Linker Description File

Das Linker Description File für den SHARC+ Kern wurde entsprechend der gezeigten Speicherzuordnung im Anhang in Tabelle 9.1 angepasst.

Konfiguration der Taktsignale

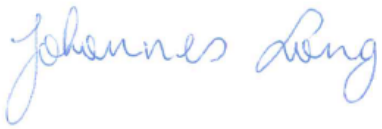
Die Konfiguration der einzelnen Taktsignale kann aus der folgenden Tabelle entnommen werden.

f_{PLLCLK}	400 MHz
$f_{CCLK0/1}$	400 MHz
f_{SYSCLK_0}	200 MHz
f_{SCLK1_0}	200 MHz
f_{FFT_CLK}	200 MHz
f_{DCLK}	25 MHz

Tabelle 9.2: Konfiguration der Taktsignale

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.



Dornbirn, am 31.08.2020 Johannes Lang