technische universität
dortmund

Bachelorarbeit

Vergleichende Studie von Arbitrierungsverfahren für Kommunikationsstrukturen in eingebetteten Multicoresystemen

Tim Harde

18. Februar 2013

Gutachter:
Prof. Dr. Peter Marwedel
Dipl.-Inf. Timon Kelter

Technische Universität Dortmund
Fakultät für Informatik
Eingebettete Systeme (LS-12)
http://ls12-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Eingebettete Systeme sind heutzutage allgegenwärtig. Ob in der Luftfahrt, in Automobilen, der (Fertigungs-)Industrie, der Telekommunikation oder der Unterhaltungselektronik - in all diesen Domänen übernehmen eingebettete Systeme eine Vielzahl von Aufgaben wie Steuerung, Automatisierung, Datenverarbeitung oder Überwachung.

Eine eindeutige und scharfe Definition von eingebetteten Systemen ist aufgrund der großen Anzahl von Anwendungsdomänen schwierig. Häufig werden deshalb relativ allgemeine Definitionen verwendet, die diese Systeme als „informationsverarbeitende Systeme, die in ein größeres Produkt integriert sind“ [13] oder als „in ein umgebendes technisches System eingebettetes und mit diesem in Wechselwirkung stehendes Computersystem“ [4] klassifizieren.

Zwei wesentliche Charakteristika von eingebetteten Systemen sind Zuverlässigkeit und Effizienz [13]. Aufgrund des Einsatzes in der Luft- und Raumfahrt oder im Automotive-Bereich ist es unumgänglich, dass die Zuverlässigkeit dieser Systeme so hoch wie möglich ist. Ein Ausfall oder eine Fehlfunktion könnte in dieser Domäne zu unvorhersagbaren Folgen führen, zu denen unter anderem die Gefährdung von Menschenleben oder hohe Sachschäden zählen können.

Unter der Effizienz versteht man i. A. die Energieeffizienz, die insbesondere bei mobilen Geräten (z. B. aus Telekommunikation oder Unterhaltungselektronik) relevant ist. Neben diesem Faktor existieren aber noch weitere Effizienzkriterien, die bei der Entwicklung eine große Rolle spielen können. So sind beispielsweise die größtmögliche Auslastung von Systemressourcen, die Herstellungskosten oder das Gewicht des fertigen Produktes bei Massenproduktionen zu beachten [13].

Bei eingebetteten Systemen handelt es sich häufig um reaktive Systeme, d. h. solche Systeme, die Zustandsänderungen der Umwelt bzw. in ihrem direkten Umfeld (mit Hilfe von Sensoren) erkennen können und in irgendeiner Form auf diese Zustandsänderungen (mit Hilfe von Aktoren) reagieren. Teilweise existieren bei eingebetteten Anwendungen Zeit-

schranken (sog. Deadlines), innerhalb derer die Reaktion des Systems erfolgen muss. So ist z. B. bei der Auslösung von Airbags nach der Detektion eines Aufpralls die Einhaltung einer solchen Deadline extrem wichtig, damit solche Systeme die größtmögliche Sicherheit für beteiligte Personen garantieren können.
Wie im vorigen Abschnitt erläutert wurde, benötigen eingebettete Systeme in den meisten Fällen ein vorhersagbares Zeitverhalten. Die Analyse und das daraus erwachsende Verständnis dieses Zeitverhaltens für das analysierte System ist notwendig, um die Einhaltung von Deadlines in jedem Fall garantieren zu können. Insbesondere die Bestimmung der *Worst Case Execution Time* (kurz: WCET) [26] - also der Ausführungszeit im ungünstigsten Fall - ist ein wichtiger Schritt bei der Entwicklung solcher reaktiver Systeme.
Neben der WCET ist die *Average Case Execution Time* (kurz: ACET) eine weitere wichtige Kenngröße eines eingebetteten Systems. Durch die Analyse der ACET lassen sich interessante Informationen bei der Entwicklung eines neuen Systems für eine gegebene Anwendung gewinnen. Neben der Bestimmung der durchschnittlichen Performanz der Systeme lassen sich ebenfalls Auslastungsgrenzen bestimmen und die Einhaltung dieser Grenzen sicherstellen. Auf diese Art und Weise wird sichergestellt, dass bereitgestellte Ressourcen nicht ungenutzt bleiben und ggf. unterdimensionierte Komponenten ausfindig gemacht werden können. Die Analyse der ACET und die daraus resultierenden Veränderungen können also zu einer signifikanten Erhöhung der Effizienz eines eingebetteten Systems führen.
In den letzten Jahren ist die Anzahl von eingebetteten Systemen (insbesondere in der Automobilindustrie) extrem gewachsen. So ist es heutzutage in Neuwagen bereits üblich, dass mehr als hundert eingebettete Systeme [4] in Form von Fahrassistenzsystemen (z. B. ABS, ESP, Bremsassistenzsysteme, etc.) die Sicherheit erhöhen oder zusätzlichen Komfort (z. B. durch automatische Klimatisierung, Freisprecheinrichtungen, Rückfahrkameras, etc.) für den Menschen bereitstellen und über im Fahrzeug befindliche Bussysteme (z. B. FlexRay oder CAN-Bus) miteinander kommunizieren. Ein Ende dieser Entwicklung ist dabei nicht abzusehen [3].
Das immense Wachstum in diesem Sektor zeigt, dass die Nachfrage an neueren und leistungsfähigeren Systemen ungebrochen ist. Aufgrund ihrer einfachen Analysierbarkeit (bzgl. der WCET) werden hauptsächlich Singlecore-Systeme eingesetzt, um zeitkritische Anwendungen zu realisieren. Diese gelangen bei der Entwicklung neuer Anwendungen aufgrund ihrer beschränkten Performance allerdings immer mehr an ihre Grenzen und können deshalb die benötigte Rechenleistung nicht mehr bereitstellen.
Es existieren zwar Möglichkeiten, die Performance von Singlecore-Systemen zu erhöhen - konventionelle Methoden zur Performancesteigerung sind beispielsweise die Verwendung einer höheren Taktfrequenz, der Einsatz von out-of-order execution oder die Einführung eines komplexen Pipeliningverhaltens [9] - diese konventionellen Maßnahmen eignen sich aber nur bedingt für eingebettete Systeme. Eine Erhöhung der Taktfrequenz führt aufgrund des quadratischen Zusammenhangs zwischen Taktfrequenz und Energieverbrauch zu einer

deutlichen Verschlechterung der Energieeffizienz [13], teilweise ist die daraus resultierende Erhöhung der Abwärme solcher Geräte ebenfalls problematisch. Der Einsatz von out-of-order execution und komplexem Pipeliningverhalten führt zu sog. *Timinig Anomalies* [19], die die Bestimmung von engen WCET-Schranken erschweren oder unmöglich machen; für solche Systeme kann somit die Einhaltung von Deadlines nicht garantiert werden.
Eine logische Weiterentwicklung zur Erhöhung der Performance ist daher die Entwicklung von Multicore- anstelle von Singlecore-Systemen. Durch die Vervielfachung von Prozessoren bieten Multicore-Systeme gegenüber Singlecore-Systemen viele Vorteile bzgl. Energieeffizienz und Herstellungskosten - durch die Nutzung gemeinsamer Ressourcen (wie z. B. Caches, Speicher, Busse oder I/O-Komponenten) reduzieren sich der Energieverbrauch und die Kosten für solche Systeme enorm.
Trotz der gerade aufgeführten Vorteile sind Multicore-Systeme in einer wesentlichen Hinsicht den Singlecore-Systemen unterlegen: aufgrund der konkurrierenden Zugriffe auf gemeinsame Ressourcen sind solche Systeme schwierig zu analysieren. Ein solcher konkurrierender Zugriff entsteht dann, wenn zwei unabhängige Master zum gleichen Zeitpunkt auf eine gemeinsame Ressource (wie z. B. Speicher oder Bus) zugreifen wollen. Diese Zugriffskonflikte müssen dann - durch einen sog. *Arbiter* - in eine eindeutige Reihenfolge gebracht werden. Diese Reihenfolge wird durch sog. *Arbitrierungsverfahren* festgelegt, welches damit direkten Einfluss auf die ACET und WCET der Anwendung hat und die Analyse (aufgrund von teilweise hochgradigem Nichtdeterminismus) somit extrem komplex wird. Gerade im Bereich der eingebetteten Systeme ist eine solche Analysierbarkeit aber unumgänglich, wenn es um die oben genannte Einhaltung harter Deadlines geht. Derzeit existieren keine generischen Analysemethoden, um ausreichend enge Schranken für die WCET bei zeitlich vorhersagbaren Multicore-Systemen zu bestimmen. Um diese Probleme zu beherrschen und die Analysierbarkeit zu verbessern, existieren zwei naheliegende Lösungsansätze: die Duplikation von gemeinsamen Ressourcen und die Reduktion von konkurrierenden Zugriffen.
Eine Duplikation von (gemeinsamen) Ressourcen bietet zwei offensichtliche Nachteile. Zum einen steigt der Energieverbrauch des gesamten Systems, da sich die Plattform wieder in Richtung von mehreren unabhängigen Singlecore-Plattformen entwickelt; dieser naheliegende Ansatz hat also direkte negative Auswirkungen auf die Energieeffizienz solcher Systeme. Des Weiteren wird durch die Duplikation die Auslastung der betroffenen Komponenten deutlich herabgesetzt.
Der zweite Ansatz - die Reduktion von konkurrierenden Zugriffen - erfolgt durch den Einsatz von geeigneten Arbitrierungsverfahren. Dabei wird im Prinzip die Zugriffszeit auf eine gemeinsame Ressource partitioniert, so dass jeder Master nach Möglichkeit garantierte Zeitfenster zur Verfügung hat, in denen er exklusiv über die Ressource verfügen kann. Ein Arbitrierungsverfahren, das konkurrierende Zugriffe durch die Partitionierung der Zugriffszeit realisiert, ist beispielsweise *Time Division Multiple Access* (kurz: TDMA) [20]. Die Verwendung solcher Arbitrierungsverfahren hat aber meistens negative Auswirkungen

auf die Auslastung der Systeme, da durch die garantierten Zugriffszeiten die Ressourcen ungenutzt bleiben, wenn ein Master in dem ihm zugeordneten Zeitfenster eine gemeinsame Ressource nicht verwendet.
Nach dem derzeitigen Forschungsstand ist der Einfluss von konkreten Arbitrierungsverfahren in Bezug auf zeitlich vorhersagbare Multicore-Systeme auf die durchschnittliche Auslastung unklar und der Einfluss auf die ACET auf realitätsnahen Plattformen nicht näher untersucht.
Ziel dieser Arbeit ist es herauszufinden, wie schlecht oder gut die durchschnittliche Auslastung solcher Systeme unter der Verwendung von ausgewählten Arbitrierungsverfahren wirklich ist. Zu diesem Zweck wird eine experimentelle Bestimmung der ACET und ein Vergleich von unterschiedlichen Arbitrierungsverfahren (Faire Arbitrierung, prioritätsbasierte Arbitrierung, TDMA und Priority Division) in einer Simulationsumgebung durchgeführt.

1.2 Ziele der Arbeit

Wie im vorigen Absatz erläutert wurde, soll innerhalb dieser Arbeit eine Auswertung der ACET und möglichen Auslastung in zeitlich vorhersagbaren Multicore-Systemen erfolgen. Im Folgenden sind die notwendigen Schritte dargestellt und die einzelnen Teilaufgaben kurz erläutert.

- **Erstellung einer Simulationsplattform**

  Um die für die Analyse notwendigen Daten experimentell bestimmen zu können, wird zunächst eine möglichst flexible Simulationsplattform erstellt. Im Rahmen der Arbeit wird je eine Plattform mit 1, 2, 4 und 8 Kernen für die späteren Benchmarks erstellt.

- **Anpassung der Infrastruktur des Compilerframeworks**

  Um die erstellte Simulationsplattform möglichst komfortabel konfigurieren und einsetzen zu können, wird eine direkte Schnittstelle zu einem bestehenden Compilerframework geschaffen. Die notwendigen Änderungen bzw. Erweiterungen werden im Rahmen dieser Arbeit durchgeführt.

- **Erstellung von automatisierten Benchmarks**

  Um ausreichende Daten für eine spätere Analyse zu sammeln, muss eine Vielzahl von Benchmarks ausgeführt und die so erzeugten Ergebnisse konsolidiert werden. Hierzu soll ein Mechanismus geschaffen werden, um die notwendigen Experimente möglichst automatisch generieren und ausführen zu können.

- **Einfluss von Arbitrierungsverfahren auf die ACET**

Aus den gesammelten Daten soll nach entsprechender Analyse der Einfluss der vier ausgewählten Arbitrierungsverfahren auf die ACET und die Auslastung der verwendeten Busse bestimmt werden.

1.3 Aufbau der Arbeit

Im zweiten Kapitel erfolgt eine Erläuterung der notwendigen Grundlagen, die für das Verständnis der Arbeit benötigt werden. Dies umfasst eine Vorstellung der verwendeten Arbitrierungsverfahren sowie einige grundlegende Erläuterungen zu Performance-Metriken. Im dritten Kapitel erfolgt die Vorstellung der Simulationsplattformen und der darin verwendeten *Arbitration Bridge* als zentrale Komponente, die den Zugriff der einzelnen Cores auf den gemeinsamen Speicher steuert. Anschließend erfolgt eine Beschreibung der Integration in das bestehende Compiler-Framework, eine Vorstellung der verwendeten Benchmark-Suiten sowie die daraus zusammengestellten Multicore-Benchmarks. Im sechsten Kapitel werden die gesammelten Ergebnisse dargestellt und analysiert. Das siebte und zugleich letzte Kapitel umfasst die Zusammenfassung der Ergebnisse und gibt einen Ausblick auf weitere Arbeiten.

Kapitel 2

Grundlagen

Für das Verständnis der Arbeit sind einige Grundlagen notwendig, die in diesem Kapitel kurz vorgestellt werden sollen. In Kapitel 2.1 werden zunächst Hintergrundinformationen zur Arbitrierung im Allgemeinen vermittelt und anschließend die unterschiedlichen verwendeten Arbitrierungsverfahren genauer vorgestellt. In Kapitel 2.2 werden anschließend die grundlegenden Performance-Metriken aufgeführt sowie kurz auf typische Analysemethoden zur Bestimmung dieser Metriken eingegangen.

2.1 Arbitrierungsverfahren

Wie in der Einleitung bereits beschrieben wurde, bietet die Verwendung von gemeinsamen Ressourcen in Multicore-Systemen immenses Potenzial für die Erhöhung von Energieeffizienz und Performance von eingebetteten Systemen. Eingangs wurde ebenfalls bereits auf die Problematik von zeitgleichen (oder zumindest überlappenden) und somit konkurrierenden Zugriffen durch unterschiedliche Master auf eine gemeinsame Ressource sowie die Notwendigkeit von Arbitrierungsverfahren für die Handhabung dieser Konflikte hingewiesen.

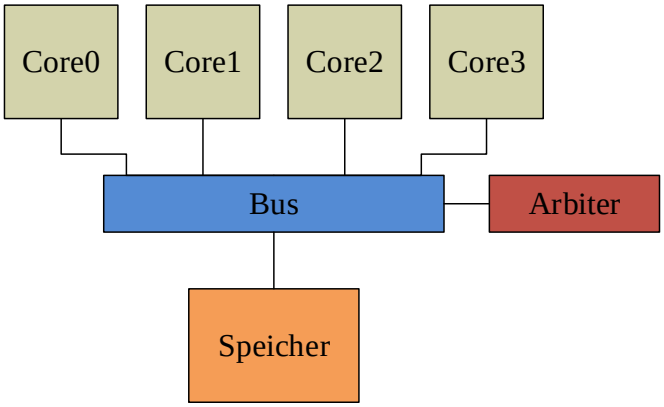


Abbildung 2.1: Schematische Darstellung eines Multicore-Systems.

Abbildung 2.1 zeigt zunächst den schematischen Aufbau eines einfachen Multicore-Systems bestehend aus vier Cores, die über einem gemeinsamen Bus auf einen gemeinsamen Speicher zugreifen. Durch diese gemeinsamen Ressourcen können also konkurrierende Zugriffe auf die jeweiligen Komponenten entstehen. Eventuell auftretende Konflikte zwischen den einzelnen Cores werden durch den *Arbiter* unter Verwendung eines geeigneten *Arbitrierungsverfahrens* aufgelöst.
Ein solches Arbitrierungsverfahren hat also die Aufgabe, überlappende und somit konkurrierende Zugriffe auf eine gemeinsame Ressource zu sequenzialisieren und somit in eine konkrete Reihenfolge zu bringen. Beispiel 2.1.1 zeigt die Anwendung eines einfachen Arbitrierungsverfahrens.

2.1.1 Beispiel. Dieses Beispiel basiert auf dem System aus Abbildung 2.1. Die einzelnen Cores führen Instruktionen I1 bis I6 aus, die jeweils einen Zugriff auf den gemeinsamen Speicher symbolisieren (siehe Abbildung 2.2).

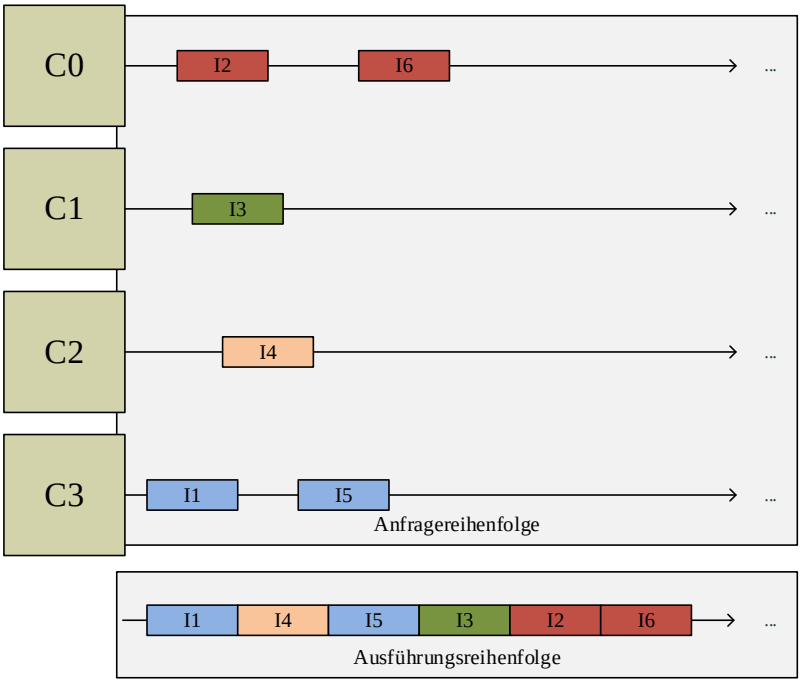


Abbildung 2.2: Anwendung eines einfachen Arbitrierungsverfahrens.

Das hier verwendete Arbitrierungsverfahren bevorzugt immer den Core mit der höchsten Core-ID. Bei eventuell konkurrierenden Zugriffsanfragen werden die Cores in absteigender Reihenfolge bedient (C3, C2, C1, C0), die Zugriffsreihenfolge sequenzialisiert und die entstandenen Konflikte aufgelöst.

Im Prinzip handelt es sich bei Arbitrierungsverfahren um ein *Scheduling-Verfahren* zur Zuteilung einer gemeinsamen Ressource. Im Allgemeinen unterscheidet man zwischen zwei Typen von Scheduling-Verfahren: Online- und Offline-Scheduling.

Beim *Offline-Scheduling* verfügt man im Vorhinein über ein detailliertes Wissen über die auszuführenden Anwendungen. Dieses Wissen kann dazu eingesetzt werden, um eine fixe und durch das System zur Laufzeit unveränderliche Zuteilung der gemeinsamen Ressource zu erstellen, die den jeweiligen Anforderungen der Anwendungen genügt.
Beim *Online-Scheduling* hingegen ist die Ausgangssituation weniger komfortabel. Häufig sind der Programmablauf und die daraus resultierenden Zugriffe auf eine gemeinsame Ressource abhängig von äußeren Einflüssen (beispielsweise bei reaktiven Systemen), so dass die Erstellung eines Offline-Schedules nicht möglich ist. In diesem Szenario erfolgt die Zuteilung der Ressource auf Anforderung (On-Demand) eines Masters.
Bei dem in 2.1.1 vorgestellten Verfahren zeigt sich allerdings ein Problem, das im Allgemeinen als *Verhungern* bezeichnet wird. Handelt es sich bei den Anwendungen, die auf den Cores mit den höchsten Core-IDs ausgeführt werden, um Anwendungen mit sehr vielen Speicherzugriffen, so kann es passieren, dass Cores mit niedrigen Core-IDs nie einen Speicherzugriff durchführen können, da der Bus bereits voll ausgelastet ist. Dieses Verhalten wird im folgenden Beispiel (siehe Beispiel 2.1.2) näher erläutert.

2.1.2 Beispiel. Auf den Cores C2 und C3 werden Anwendungen zur Ausführung gebracht, die eine relativ hohe Anzahl an Speicherzugriffen benötigen. Aufgrund der „Bevorzugung“ der Anfragen von C2 bzw. C3 lasten diese Anfragen den gemeinsamen Bus bereits komplett aus, so dass die Cores C0 bzw. C1 keine Speicherzugriffe tätigen können, da ihnen die gemeinsame Ressource vom Arbiter nicht zugeteilt wird (siehe Abbildung 2.3).

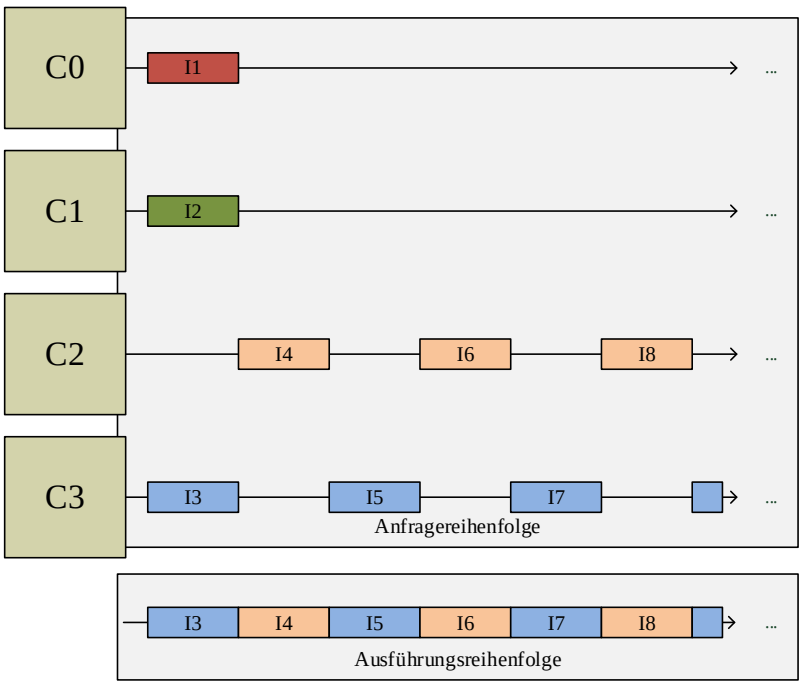


Abbildung 2.3: Verhungern der Cores C0 und C1.

Dieses Phänomen kann, wenn die Auslastung des jeweiligen Systems (bzw. in diesem Fall die Auslastung der gemeinsamen Ressource durch die jeweiligen Cores) bekannt ist, ausgeschlossen werden. In der Scheduling-Theorie existieren Analysemethoden, um die „Schedulability“ eines gegebenen Problems festzustellen und auch im Falle des Einsatzes eines Online-Schedulingverfahrens zu garantieren.
Neben dem in Beispiel 2.1.1 vorgestellten Arbitrierungsverfahren gibt es unzählige weitere Ansätze für die Zuteilung einer gemeinsamen Ressource [18]. Diese Verfahren unterscheiden sich maßgeblich in der Analysierbarkeit (durch ggf. vorhandene garantierte Zugriffszeiten für bestimmte Master, beschränkte maximale Verzögerungen von Zugriffen, etc.), Performance und Fairness. Für diese Arbeit wurden vier prototypische, und in Bezug auf ihre Realzeitfähigkeit interessante, Verfahren ausgewählt. Diese Arbitrierungsverfahren sollen in den folgenden vier Abschnitten genauer vorgestellt werden.

2.1.1 Faire Arbitrierung (FA)

Die *Faire Arbitrierung* (engl.: Fair Arbitration, kurz: FA) ist eine Form der Arbitrierung, in der die gemeinsame Ressource durch eine Art Rundlaufverfahren (engl.: Round-Robin, kurz: RR) an die konkurrierenden Master zugeteilt wird.
FA ist dabei ein extrem einfaches Verfahren, das eine möglichst faire Verteilung einer gemeinsamen Ressource ermöglicht. Diese „Fairness“ kommt im Falle einer hohen Frequentierung der Ressource durch alle beteiligten Master durch eine sehr gleichmäßige Verteilung der Ressource zu Stande. Ein weiterer großer Vorteil der FA ist die Tatsache, dass einzelne Master nicht verhungern können.

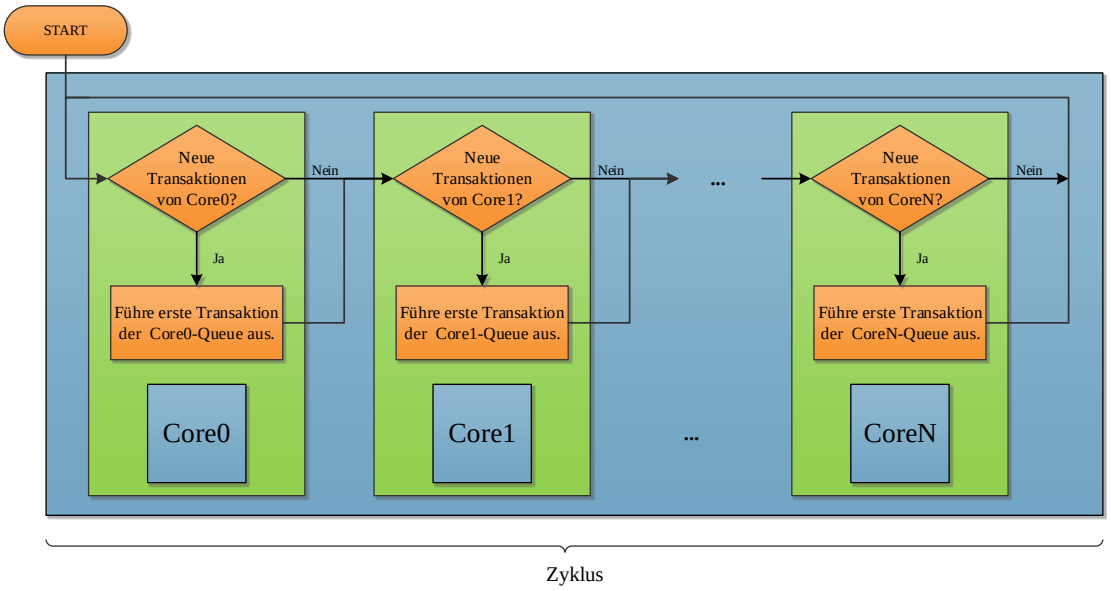


Abbildung 2.4: Scheduling mit FA unter Verwendung von FIFO-Warteschlangen für jeden Bus-Master.

Ihren Ursprung haben die sog. Round-Robin-Verfahren im Scheduling von Prozessen (hier wird eine gemeinsame Ressource den einzelnen Prozessen jeweils für vorher definierte Zeitschlitze zur Verfügung gestellt und danach wieder entzogen) oder in der Lastverteilung (z. B. werden mehrere Webserver-Anfragen durch einen sog. Load-Balancer auf unterschiedliche Zielserver nach einem Round-Robin-Verfahren verteilt). In Bezug auf Kommunikationsstrukturen in eingebetteten Multicore-Systemen kann die FA zur Arbitrierung eines gemeinsamen Busses eingesetzt werden, den sich mehrere Cores zum Zugriff auf einen gemeinsamen Speicher teilen. Hierbei erfolgt die Zuteilung der Ressource (in diesem Fall der gemeinsame Bus) immer für die Dauer eines Speicherzugriffes, bevor der nächste Master einen anderen Request absetzen kann. Beispiel 2.1.3 illustriert die Anwendung des Verfahrens.
Die Anwendung von FA basiert auf Zyklen (siehe Abbildung 2.4). Innerhalb eines jeden Zyklus erhält jeder Core die Möglichkeit, eine Transaktion auf dem gemeinsamen Bus auszuführen - die Reihenfolge der zyklischen Abarbeitung der Master ist dabei immer identisch. Jeder Master verfügt dabei über eine eigene FIFO-Warteschlange, in der Anfragen des jeweiligen Masters gesammelt werden.

2.1.3 Beispiel. Die vier Cores führen Instruktionen I1 bis I11 aus, die erneut Operationen zum Zugriff auf den gemeinsamen Speicher symbolisieren (siehe Abbildung 2.5).

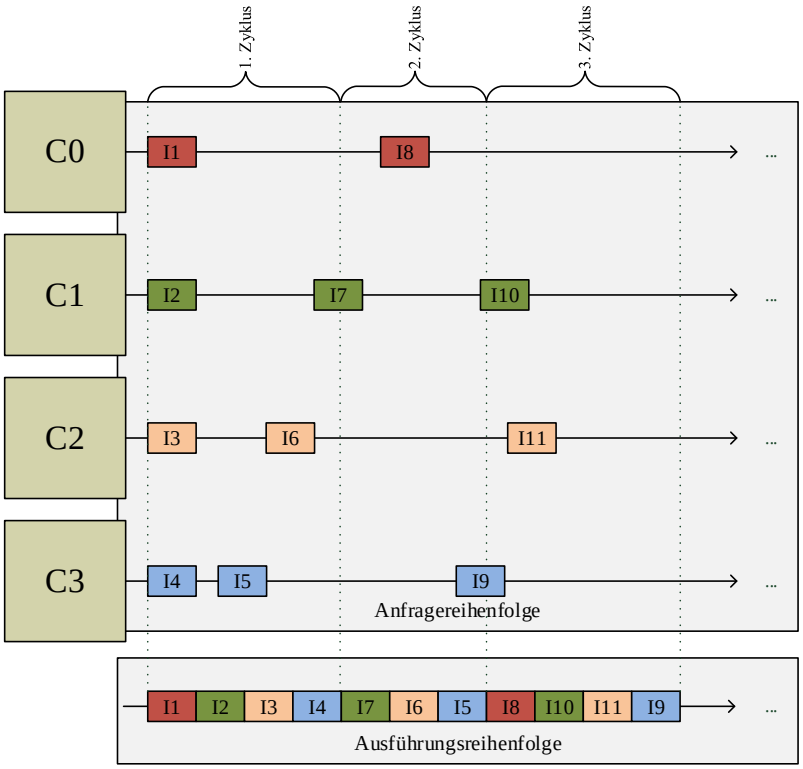


Abbildung 2.5: Anwendung von Fair Arbitration.

Während des ersten Zyklus führt jeder Core einen Speicherzugriff aus. Da die Instruktion I8 von C0 zu Beginn des zweiten Zyklus (also zu dem Zeitpunkt, zu dem C0 die gemeinsame Ressource zugeteilt werden würde) dem Arbiter noch nicht bekannt ist, erhält C0 innerhalb dieses Zyklus keine Möglichkeit mehr einen Speicherzugriff auszuführen - stattdessen wird die Ausführung von I8 bis zum Beginn des dritten Zyklus verschoben. Im dritten Zyklus führt schließlich wieder jeder Core eine Transaktion aus.
Nichtsdestotrotz kann die Zuteilung einer Ressource mit FA zu einer gewissen Unfairness führen. Problematisch wird es genau dann, wenn es bei der Beanspruchung der Ressource im Falle von atomaren Operationen extreme Unterschiede in den Ausführungszeiten existieren. Genau dieses Problem wird in Beispiel 2.1.4 erläutert.

2.1.4 Beispiel. Die vier Cores führen die Instruktionen I1 bis I8 aus (siehe Abbildung 2.6). Während die Zuteilung der gemeinsamen Ressource an die Cores C0, C1 und C2 immer nur relativ kurze Zeit andauert, wird diese von C3 durch die deutlich größeren Ausführungszeiten wesentlich länger gehalten. Die Instruktionen I4 und I6 haben jeweils die gleiche Ausführungszeit wie I1, I2 und I3 bzw. I5, I7 und I8 zusammen - somit ergibt sich eine gewisse „Unfairness“, da die Ressource in 50% der Zeit exklusiv durch den Core C3 genutzt wird.

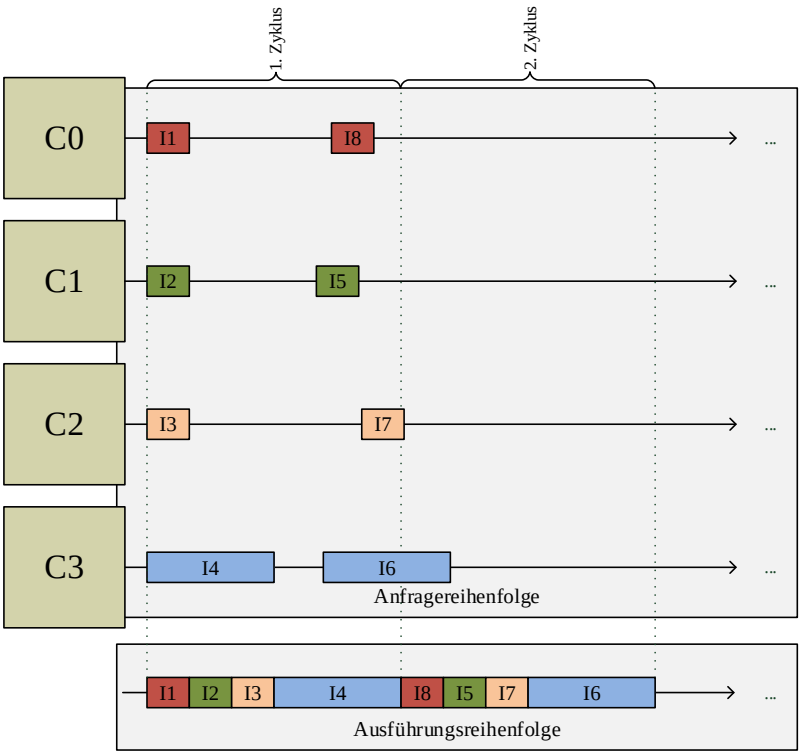


Abbildung 2.6: Unfairness von Fair Arbitration.

Dieses Problem der ungleichmäßigen Auslastung einer gemeinsamen Ressource durch unterschiedliche Master kann durch eine Gewichtung der Cores beherrscht werden. Dies ge-

schieht durch den Einsatz eines sog. gewichteten Round-Robin-Verfahrens (engl.: Weighted Round-Robin, kurz: WRR) [21]. Durch die unterschiedlichen Gewichtungen kann Cores mit durchschnittlich kurzen Ausführungszeiten von Instruktionen eine Ressource häufiger zugeteilt werden, die somit gleichmäßiger und fairer verteilt wird.

2.1.2 Time Division Multiple Accsess (TDMA)

Das zweite hier vorgestellte Arbitrierungsverfahren ist *Time Division Multiple Access* (kurz: TDMA). Hierbei handelt es sich um ein Zeitmultiplexverfahren, dass die Zugriffszeit auf eine gemeinsame Ressource in sog. *Zeitschlitze* (engl.: Time Slices oder Time Slots) partitioniert; diese Zeitschlitze sind dann den jeweiligen Mastern zugeordnet. Die einzelnen Slots sind innerhalb eines *TDMA-Rahmens* (engl.: TDMA-Frame) angeordnet. Auf diese Art und Weise erhält jeder Master innerhalb eines TDMA-Frames (siehe Abbildung 2.7) eine garantierte Zugriffszeit.

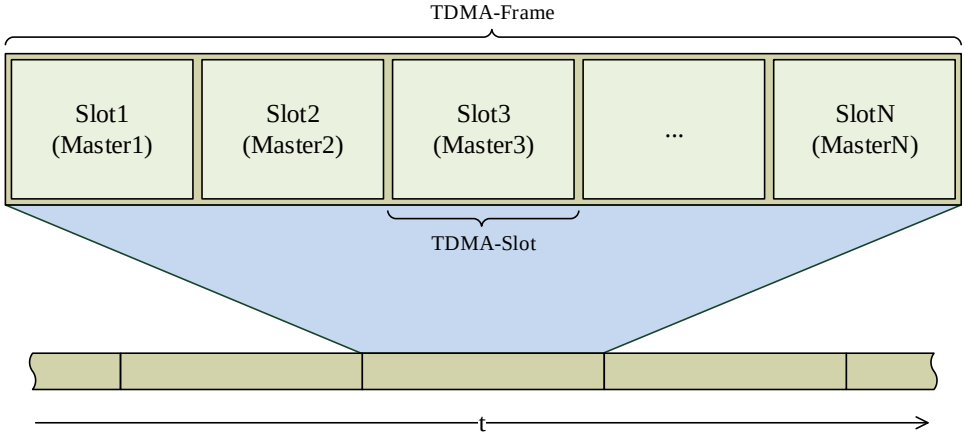


Abbildung 2.7: Darstellung eines TDMA-Frames.

Beim klassischen TDMA wird für jeden Slot eine einheitliche Länge verwendet. Jeder dieser Slots wird dann einem Master zugeordnet und ist exklusiv durch diesen Master nutzbar. TDMA ist ein Zugriffsverfahren, das in eingebetteten Systemen mit harten Echtzeitanforderungen häufig eingesetzt wird [15]. Aufgrund der fixen Unterteilung in Slots ergibt sich eine sehr gute Analysierbarkeit, was die Bestimmung und Einhaltung von WCET-Schranken sehr vereinfacht. Diese Analysierbarkeit basiert auf einer garantierten maximalen Wartezeit und einer fixen und garantieren Bandbreite, die jedem Master zur Verfügung gestellt wird.
Einer der größten Nachteile von TDMA ist die möglicherweise schlechte Auslastung der gemeinsamen Ressource. Da die einzelnen Master nur innerhalb ihrer eigenen Zeitschlitze Anfragen absetzen können und somit Zeitfenster ggf. komplett ungenutzt bleiben, wenn der jeweilige Master keinen Zugriff tätigt, kann die Auslastung gegenüber anderen Arbitrierungsverfahren extrem einbrechen (siehe Beispiel 2.1.5).

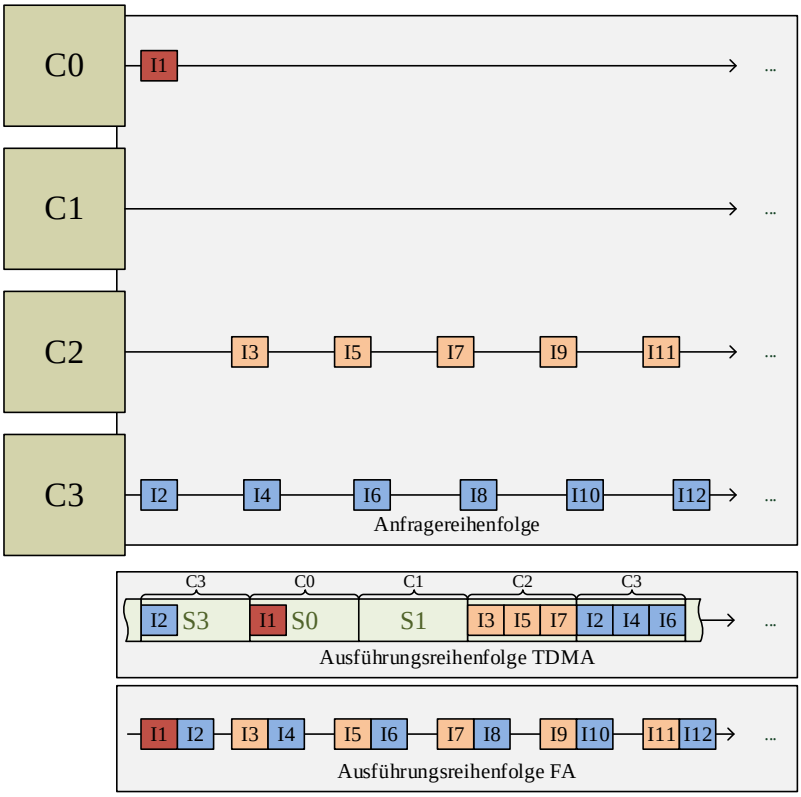


Abbildung 2.8: Vergleich der Auslastung und Ausführungsreihenfolge zwischen TDMA (oben) und FA (unten).

2.1.5 Beispiel. Die vier Slots S0, S1, S2 und S3 sind den einzelnen Cores C0, C1, C2 und C3 zugeordnet (siehe Abbildung 2.8). Der Slot S1 bleibt komplett ungenutzt, da der zugeordnete Core C1 keine Anfrage zur Nutzung der gemeinsame Ressource stellt. Die Auslastung von S0 ist ebenfalls sehr gering, da vom zugeordneten Core lediglich eine einzige Anfrage gestellt wird und somit der Slot S0 nach Ausführung der Instruktion I1 ungenutzt bleibt. C2 und C3 können bei der Verwendung von FA deutlich mehr Instruktionen ausführen, somit ist die Auslastung der gemeinsamen Ressource mit diesem Arbitrierungsverfahren deutlich höher.

Ein weiterer Grund für die potenziell schlechte Auslastung bei TDMA ist die Tatsache, dass Anfragen unter keinen Umständen in den nächsten Slot (eines anderen Masters) „hineinragen“ dürfen. Da Speicheroperationen aber (z. B. durch die Verwendung von Caches) unterschiedliche Laufzeiten haben können, muss die maximale Ausführungszeit einer Transaktion bekannt sein und bei der Arbitrierung der Ressource zu Grunde gelegt werden, um genau dieses Verhalten sicherzustellen. Durch eben dieses Zugrundelegen der maximalen Ausführungszeit kann es aber vorkommen, dass eine Transaktion unnötig verzögert wird, da die tatsächliche Ausführungszeit im Vorhinein nicht bekannt ist. Beispiel 2.1.6 illustriert dieses Problem und die negativen Auswirkungen auf die Auslastung.

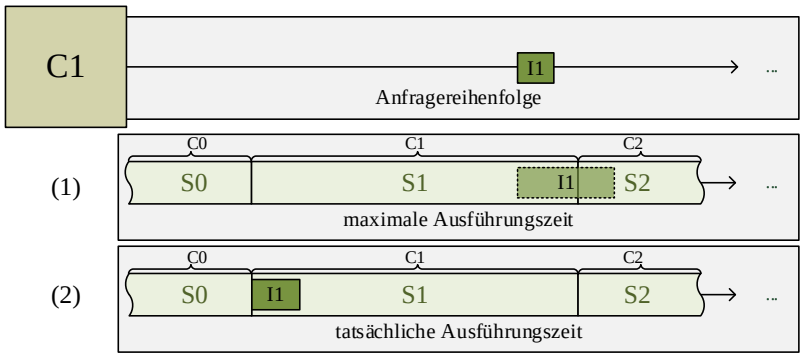


Abbildung 2.9: Problematik der künstlichen Verzögerung von Transaktionen bei TDMA.

2.1.6 Beispiel. In diesem Beispiel wird lediglich die Instruktion I1 des Cores C1 betrachtet (siehe Abbildung 2.9). (1) zeigt dabei die maximale Ausführungszeit der Transaktion I1 bei sofortiger Zuteilung der gemeinsamen Ressource. Da bei der verzögerungsfreien Ausführung der Transaktion aber nicht garantiert werden kann, dass der Zugriff nicht in den nächsten Slot S2 (von C2) hineinragt, wird die Transaktion bis in den nächsten Frame verzögert. Hier wird ersichtlich, dass die tatsächliche Ausführungszeit weit unter der maximalen Ausführungszeit geblieben ist (2) und eine sofortige Ausführung problemlos möglich gewesen wäre, ohne die Deadline (Anfang von S2) zu überschreiten.

Insgesamt ist TDMA ein Arbitrierungsverfahren, das seine extremen Vorteile bei der Bestimmung von möglichst engen WCET-Schranken durch die einfache Analysierbarkeit und das vorhersagbare Zeitverhalten zu Gunsten von relativ schlechter Auslastung erkauft. Nichtsdestotrotz gibt es häufig keine Alternative zur Anwendung von TDMA, wenn es um zeitkritische Anwendungen und die Einhaltung von harten Zeitschranken in eingebetteten Systemen geht.

2.1.3 Prioritätsbasierte Arbitrierung (PA)

Bei der *prioritätsbasierten Arbitrierung* (engl.: Priority Arbitration, kurz: PA) wird ein gegenüber FA und TDMA völlig anderer Ansatz verwendet. Dieses Arbitrierungsverfahren basiert auf eindeutigen Prioritäten, die den einzelnen Mastern zugeordnet werden. Bei konkurrierenden Zugriffen erhält immer der Master mit der jeweils höchsten Priorität den Zugriff auf die gemeinsame Ressource.
PA ist bei mehreren Anwendungen mit harten Deadlines und der Verwendung einer oder mehrerer gemeinsamer Ressourcen nur eingeschränkt einsetzbar, da die Möglichkeit besteht, dass einzelne Master verhungern. Generell gibt es zwei unterschiedliche Ansätze: PA mit und ohne *Unterbrechungen*.
PA mit Unterbrechungen kann beispielsweise beim Scheduling von Prozessen angewendet werden. Notwendige Bedingung ist dabei, dass die gemeinsame Ressource (in diesem Fall

der Prozessor) dem jeweiligen Master entzogen werden kann, falls dies notwendig werden sollte. Nach Abarbeitung einer höher priorisierten Aufgabe (z. B. im Falle eines Interrupts das daraus resultierende Interrupt-Handling) kann dann mit der Ausführung der niedriger priorisierten Aufgabe fortgefahren werden. Durch das Entziehen der Ressource ist es durchaus möglich, dass die unterbrochene Anwendung nicht fortgesetzt werden kann und somit von Beginn an abgearbeitet werden muss.
Im Falle von Kommunikationsstrukturen in eingebetteten Multicore-Systemen ist eine Realisierung von Unterbrechungen allerdings nicht möglich, da die entsprechende Ressource - in diesem Fall der Bus bei einem Zugriff auf den gemeinsamen Speicher - nicht entzogen werden kann. Die Ursache hierfür ist die Tatsache, dass es sich bei einem einzelnen Speicherzugriff eines Cores um eine atomare und somit nicht unterbrechbare Operation handelt. Das folgende Beispiel (siehe 2.1.7) illustriert den Unterschied von PA mit und ohne Unterbrechungen.
Anmerkung: In dem hier betrachteten Modell werden sog. *Split Transactions*“ nicht unterstützt. Bei diesen Transaktionen handelt es sich um Transaktionen, die den Bus nur zum Absetzen der eigentlichen Anfrage und zum Empfangen des Ergebnisvektors belegen; in der Zwischenzeit bleibt der Bus frei und kann durch andere Master verwendet werden. Da in dem gewählten Modell die Laufzeiten der einzelnen Speicherzugriffe relativ kurz sind, ist die Verwendung von Split Transactions nicht notwendig.

2.1.7 Beispiel. Den vier Mastern sind unterschiedliche Prioritäten (C1: 1, C2: 2, C3: 3 und C4: 4) zugeordnet (siehe Abbildung 2.10). Im Falle von PA mit Unterbrechungen (1) wird die Instruktion I2 durch die Instruktion I4 unterbrochen, da dem Core C3 eine höhere Priorität zugeordnet ist. Nachdem die Abarbeitung von I4 abgeschlossen ist, wird mit der Ausführung der unterbrochenen Instruktion I2 fortgefahren. Die Instruktion muss aufgrund der Unterbrechung komplett neu ausgeführt werden.
Fall (2) zeigt die Ausführungsreihenfolge der Instruktionen unter Verwendung von PA ohne Unterbrechungen. Mit der Abarbeitung der höher priorisierten Instruktion I4 wird erst nach Abschluss der Instruktion I2 fortgefahren, eine Unterbrechung findet nicht statt.

PA verfügt insgesamt über eine sehr hohe Busauslastung, eignet sich aber aufgrund der schwierigen Analysierbarkeit und gegebenenfalls nicht vorhandenen Realzeitfähigkeit nur bedingt für die Arbitrierung von gemeinsamen Ressourcen in eingebetteten Systemen, wenn es um die Einhaltung von WCET-Schranken geht. PA kann genau dann eingesetzt werden, wenn nur ein Master Realzeitanforderungen hat - diesem Master wird in diesem Fall die höchste Priorität zugeordnet - oder wenn die maximale Last pro Master bekannt ist, so dass die Schedulability der unterschiedlichen Tasks garantiert werden kann [18].
Beim Online-Scheduling ohne bekannte Lastgrenzen der einzelnen Master verfügt lediglich der Master mit der höchsten Priorität über ein vorhersagbares Zeitverhalten. Dieses Zeitverhalten kann im Falle von unterbrechbaren Operationen exakt vorhergesagt werden,

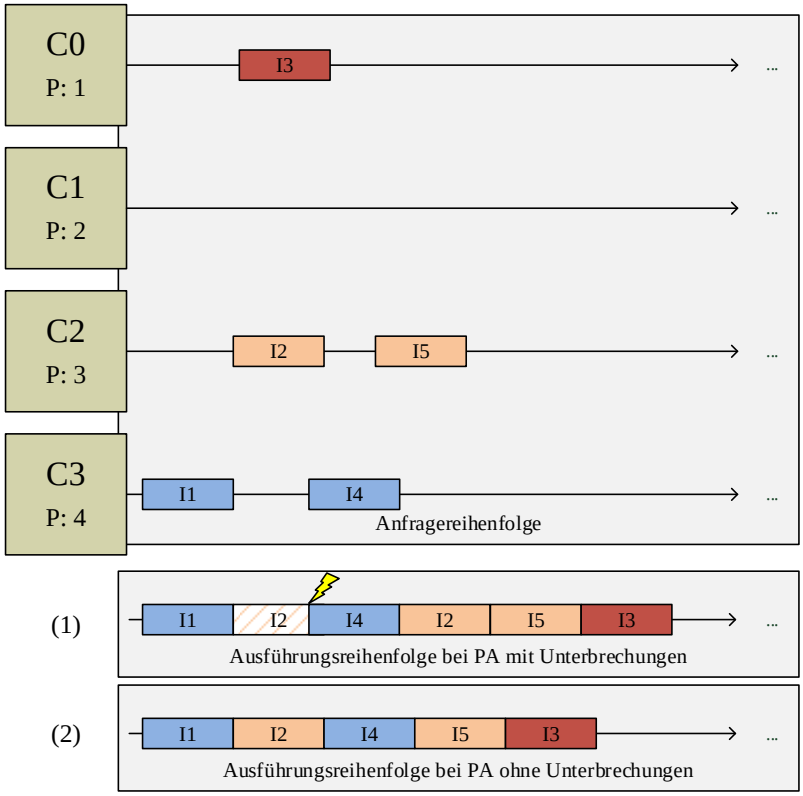


Abbildung 2.10: Prioritätsbasierte Arbitrierung mit (1) und ohne (2) Unterbrechungen.

während es sich bei nicht unterbrechbaren Operationen lediglich einigermaßen exakt bestimmen lässt. Für andere Master mit niedrigeren Prioritäten kann in diesem Fall aber aufgrund der nicht vorhandenen Fairness keine Zugriffszeit garantiert werden.

2.1.4 Priority Division (PD)

Priority Division (kurz: PD) ist eine Erweiterung von TDMA, welche versucht, die Vorteile TDMA und PA zu vereinen bzw. bestehende Nachteile abzuschwächen oder zu beseitigen [21]. Durch die Veränderungen soll ein Arbitrierungsverfahren entstehen, das sowohl ausreichend gut analysierbar ist, um möglichst enge WCET-Schranken garantieren zu können als auch über eine deutlich höhere Auslastung als reines TDMA ermöglicht.

Grundlage hierfür ist - wie bei normalem TDMA - die Partitionierung der Zugriffszeit auf die gemeinsame Ressource in Zeitschlitze identischer Länge. Anders als bei normalem TDMA erfolgt jedoch keine 1:1-Zuordnung zwischen den unterschiedlichen Mastern und Slots - stattdessen erhält jeder Core in jedem Slot eine eigene, statische Priorität. Um außerdem noch verlässliche Reaktionszeiten und maximale Verzögerungen garantieren zu können, erhält jeder Master in mindestens einem Slot des *PD-Frames* (siehe Abbildung 2.11) die höchste Priorität.

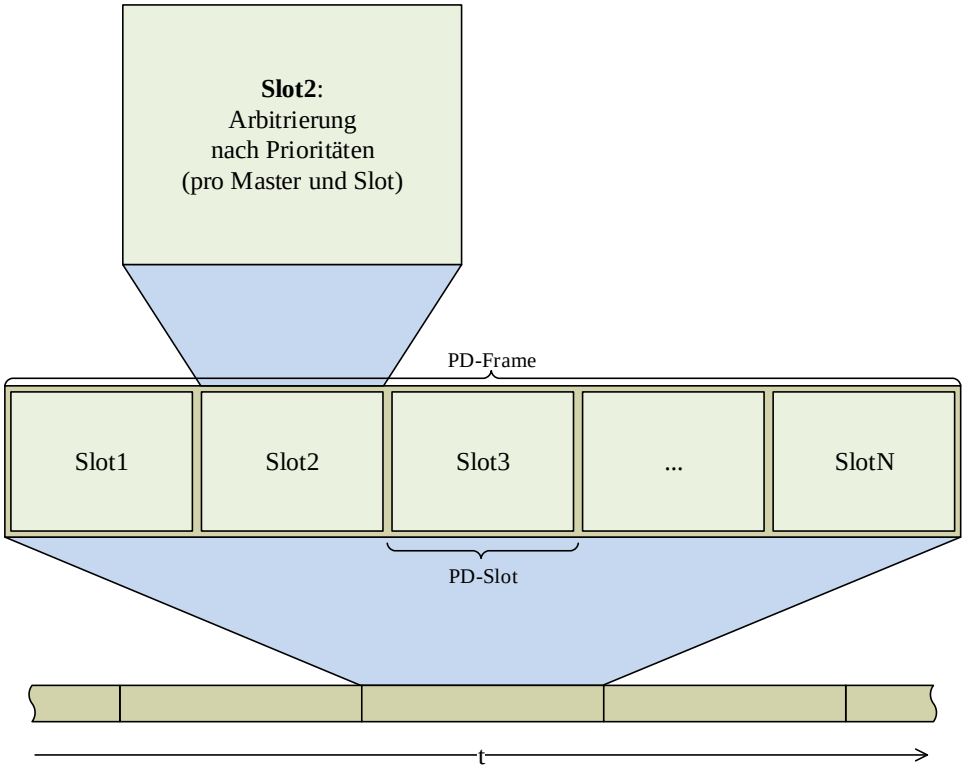


Abbildung 2.11: Darstellung eines PD-Frames.

Dieser Aufbau des Frames bietet einige Vorteile: Zum einen kann sichergestellt werden, dass keiner der Master durch die Zuweisungen einer niedrigen statischen Priorität verhungert. Die stetige Verdrängung durch einen Master mit höherer Priorität begrenzt sich idealerweise auf einige wenige Slots. Darüber hinaus ergibt sich eine deutlich höhere Auslastung, da ungenutzte Slots durch wartende Transaktionen anderer Master aufgefüllt werden können. Nichtsdestotrotz muss wie bei reinem TDMA die maximale Ausführungszeit von einzelnen Instruktionen bekannt sein, um die Ausführung innerhalb eines Slots garantieren zu können. Das folgende Beispiel (siehe Beispiel 2.1.8) zeigt eine Anwendung von PD und einen Vergleich der Auslastung einer gemeinsamen Ressource bei der Verwendung von PD und TDMA.

2.1.8 Beispiel. Die vier Cores führen die Instruktionen I1 bis I11 aus. Der Einfachheit halber sind den vier Mastern in jedem Frame, der ihnen nicht gehört, identische Prioritäten (C1: 1, C2: 2, C3: 3 und C4: 4) zugeordnet (siehe Abbildung 2.12). In ihrem eigenen Slot besitzt jeder Master die Priorität 5.
Bei der Anwendung von Priority Division (1) zeigt sich eine deutlich höhere Auslastung der gemeinsamen Ressource. So werden der Slot S3 im ersten Frame sowie die Slots S0 und S1 im zweiten Frame mit Instruktionen aufgefüllt und somit die Zeit, in der die gemeinsame Ressource ungenutzt gewesen wäre, minimiert. Hingegen kann bei der Verwendung von TMDA (2) der Core C2 gerade einmal drei seiner sieben Anfragen ausführen.

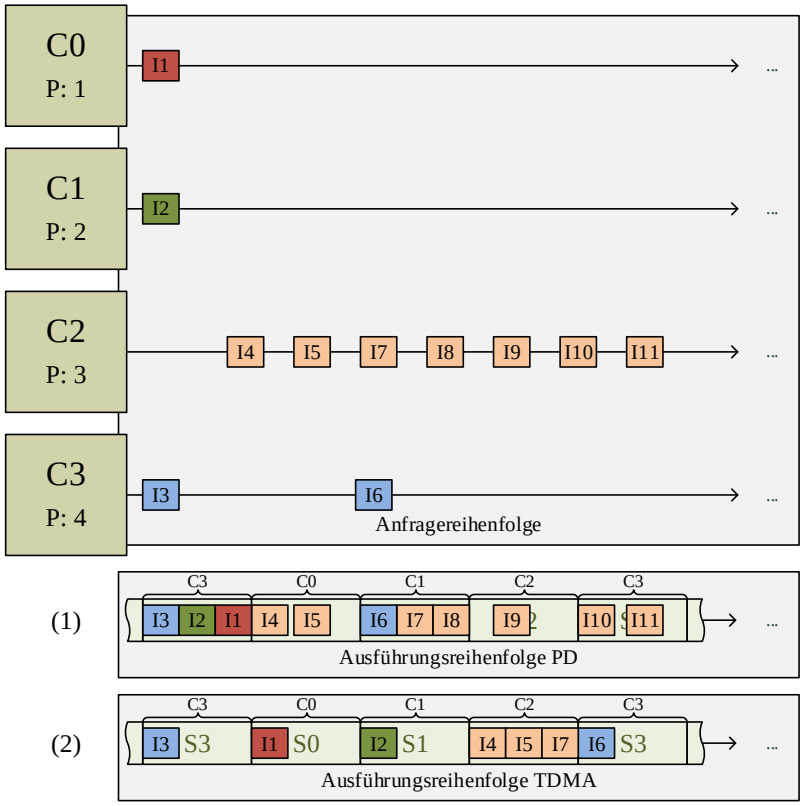


Abbildung 2.12: Anwendung von Priority Division und Vergleich der Ausführungsreihenfolge mit reinem TDMA.

Für Kommunikationsstrukturen ergeben sich allerdings auch wieder einige Einschränkungen. Problematisch ist erneut, dass es sich bei den Instruktionen wieder um atomare und somit nicht unterbrechbare Operationen handelt. Dies kann eine Verdrängung von Instruktionen in einen späteren Slot oder sogar in den nächsten Frame zur Folge haben, da die prioritätsbasierte Arbitrierung, wie im Abschnitt 2.1.3 dargestellt, unter der Annahme von nicht unterbrechbaren Operationen keinen sofortigen Zugriff durch den höchstprioren Core ermöglicht. Dieses Verhalten wird in Beispiel 2.1.9 näher beschrieben.

2.1.9 Beispiel. Die vier Cores führen erneut die Instruktionen I1 bis I11 aus (siehe Abbildung 2.13). Durch die Verwendung von Priority Division (1) wird die Instruktion I3 in einem „fremden“ Slot ausgeführt. Die Instruktion I5 hätte bei der Verwendung von TDMA (2) eigentlich noch zur Ausführung kommen können. Um das Hineinragen in den nächsten Slot zu verhindern, muss I5 um beinahe einen ganzen Frame verzögert werden. Hier zeigt sich, dass das Zeitverhalten gegenüber TDMA schlechter vorhersagbar ist.

PD ist ein Verfahren, was aufgrund der in feste Zeitschlitze partitionierten Zugriffszeit immer noch eine gute Analysierbarkeit besitzt und zudem die Auslastung einer Ressource extrem erhöhen kann. Allerdings lassen sich aufgrund von verdrängten Instruktionen keine vergleichbar engen Zeitschranken garantieren wie bei der Verwendung von TDMA.

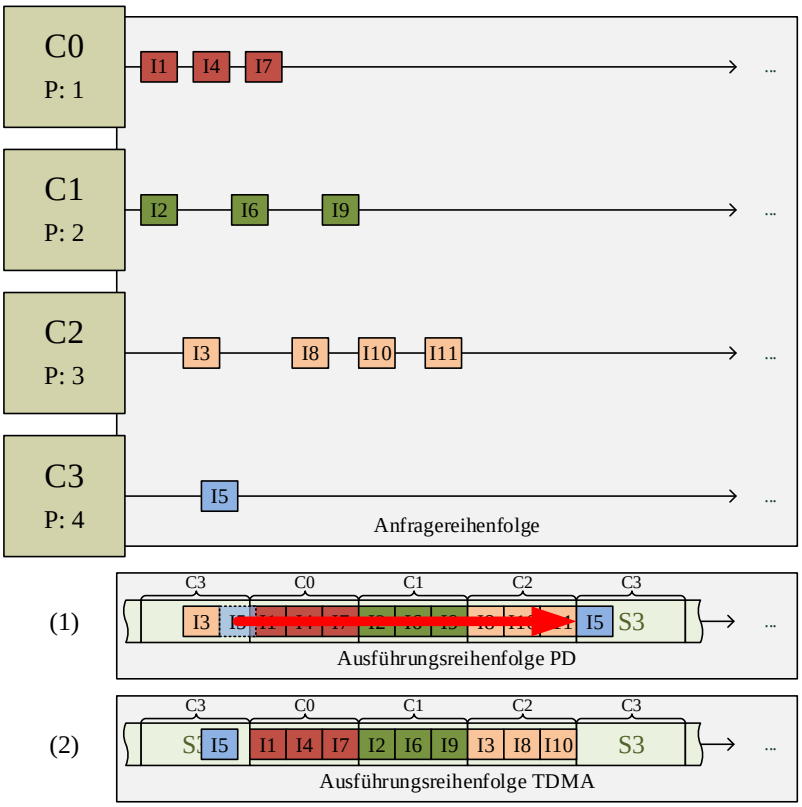


Abbildung 2.13: Verdrängung von Instruktionen bei der Verwendung von PD.

2.2 Performance-Metriken

Es existieren unterschiedliche Performance-Metriken (siehe Abbildung 2.14), um die Ausführungszeit von Programmen oder Anwendungen zu beschreiben.

Im alltäglichen Umfeld (wie z. B. bei Office- oder anderen nicht zeitkritischen Anwendungen) steht die durchschnittliche Performance im Vordergrund. Die *Average Case Execution Time* (kurz: ACET) ist eine Kennziffer, die die Ausführungszeit im durchschnittlichen Fall angibt. Sie berechnet sich als einfacher Mittelwert einer Menge von Messwerten von Ausführungszeiten und gibt somit Auskunft über die durchschnittliche Performance einer Anwendung.

Bei Anwendungen mit Realzeitanforderungen ist es notwendig, eine obere Schranke für die Ausführungszeit zu kennen und die Einhaltung dieser Schranke unter allen Umständen garantieren zu können. Diese obere Schranke für die Ausführungszeit ist die *Worst Case Execution Time* (kurz: WCET) [26] - also die größtmögliche Ausführungszeit für alle möglichen Eingaben und Systemzustände. Häufig ist die exakte Bestimmung der tatsächlichen WCET ($WCET_{REAL}$) einer Anwendung aufgrund von komplexen Programmstrukturen und daraus resultierender schwieriger Analysierbarkeit nicht möglich. In solchen Fällen

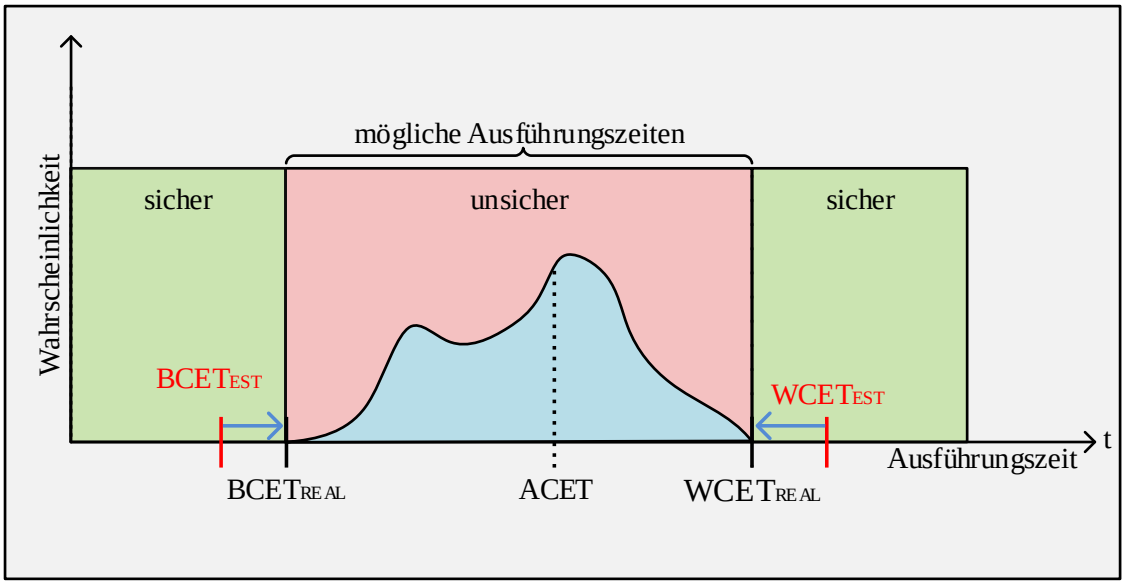


Abbildung 2.14: Unterschiede von BCET, ACET und WCET.

versucht man, eine möglichst exakte obere Schranke für die tatsächliche WCET zu bestimmen.

Die vorhergesagte WCET ($WCET_{EST}$) ist *sicher*, wenn $WCET_{REAL} \leq WCET_{EST}$ gilt. $WCET_{EST}$ ist im Fall $WCET_{EST} = WCET_{REAL}$ eine *exakte* Schranke, während es sich im Fall $WCET_{EST} > WCET_{REAL}$ um eine Überabschätzung der WCET handelt. Ziel ist eine möglichst exakte Schätzung der WCET und die daraus resultierende Minimierung von $WCET_{REAL} - WCET_{EST}$.

Unterabschätzungen ($WCET_{EST} < WCET_{REAL}$) sind unsichere Schranken und müssen bei Systemen mit harten Echtzeitanforderungen unter allen Umständen vermieden werden, da diese potenziell zur Nichteinhaltung von Deadlines führen können.

Die zur WCET konträre Kennziffer ist die *Best Case Execution Time* (kurz: BCET). Hierbei handelt es sich um die minimale Ausführungszeit einer Anwendung, also um die Ausführungszeit im besten Fall. Analog zur WCET kann auch hier die vorhergesagte BCET ($BCET_{EST}$) bestimmt werden, die sich exakt umgekehrt zur $BCET_{REAL}$ wie die $WCET_{EST}$ zur $WCET_{REAL}$ in Bezug auf Sicherheit und Exaktheit verhält.

Es existieren unterschiedliche Ansätze zur Bestimmung des Zeitverhaltens von Applikationen; diese lassen sich in zwei grundlegende Kategorien einteilen: statische und dynamische Analyseverfahren.

Statische Verfahren basieren auf mathematischen Berechnungen, die auf dem Kontrollflussgraphen der Anwendung basieren. Diese Verfahren können durch Wert-, Schleifen-, Speicher-, Cache- und Pipelineanalysen die WCET einer Anwendung mehr oder weniger exakt bestimmen, setzen aber ein detailliertes mathematisches Modell der untersuchten Plattform voraus.

Dynamische Verfahren verfolgen einen anderen Ansatz: Die Ausführungszeit wird mit unterschiedlichen Eingaben gemessen und durch die Verwendung möglichst ungünstiger Eingaben wird versucht, eine möglichst exakte Bestimmung der WCET zu erreichen. Bedauerlicherweise ist aufgrund der hohen Komplexität der Anwendungen die Generierung von möglichst ungünstigen Eingaben im Allgemeinen nicht effizient möglich, was dieses Verfahren für eingebettete Systeme mit harten Echtzeitanforderungen ungeeignet macht. Eine vollständige Enumeration des Eingaberaums ist wegen der mit der Eingabelänge exponentiell steigenden Anzahl an Möglichkeiten im Allgemeinen nicht durchführbar.
Insgesamt ist die Bestimmung von sicheren und möglichst exakten BCET- und WCET-Schranken ein hoch komplexes Thema, das im weiteren Teil der Arbeit nicht weiter betrachtet werden soll. Die genaue Untersuchung der beiden Arbitrierungsverfahren TDMA und PD im Hinblick auf Auslastung und ACET ist aber insbesondere wegen ihrer exzellenten Eigenschaften bei der Bestimmung der WCET bzw. BCET interessant.

Kapitel 3

Hardwareplattform

Ziel der Arbeit ist es, die Auswirkungen von Arbitrierungsverfahren auf die ACET und WCET verschiedener Anwendungen zu untersuchen und zu analysieren. Um die notwendigen Daten komfortabel erheben zu können und bei der notwendigen Parametrisierung der Komponenten möglichst flexibel zu sein, fiel die Entscheidung auf eine virtuelle Ausführungsplattform. Zu diesem Zweck wurden eine Referenzplattform mit einem einzelnen ARM-Core sowie je eine X2-, X4- und X8-Plattform in VaST CoMET [24] entwickelt. Hierbei handelt es sich um ein Werkzeug zur Erstellung von sog. virtuellen System-Prototypen (engl: Virtual System Prototype, kurz: VSP) für eingebettete Systeme oder Ein-Chip-Systeme (engl: System-on-a-Chip, kurz: SoC).
Im folgenden Kapitel sollen die entwickelten und für die Benchmarks verwendeten virtuellen Ausführungsplattformen vorgestellt und einige Designentscheidungen näher erläutert werden. Dabei wird in Unterkapitel 3.1 zunächst auf die grundlegende Topologie eingegangen und im folgenden Unterkapitel 3.2 erfolgt die Vorstellung der zentralen Komponente der Plattform: die Arbitration Bridge.

3.1 Topologie

Abbildung 3.1 zeigt die grundlegende Topologie der Plattform. Die einzelnen Core-Systeme (grün) können über einen gemeinsamen Bus, den MemoryBus, auf den gemeinsamen Speicher zugreifen. Die zentrale Komponente - die Arbitration Bridge (hellblau) - regelt den Zugriff auf den Speicher und sequenzialisiert konkurrierende Zugriffe auf den gemeinsamen Bus. Die einzelnen Core-Systeme verfügen dabei jeweils über:

- einen ARM-Core (ARM7TDMI)
- einen privaten BootROM [8MB]
- ein privates Scratchpad für Daten (D-SPM) [512KB]

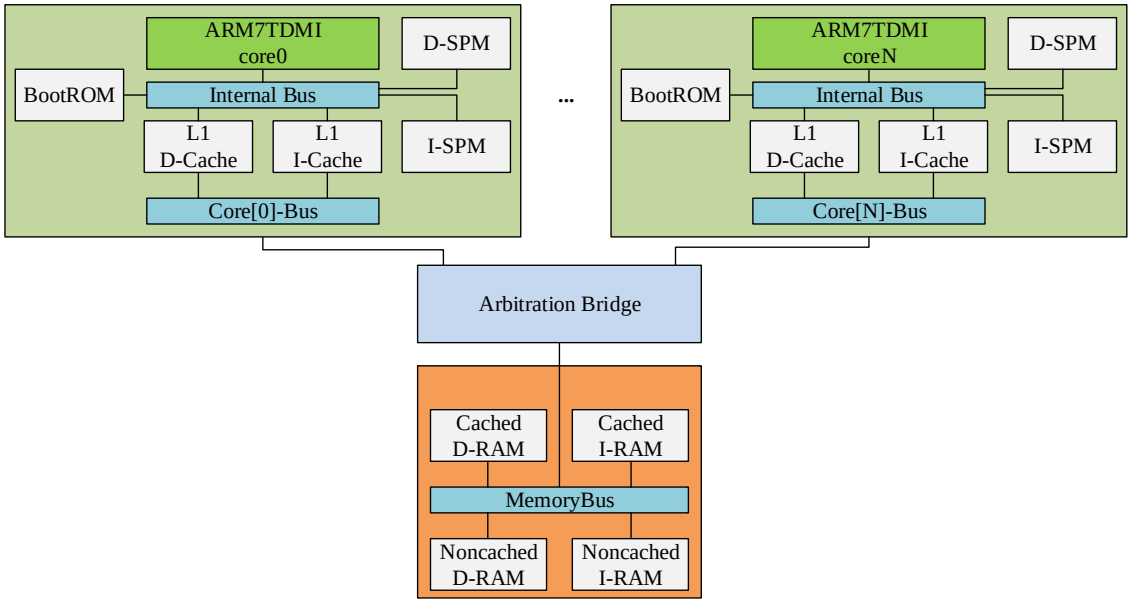


Abbildung 3.1: Schematische Darstellung der Architektur der Hardwareplattform.

- ein privates Scratchpad für Instruktionen (I-SPM) [512KB]
- einen privaten L1-Cache für Daten (L1 D-Cache) [konfigurierbare Größe]
- einen privaten L1-Cache für Instruktionen (L1 I-Cache) [konfigurierbare Größe]

Bei der Wahl des Prozessors fiel die Wahl auf einen ARM7TDMI. Hierbei handelt es sich um einen 32-Bit Universal-Mikroprozessor, der eine hohe Performance bei einem gleichzeitig sehr niedrigem Energieverbrauch bietet [2]. Dieser Prozessor eignet sich aufgrund seines gut vorhersagbaren Zeitverhaltens und seiner hohen Energieeffizienz besonders für den Einsatz in eingebetteten Systemen. Das Zeitverhalten ergibt sich durch die Verwendung eines RISC-Befehlssatzes (Reduced Instruction Set Computer, kurz: RISC) und einer nur dreistufigen Pipeline (Fetch → Decode → Execute), die das Verhalten insgesamt gut vorhersagbar machen.
An dieser Stelle sei erwähnt, dass die Wahl explizit nicht auf den ARM Cortex-M0 - den Nachfolger des ARM7TDMI - gefallen ist. Dies liegt darin begründet, dass eine sehr gute Unterstützung für den ARM7TDMI innerhalb des gewählten Compilerframeworks vorhanden ist und die zur Verfügung stehenden Benchmarks alle auf anderen Plattformen mit einem ARM7TDMI getestet wurden. Die Ergebnisse lassen sich aber wegen der Ähnlichkeit der Plattformen direkt auf den Cortex-M0 übertragen.
Der gemeinsame Speicher verfügt über unterschiedliche Speicherbereiche. Die mit '*' gekennzeichneten Bereiche sind für die einzelnen Cores weiter partitioniert, so dass jeder Core innerhalb einer privaten Partition im jeweiligen Speicherbereich arbeiten kann. Eventuell notwendige Inter-Core-Kommunikation wird dabei über einen GlobalData-Bereich realisiert. Nachfolgend findet sich eine Übersicht über die vorhandenen Speicherbereiche:

- globaler BootROM [128MB]
- globaler BootRAM [16MB]
- Daten-RAM (cached*/uncached*) [512KB pro Core]
- Instruktions-RAM (cached*/uncached*) [512KB pro Core]
- GlobalData-Bereich [256 MB]
- Flash-Speicher [32MB]

Die unterschiedlichen Speichersegmente der Cores befinden sich sowohl im gemeinsamen Speicher als auch in den SPMs der einzelnen Core-Systeme. Das .data-Segment (globale und statische vorinitialisierte Variablen) und das .bss-Segment (globale und statische nicht initialisierte Variablen) residieren innerhalb des uncached D-RAM des gemeinsamen Speichers. Die .text-Sektion, die den eigentlichen Programmcode enthält, befindet sich innerhalb des privaten I-SPM, wohingegen das .stack-Segment sich innerhalb des D-SPM des zugehörigen Cores befindet. Zugriffe auf die einzelnen Datenbereiche des gemeinsamen Speichers werden also über die Bridge abgewickelt, wohingegen Fetches von Instruktionen innerhalb der einzelnen Core-Systeme ablaufen. Diese Transaktionen gehen also nicht über den gemeinsamen Bus und können somit nicht die Zugriffe auf die jeweiligen Datenbereiche des gemeinsamen Speichers beeinträchtigen. Außerden wird hierdurch eine unrealistisch hohe Auslastung des gemeinsamen Busses verhindert.
Die einzelnen SPMs innerhalb der Core-Systeme sowie der BootROM bzw. BootRAM des gemeinsamen Speichers verursachen beim Zugriff eine Verzögerung von nur einem Takt, wohingegen die einzelnen Daten- und Instruktions-RAMs sowie der GlobalData-Bereich eine Reaktionszeit von drei Zyklen bei einem Zugriff aufweisen. Der Flash-Speicher verfügt über eine Verzögerung von 6 Zyklen.

Speicherzugriff-Timings (Core-System)		
Speicherbereich	**Read-Op** (Zyklen)	**Write-Op** (Zyklen)
D-SPM	1	1
I-SPM	1	1

Speicherzugriff-Timings (Gemeinsamer Speicher)		
Speicherbereich	**Read-Op** (Zyklen)	**Write-Op** (Zyklen)
BootROM	1	1
BootRAM	1	1
D-RAM	3	3
I-RAM	3	3
GlobalData	3	3
Flash-Speicher	6	6

Mit Hilfe der vorgestellten Plattformen kann nach der Integration in das Compilerframework der Einfluss von Arbitrierungsverfahren auf die Ausführungszeiten der auf den einzelnen Cores laufenden Anwendungen unter Verwendung eines gemeinsamen Speichers näher untersucht werden. Von zentraler Bedeutung ist hierbei die Arbitration Bridge, die im folgenden Abschnitt 3.2 näher vorgestellt wird.

3.2 Arbitration Bridge

Die Arbitration Bridge ist die zentrale Komponente zur Arbitrierung und Datenerhebung. Zunächst soll an dieser Stelle eine kurze Übersicht über die implementierten Features gegeben sowie auf die generelle Funktionsweise der Bridge und einige Implementierungsdetails eingegangen werden; anschließend wird das Modul zur statistischen Datenerhebung vorgestellt.

3.2.1 Features

Nachfolgend werden die implementierten Features, von denen einige anschließend kurz erläutert werden, aufgeführt:

- Unterstützung für maximal 64 Cores: Eine Festlegung der Maximalanzahl an unterstützten Cores war notwendig, da für einzelne Komponenten innerhalb der CoMET-Simulationsumgebung eine feste Anzahl von Schnittstellen definiert werden muss.
- Unterstützung für maximal 64 Slots: Hier gelten die gleichen Einschränkungen wie bei der maximalen Anzahl an unterstützen Cores.
- Unterstütze Arbitrierungsverfahren:
  - **Faire Arbitrierung**: Die Arbitrierung erfolgt in diesem Fall nach dem einfachen Round-Robin Verfahren (siehe Abschnitt 2.1.1).
  - **TDMA**: Hierbei handelt es sich um ein im Vergleich zur Vorstellung in Abschnitt 2.1.2 leicht modifiziertes Verfahren. Das hier implementierte Verfahren erlaubt Zeitschlitze unterschiedlicher Länge, um ggf. unterschiedliche Anzahlen an Speicherzugriffen zwischen den einzelnen Cores besser handhaben zu können.
  - **Prioritätsbasierte Arbitrierung**: Die Arbitrierung der Ressource erfolgt in diesem Fall nach globalen Prioritäten (siehe Abschnitt 2.1.3).
  - **Priority Division**: Bei der hier implementierten Form von PD handelt es sich um eine leicht modifizierte Version der in Abschnitt 2.1.4 vorgestellten Variante. Die Prioritäten können dabei lediglich global pro Core und nicht pro Core je Slot definiert werden, um den Konfigurationsaufwand besser handhaben zu können.

Des Weiteren wurden sog. Slot-Modes eingeführt: hierbei kann für jeden PD-Slot ein separates Arbitrierungsverfahren (reines PD, reines TDMA oder reine PA) angegeben werden, nach dem die gemeinsame Ressource innerhalb dieses Zeitschlitzes arbitriert wird. Diese Slot-Modes ermöglichen eine feingranulare Konfiguration der Bridge bei der Verwendung von PD.

- Memory-Mapped-Register:
  - **Anpassung von Konfigurationsparametern**: Die Verwendung von Memory-Mapped Registern ermöglicht eine Anpassung aller Konfigurationsparameter zur Laufzeit.
  - **Warteregister**: Hierbei handelt es sich um ein spezielles Register zur Vereinfachung der WCET-Analyse.
  - **Statistik-Register**: Über diese Register kann das genaue Zeitfenster definiert werden, in dem die statistischen Daten gesammelt werden.
- Statistische Datenerhebung:
  - **Ausführungszeiten**: Ausführungszeit der einzelnen Core-Anwendungen sowie der gesamten Benchmarks.
  - **Speicherzugriffszeiten**: Analyse von Timing-Werten von unterschiedlichen Speicherzugriffen.
  - **Busauslastung**: Auswertung der Busauslastung insgesamt (sowohl pro einzelnem Core als auch die kumulierten Werte aller Cores) und pro Slot (bei Verwendung von TDMA und PD).

3.2.2 Funktionsweise

Dieser Abschnitt gibt Aufschluss über die grundlegende Funktionsweise der Arbitration Bridge. Abbildung 3.2 illustriert dies beispielhaft anhand der Ausführung einer Transaktion.

Neue Transaktionen der einzelnen Cores (1) werden von der Bridge entgegengenommen und in einer Queue für neue Transaktionen abgelegt (2). Von dort werden sie auf die einzelnen Core-Queues verteilt (3), die ihrerseits wieder Teil einer globalen Prioritätswarteschlange (4) sind. Wenn der gemeinsame Bus frei ist, entnimmt die Arbitrierungslogik (5) - je nach Arbitrierungsverfahren - die nächste Transaktion aus der korrespondierenden Queue und leitet diese an den gemeinsamen Speicher weiter (6). Abgeschlossene, zurückkehrende Transaktionen (7) werden in einer Queue für abgeschlossene Transaktionen zwischengespeichert (8), bevor sie an den Core zurückgeliefert werden (9). Das Modul zur statistischen Datenerhebung (10) protokolliert dabei Zugriff- und Wartezeiten sowie weitere Zugriffsstatistiken.

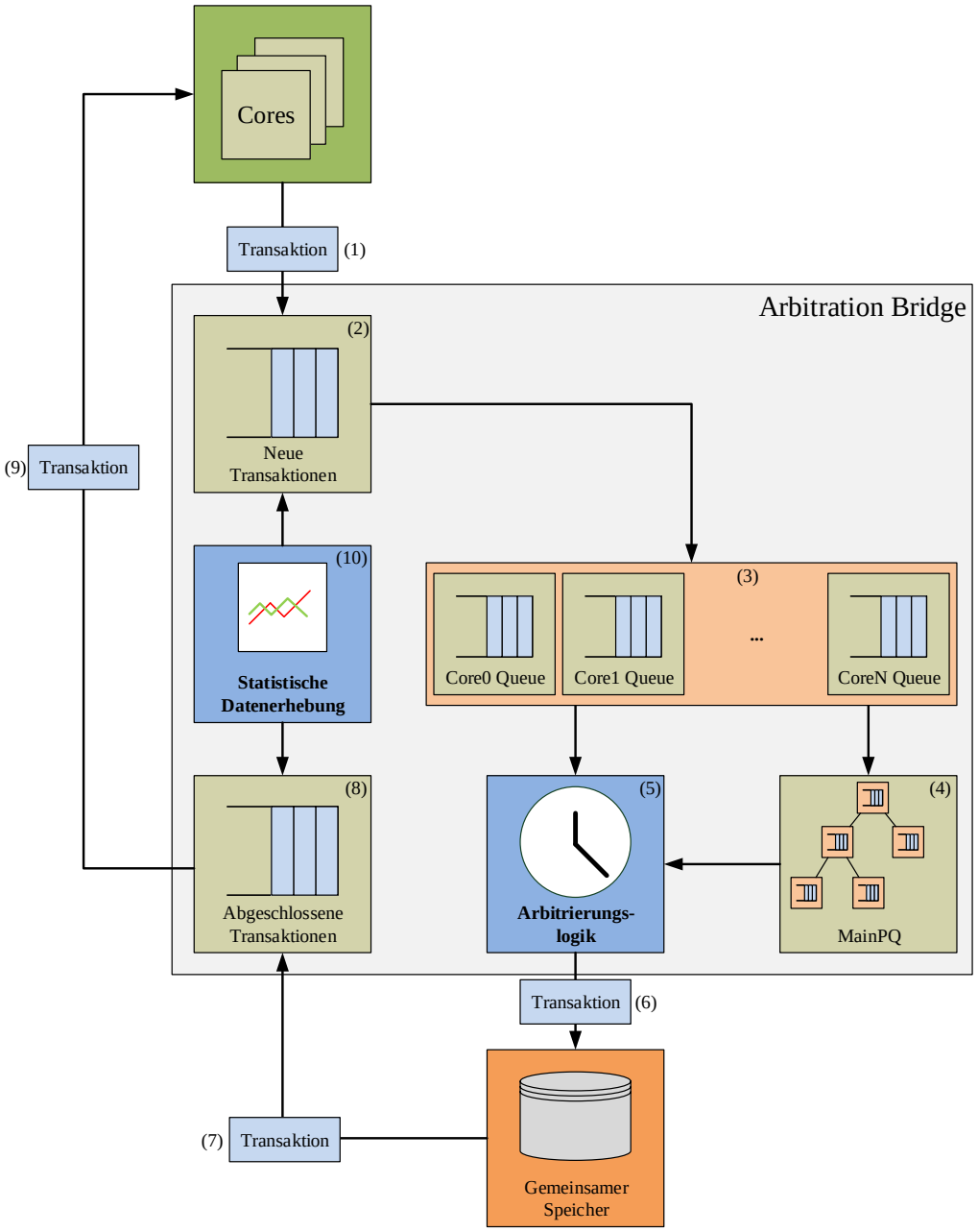


Abbildung 3.2: Funktionsweise der Arbitration Bridge.

3.2.3 Statistische Datenerhebung

Neben der Konfiguration der einzelnen Parameter zur Laufzeit bietet die Bridge die Möglichkeit, statistische Daten zu den Ausführungszeiten der einzelnen Anwendungen, den Speicherzugriffszeiten und der Auslastung des Busses während der Simulation zu sammeln. Nachfolgend sollen die unterschiedlichen Kenngrößen ebenfalls kurz vorgestellt und erläutert werden.

- **Ausführungszeit** [pro Core/gesamt]: Die Gesamtlaufzeit aller Anwendungen sowie die Laufzeiten der einzelnen Core-Anwendungen (siehe Abbildung 3.3).

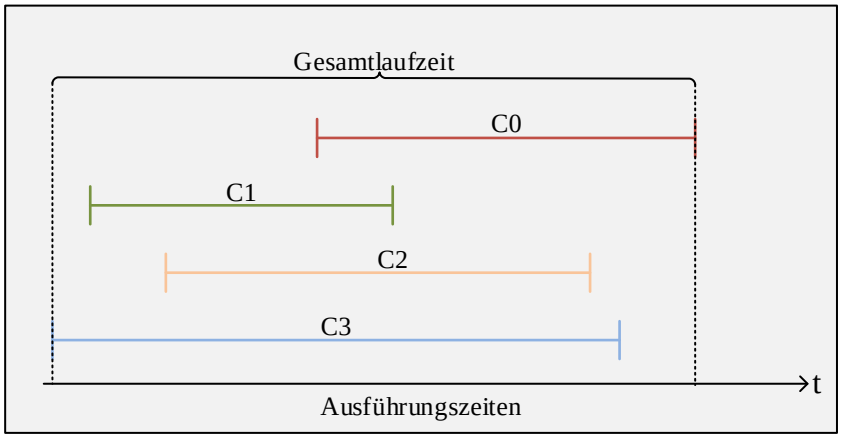


Abbildung 3.3: Ausführungszeiten und Gesamtlaufzeit.

- **Speicherzugriffs-Timings**:
  Die unterschiedlichen Speicherzugriff-Timings werden anhand von Abbildung 3.4 verdeutlicht.

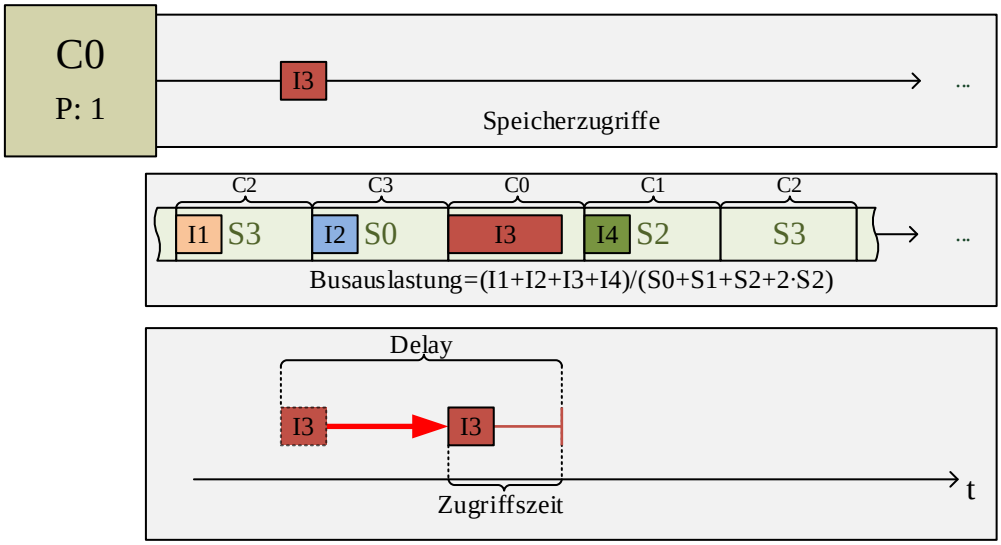


Abbildung 3.4: Veranschaulichung der Speicherzugriffs-Timings.

  - **Speicherzugriffszeit** [pro Core und gesamt je Speicherbereich]: Die tatsächliche Ausführungszeit der jeweiligen Transaktionen beim Zugriff auf den gemeinsamen Speicher.
  - **Speicherzugriffs-Delay** [pro Core und gesamt je Speicherbereich]: Die effektive Ausführungszeit der jeweiligen Transaktionen (Wartezeit + Ausführungszeit).

  - **Extremwerte** [pro Core und gesamt je Speicherbereich]: Minimum und Maximum der jeweiligen Timings.

- **Bus-Auslastung** [pro Slot/gesamt]: Auslastung pro Slot (bei TDMA/PD) und Gesamtauslastung des gemeinsamen Busses hinter der Bridge.

- **Anzahl der Speicherzugriffe** (Read/Write/Fetch [pro Core/gesamt] je Speicherbereich): Anzahlen und Verteilung der einzelnen Speicherzugriffstypen.

3.2.4 Parametrisierung

Die Grundkonfiguration der Arbitration Bridge und die Definition von Standard-Konfigurationsparametern erfolgt in CoMET selbst bzw. im zugehörigen Eclipse-Plugin. Da diese Konfigurationsmöglichkeit allerdings bei mehreren unabhängigen Simulationsläufen wenig komfortabel ist, wurden zusätzliche Ebenen zur Konfiguration der Arbitration Bridge geschaffen.
Bei der zweiten Stufe der Konfiguration handelt es sich um die sog. Memory-Mapped-Register, die außerdem eine Rekonfiguration der Parameter zur Laufzeit ermöglichen. Über die Register lassen sich sämtliche Konfigurationsparameter verändern - aus diesem Grund werden die einzelnen Parameter am Beispiel des Registerlayouts vorgestellt. Nachfolgend finden sich alle Informationen über das Layout der Memory-Mapped-Register der Arbitration Bridge und die Funktion der jeweiligen Bereiche.

Allgemeine Parameter (0x00000000 - 0x000000FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x0	CurrentOperationalMode	RO	Aktuelles Arbitrierungsverfahren
0x4	DesiredOperationalMode	RW	Gewünschtes Arbitrierungsverfahren
0x8	NumSlots	RW	Anzahl der Slots
0xC	DefaultSlotLength	RW	Standard-Slotlänge (Default: 20)
0x10	MaximumAccessTime	RW	Maximale Dauer eines Speicherzugriffs
0x14	StartInBootMode	RO	Flag für Boot-Mode
0x18	DelayRegister	WO	Warteregister

Die allgemeinen Parameter dienen zur grundlegenden Konfiguration der Bridge. Über das Register mit Offset 0x0 lässt sich das aktuell verwendete Arbitrierungsverfahren abfragen. Eine Änderung des Verfahrens lässt sich mit einem schreibenden Zugriff auf das Register 0x4 initiieren, das erfolgreiche Umschalten kann dann durch Abfrage von Register 0x0 verifiziert werden.
Das Setzen einer neuen DefaultSlotLength über das Register 0xC weist allen Slots eine neue Standard-Slotlänge zu und überschreibt damit alle abweichend definierten Slotlängen. Hierbei wird standardmäßig eine Slotlänge von 20 Zyklen vergeben. Die MaximumAccessTime

definiert die maximale Ausführungszeit für eine Speichertransaktion. Dieser Parameter ist notwendig, um bei TDMA bzw. PD das Hineinragen von Transaktionen in nachfolgende Slots zu verhindern (siehe Abschnitt 2.1.2).
Das Register mit Offset 0x14 ermöglicht die Verwendung des sog. Boot-Modes. Bei PA kann es passieren, dass einzelne Cores während des Bootloader-Vorgangs verhungern. Um dieses Problem zu umgehen, werden beim Einschalten des Boot-Modes alle Speicherzugriffe während des gesamten Bootloader-Prozesses mit FA abgehandelt. Nachdem der Bootloader-Prozess abgeschlossen ist, erfolgt automatisch das Umschalten auf das gewünschte Arbitrierungsverfahren.
Das Warteregister an Offset 0x18 wurde eingeführt, um die WCET-Analyse (bei TDMA bzw. PD) zu vereinfachen. Ein Zugriff auf das jeweilige Register wird bis zum ersten Slot des jeweiligen Cores innerhalb eines Frames verzögert (siehe Abbildung 3.5). Der erste Fall (1) zeigt die Anwendung für den Core C0 und nur einen zugeordneten Slot S0, (2) zeigt die Anwendung bei der Zuordnung mehrerer Slots (S0 und S2) zu einem Core C0.
Diese Funktion kann von WCET-bewussten Compilern genutzt werden, um die Ausführung von Instruktionen bewusst an eine bestimmte Stelle im TDMA-Zyklus zu binden.

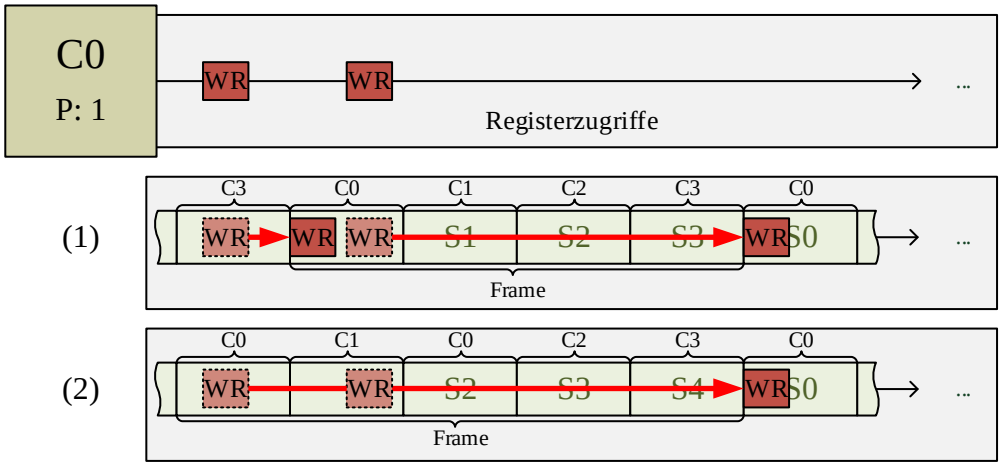


Abbildung 3.5: Funktion des Warteregisters.

Core-Prioritäten (0x00000100 - 0x000001FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x100	PriorityCore0	RW	Priorität von Core0
0x104	PriorityCore1	RW	Priorität von Core1
...	...	...	...
0x1FC	PriorityCore63	RW	Priorität von Core63

Durch die Register im Bereich 0x100-0x1FF können die Prioritäten der einzelnen Cores verändert und abgefragt werden. Gültige Prioritätswerte sind für prioritätsbasierte Arbitrierung > 0, für Priority Division ≥ 0 (Cores mit Priorität 0 haben in allen Slots, denen sie

nicht direkt als Besitzer zugeordnet sind, keine Berechtigung zur Ausführung von Transaktionen). Ein Core mit höherer Priorität erhält im Falle eines konkurrierenden Zugriffs den Zuschlag. Die zugeordneten Prioritäten der unterschiedlichen Cores müssen dabei immer paarweise verschieden sein.

Slot-Besitzer (0x00000200 - 0x000002FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x200	OwnerSlot0	RW	Besitzer von Slot0
0x204	OwnerSlot1	RW	Besitzer von Slot1
...	...	...	...
0x2FC	OwnerSlot63	RW	Besitzer von Slot63

Über Registerzugriffe im Bereich 0x200-0x2FF können die Besitzer der jeweiligen Slots gesetzt oder abgefragt werden. Die Core-IDs der jeweiligen Cores dienen dabei als Identifikation für die jeweilige Zuordnung, gültige Werte liegen dabei also im Bereich von 0 bis 63.

Slotlängen (0x00000300 - 0x000003FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x300	LengthSlot0	RW	Länge von Slot0
0x304	LengthSlot1	RW	Länge von Slot1
...	...	...	...
0x3FC	LengthSlot63	RW	Länge von Slot63

Über den Bereich 0x300-0x3FF können spezifische Slotlängen, die sich von der konfigurierten DefaultSlotLength unterscheiden, konfiguriert werden.

Statistik-Register (0x00000400 - 0x000004FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x400	StatisticsRegisterCore0	RW	Statistik-Register für Core0
0x404	StatisticsRegisterCore1	RW	Statistik-Register für Core1
...	...	...	...
0x4FC	StatisticsRegisterCore63	RW	Statistik-Register für Core63

Die Statistik-Register dienen zur Definition des Zeitraums, in dem die statistischen Daten für den jeweiligen Core gesammelt werden sollen. Der erste schreibende Zugriff auf das jeweilige Register aktiviert dabei das Statistikmodul, ein folgender zweiter schreibender Zugriff deaktiviert es wieder und sorgt für eine Ausgabe der statistischen Daten für den jeweiligen Core. Die globalen Statistiken werden ausgegeben, sobald der letzte Core die Ausführung des Benchmarks abgeschlossen hat.
Dieser Mechanismus ist notwendig, da der Bootloader-Prozess und die innerhalb dieses Prozesses ablaufenden Speicherzugriffe nicht die spätere Auswertung beeinflussen sollen (siehe Abbildung 3.6).

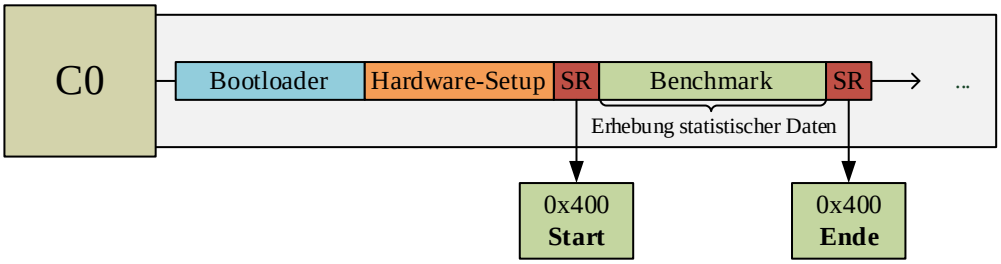


Abbildung 3.6: Funktion des Statistik-Registers am Beispiel von Core C0.

Slot-Modes (0x00000500 - 0x000005FF)			
Offset	**Name**	**Zugriff**	**Bemerkungen**
0x500	ModeSlot0	RW	Slot-Mode für Slot0
0x504	ModeSlot01	RW	Slot-Mode für Slot1
...	...	...	...
0x5FC	ModeSlot063	RW	Slot-Mode für Slot63

Der letzte Registersatz im Bereich von 0x500-0x5FF dient zur Konfiguration der einzelnen Slot-Modes bei der Verwendung von Priority Division. Dabei kann jedem Slot ein eigener Arbitrierungsmodus (reines PD (PURE_PD), reines TDMA (PURE_TDMA) und reines PA (PURE_PA) zugewiesen werden.
Bei der Verwendung von PURE_PD erhält der Slot-Besitzer die höchste Priorität, durch ihn nicht genutzte Zugriffszeit kann hierbei von Cores mit niedrigerer Priorität verwendet werden. Der Slot-Modus PURE_TDMA ermöglicht die Einführung von TDMA-Slots, die explizit nur von dem jeweiligen Slot-Besitzer in Anspruch genommen werden können; Zugriffe in PURE_PA-Slots werden durch prioritätsbasierte Arbitrierung behandelt.
Durch die gleichzeitige Verwendung von PURE_TDMA- und PURE_PD-Slots lassen sich zeitlich exzellent vorhersagbare Systeme mit gleichzeitig relativ hoher Auslastung entwickeln. Während Anwendungen mit hohen Realzeitanforderungen dedizierte TDMA-Slots zugeordnet sind und somit über garantiere Zugriffszeiten und Bandbreiten innerhalb eines PD-Frames verfügen, teilen sich weniger zeitkritischen Anwendungen die PURE_PD-Slots und können somit die Auslastung des Gesamtsystems deutlich erhöhen. Transaktionen von Anwendungen mit Realzeitanforderungen laufen dabei keine Gefahr in den nächsten Frame verdrängt zu werden (siehe Abschnitt 2.1.4), was einen der größten Nachteile von PD deutlich entschärfen kann.

Kapitel 4

WCC-Integration

Der WCET-aware C Compiler (kurz: WCC) ist ein ANSI-C Compiler, der speziell für die Minimierung der WCET entwickelt wurde [6]. Dazu verwendet er unter anderem WCET-Analysetools wie beispielsweise aiT [1] zur statischen WCET-Analyse und komplexere Analysetechniken wie beispielsweise Werte-, Pipeline- und Cacheanalysen zur weiterführenden Optimierung [12][7][5].
Um die Nutzung der Plattformen und Konfiguration der Bridge dabei möglichst komfortabel zu gestalten, sind unterschiedliche Schnittstellen zur bestehenden Infrastruktur des WCC geschaffen worden. Dieses Kapitel beschreibt die Integration der entwickelten virtuellen Ausführungsplattformen in den WCC. Dabei wird in Unterkapitel 4.1 zunächst auf die Task-Beschreibungen der einzelnen Benchmarks eingegangen, während im folgenden Unterkapitel 4.2 die Konfigurationsmöglichkeiten für die Arbitration Bridge vorgestellt werden.
Abbildung 4.1 veranschaulicht dabei die grundlegende Integration in den WCC. Grundlage für einen Simulationslauf bilden die Taskbeschreibung (1), die Bridge-Konfiguration (2) und das zugehörige Memory Layout (3). Für jeden Core - wie beispielsweise Core0 (4) - wird der jeweilige Task in einem regulären Compilerlauf kompiliert und zu einem lauffähigen Binary gelinkt (5). In einem weiteren Schritt (6) werden alle so entstandenen Core-Binaries und der Bootloader durch den BootROM-Builder zu einem kombinierten SREC (7) [16] zusammengefasst. In der entsprechenden Zielplattform lädt der Bootloader die einzelnen Core-Binaries in die Speicherbereiche des jeweiligen Cores (8). Die Simulation wird durchgeführt (9) und die statistischen Daten gesammelt (10). Diese Simulationsergebnisse können ggf. für weitere High- (11) und Low-Level-Optimierungen (12) verwendet werden.

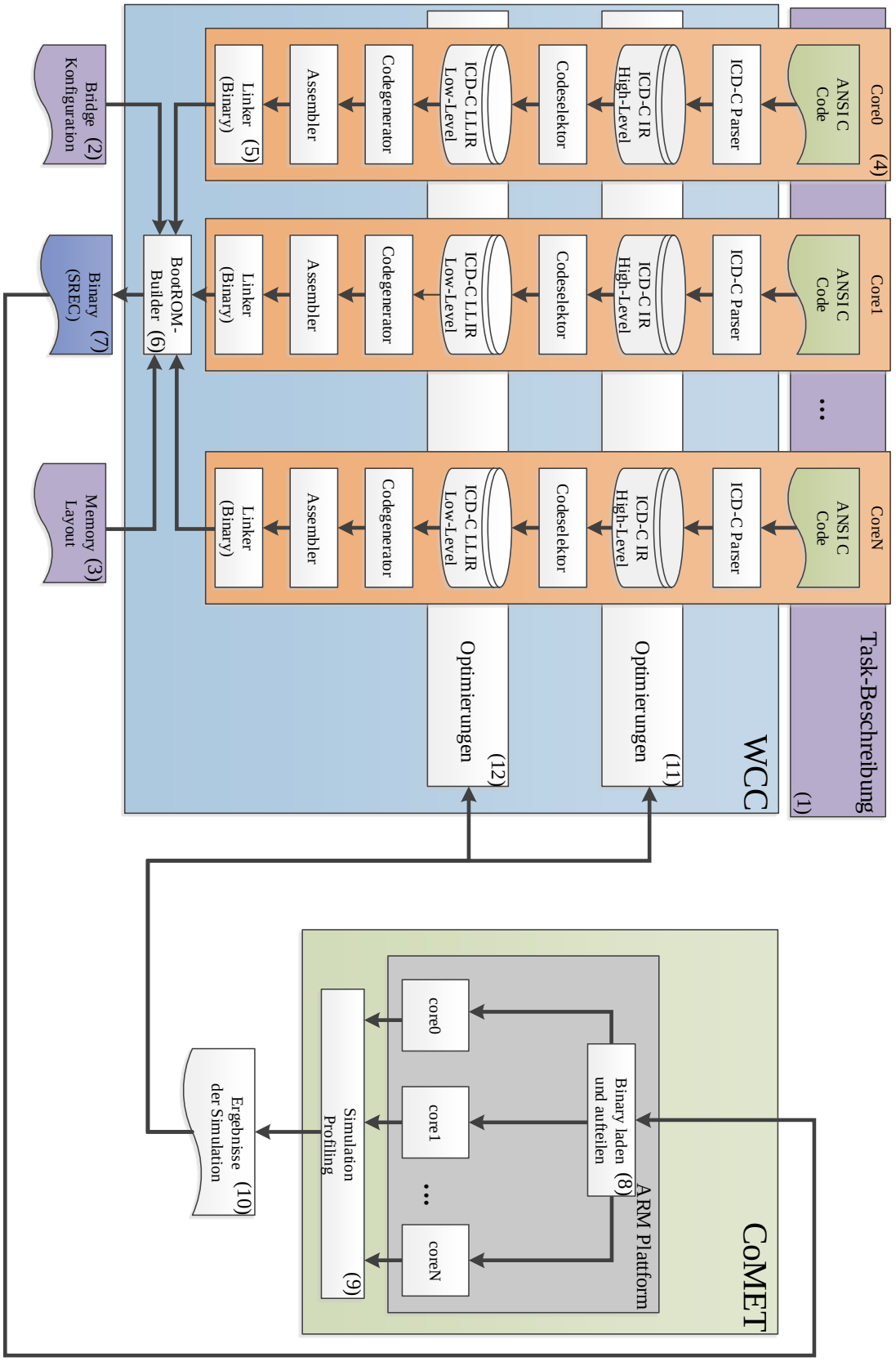


Abbildung 4.1: Integration in den WCC.

4.1 Task-Beschreibungen

Grundlage für die Simulation sind sog. Task-Beschreibungen, in der alle notwendigen Informationen zu den auszuführenden Benchmarks und der Verteilung auf die jeweiligen Cores aufgeführt sind. Abbildung 4.2 zeigt den Aufbau und die notwendigen Parameter einer solchen Task-Beschreibung.
Jeder Task besteht dabei aus einem Task-Namen (<name>...</name>), einem zugeordneten Core (<core>...</core>) und den zugehörigen Quellen (<source>...</source>), in der jede Quelldatei jeweils durch separate <file>...</file>-Tags eingebunden ist. Für jeden Task kann eine zugehörige *.wccrc*-Datei angegeben werden (<wccrc>...</wccrc>), in der weitere taskspezifische Konfigurationsparameter enthalten sind.

```
<task>
  <name>sqrt</name>
  <core>0</core>
  <sources>
    <file>../../../MRTC/sqrt/sqrt.c</file>
  </sources>
  <wccrc>../../../MRTC/sqrt/.wccrc</wccrc>
</task>
<task>
  <name>fdct</name>
  <core>1</core>
  <sources>
    <file>../../../MRTC/fdct/fdct.c</file>
  </sources>
  <wccrc>../../../MRTC/fdct/.wccrc</wccrc>
</task>
...
```

Abbildung 4.2: Beispiel einer Task-Beschreibung.

Diese Task-Beschreibungen dienen als Eingabe für den WCC, der hieraus die notwendigen Informationen extrahiert, um die einzelnen Core-Binaries zu generieren. Zusätzlich wurde eine Schnittstelle geschaffen, um die einzelnen Konfigurationsparameter der Bridge konfigurieren zu können. Diese Schnittstelle wird im folgenden Abschnitt erläutert.

4.2 Konfiguration der Arbitration Bridge

Grundlage für die Konfiguration der Bridge bilden die sog. *.wccrc*-Dateien. Der WCC wertet die Dateien nach einer vordefinierten Reihenfolge aus:

1. globale wccrc für ARM7-Multicore (aus dem Installationsverzeichnis des WCC)

2. Heimatverzeichnis des Benutzers

3. aktuelles Arbeitsverzeichnis

4. Taskspezifische .wccrc

Diese Konfigurationsdateien werden bei einem Aufruf des WCC automatisch eingelesen, analysiert und bekannte Konfigurationsparameter ausgewertet. Die mehrfache Konfiguration eines identischen Parameters wird dabei durch das jeweils spätere Vorkommen überschrieben. Durch diesen Mechanismus kann der Compiler komfortabel und einfach konfiguriert werden.
Im Fall der Bridge-Konfiguration enthalten die *.wccrc*-Dateien alle Konfigurationsparameter, die für den jeweiligen Simulationslauf verwendet werden sollen. Die so übergebenen Parameter werden vom WCC ausgewertet und in Form der Header-Datei *hardware_ setup.h* exportiert. Diese Header-Datei wird durch die Datei *hardware_ setup.c* inkludiert, die kompiliert und zu jedem Core-Binary hinzugelinkt wird. Nach Laden der Core-Binaries wird die Funktion *do_ hardware_ setup()* durch den WCC-generierten Startup-Code aufgerufen, die die Memory-Mapped-Register der Bridge verwendet, um die jeweiligen Konfigurationsparameter zu setzen.

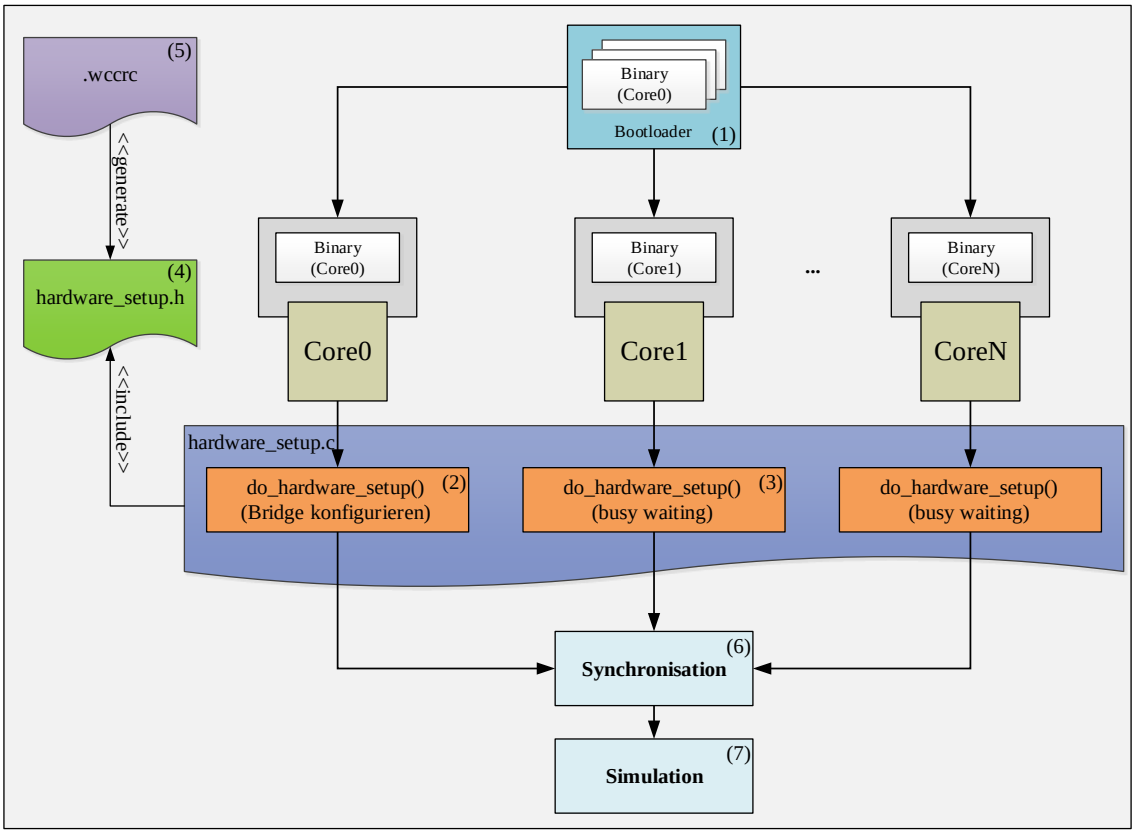


Abbildung 4.3: Ablauf des Hardware-Setups.

Der genaue Ablauf wird in Abbildung 4.3 näher verdeutlicht. Der Bootloader lädt die einzelnen Core-Binaries in den privaten Speicherbereich der jeweiligen Cores (1). Nach dem erfolgreichen Laden führen die Cores die Binaries aus, insbesondere wird die Funktion *do_hardware_setup()* aufgerufen. Während Core0 innerhalb dieser Funktion die Arbitration Bridge konfiguriert (2), warten die anderen Cores, bis das Hardware-Setup abgeschlossen ist (3). Die dazu notwendigen Konfigurationsparameter werden aus der Datei *hardware_setup.h* inkludiert, die durch den WCC aus den zugehörigen *.wccrc*s generiert wurde (5). Nach erfolgreicher Synchronisation (6) wird die eigentliche Simulation der Benchmarks durchgeführt (7).

Kapitel 5

Benchmarks

Bei den verwendeten Benchmarks handelt es sich hauptsächlich um Kombinationen von statischen und voneinander unabhängigen Singlecore-Benchmarks. Innerhalb dieses Kapitels sollen zunächst in Unterkapitel 5.1 die einzelnen Benchmark-Suiten kurz vorgestellt werden. Im anschließenden Unterkapitel 5.2 erfolgt die Vorstellung des Frameworks zur Automatisierung der Benchmarks (5.2.1); anschließend werden die durchgeführten Testläufe, die sich in die Referenz-Messungen (Kapitel 5.2.2) und die eigentlichen Multicore-Messungen (Kapitel 5.2.3) unterteilen, vorgestellt.

5.1 Benchmark-Suiten

Die folgende Übersicht zeigt die verwendeten Benchmark-Suiten und gibt einen kurzen Einblick in die enthaltenen Benchmarks.

- **DSPstone** [25]: Die DSPstone-Benchmarksuite wurde entwickelt, um aussagekräftige Benchmark-Ergebnisse explizit für *Digitale Signalprozessoren* (kurz: DSP-Prozessoren) generieren zu können. Aufgrund der Tatsache, dass bis dahin existierende Computer-Benchmarks für das Benchmarking von DSP-Prozessoren wegen grundverschiedener Operationsarten ungeeignet waren, wurde die DSPstone mit explizit für DSP-Prozessoren geeigneten Benchmarks entwickelt.

  Die einzelnen Benchmarks unterteilen sich in Kernel-Tasks (z. B. digitale Filter, Fourier-Transformationen, etc.), C-Kernel-Benchmarks (Schleifen, Funktionsaufrufe, etc.) sowie komplexere Applikationen.

- **MediaBench** [10]: Bei der MediaBench-Benchmarksuite handelt es sich um eine Suite zur Evaluierung von Multimedia- und Kommunikationssystemen. Diese Benchmarks sind speziell für die Verwendung auf *Very Long Instruction Word-Prozessoren* (kurz: VLIW-Prozessoren) bzw. *Single Instruction Multiple Data-Prozessoren* (kurz:

SIMD-Prozessoren) abgestimmt. Dies beinhaltet beispielsweise Benchmarks zur GSM-Transkodierung, MPEG2 En- und Dekodierung sowie JPEG-Kompression.

- **MiBench** [8]: Die MiBench-Benchmarksuite ist eine frei verfügbare Suite für das Benchmarking eingebetteter Systeme. Da die Anforderungen in unterschiedlichen Domänen sehr divergieren können, enthält die Benchmarksuite insgesamt 35 Benchmarks aus den sechs Domänen Steuerungssysteme (im Automotive-Bereich sowie der Industrie), Endverbraucher-Geräte (PDAs, Digitalkameras, etc.), Büro-Automatisierung (Drucker, Faxgeräte, Textmanipulationsalgorithmen), Netzwerke (CRC, SHA, etc.), Sicherheit (Ver- und Entschlüsselung, Hashing) und Telekommunikation (Fourier-Transformationen, GSM En-/Dekodierung, etc.).

- **MRTC** [17]: Die MRTC-Benchmarksuite (Mälardalen Real-Time Research Centre) ist eine Sammlung von Benchmarks zum Vergleich und zur Bewertung von unterschiedlichen WCET-Analysemethoden. Zu diesem Zweck wurden die einzelnen Benchmarks von der MRTC-WCET-Gruppe zusammengetragen.

- **Netbench** [14]: Diese Benchmarksuite umfasst unterschiedliche Anwendungen, die speziell für *Netzwerk-Prozessoren* entwickelt wurden. Diese Anwendungen unterteilen sich in drei Kategorien: Micro-Level-Programme (z. B. CRC-Berechnung), IP-Level-Programme (z. B. Routing, Network Address Translation, etc.) und Application-Level-Programme (z. B. Diffie-Hellman, MD5-Berechnung, etc.).

- **StreamIT** [23]: StreamIT ist eigentlich eine Kombination einer proprietären Programmiersprache und Compiler-Infrastruktur zur Entwicklung von komplexen Streaming-Anwendungen. Die hier verwendeten C-Benchmarks dienen zur Bewertung von Streaming-Optimierungen und neu entwickelten Streaming-Architekturen.

- **UTDSP** [11]: Das Ziel der UTDSP-Benchmarksuite ist die Bewertung von Code, der aus High-Level Beschreibungen (wie z. B. C) speziell für DSP-Prozessoren generiert wurde. Sie wurde hauptsächlich entwickelt, um die Entwicklung von Compiler-Optimierungen in diesem Bereich voranzutreiben oder Änderungen an der eigentlichen System-Architektur zu testen und zu bewerten.

  Die einzelnen Benchmarks unterteilen sich dabei in Kernel-Tasks (Filter, Fourier-Transformationen, Matrix-Multiplikationen) und fertige Anwendungen (Kompression, G.721-Modem-Protokoll, etc.), die die einzelnen Kernel-Tasks verwenden.

5.2 Messverfahren

Innerhalb dieses Kapitels sollen die Automatisierung der Benchmarks und die durchgeführten Messungen (sowohl Singlecore-Referenzmessungen als auch die tatsächlichen Multicore-Messungen) kurz erläutert werden.

5.2.1 Automatisierung der Benchmarks

Um eine fundierte Basis für die spätere Auswertung der Ergebnisse zu schaffen, mussten eine Vielzahl von Benchmarks auf den unterschiedlichen Plattformen simuliert und die Ergebnisse konsolidiert werden. Um diese Problematik mit einem überschaubaren Aufwand zu bewältigen, wurden Mechanismen geschaffen, um die Benchmarks automatisch zu generieren, durchzuführen und anschließend die Ergebnisse strukturiert abzuspeichern.

1. **Generierung der Benchmark-Ordnerstruktur**

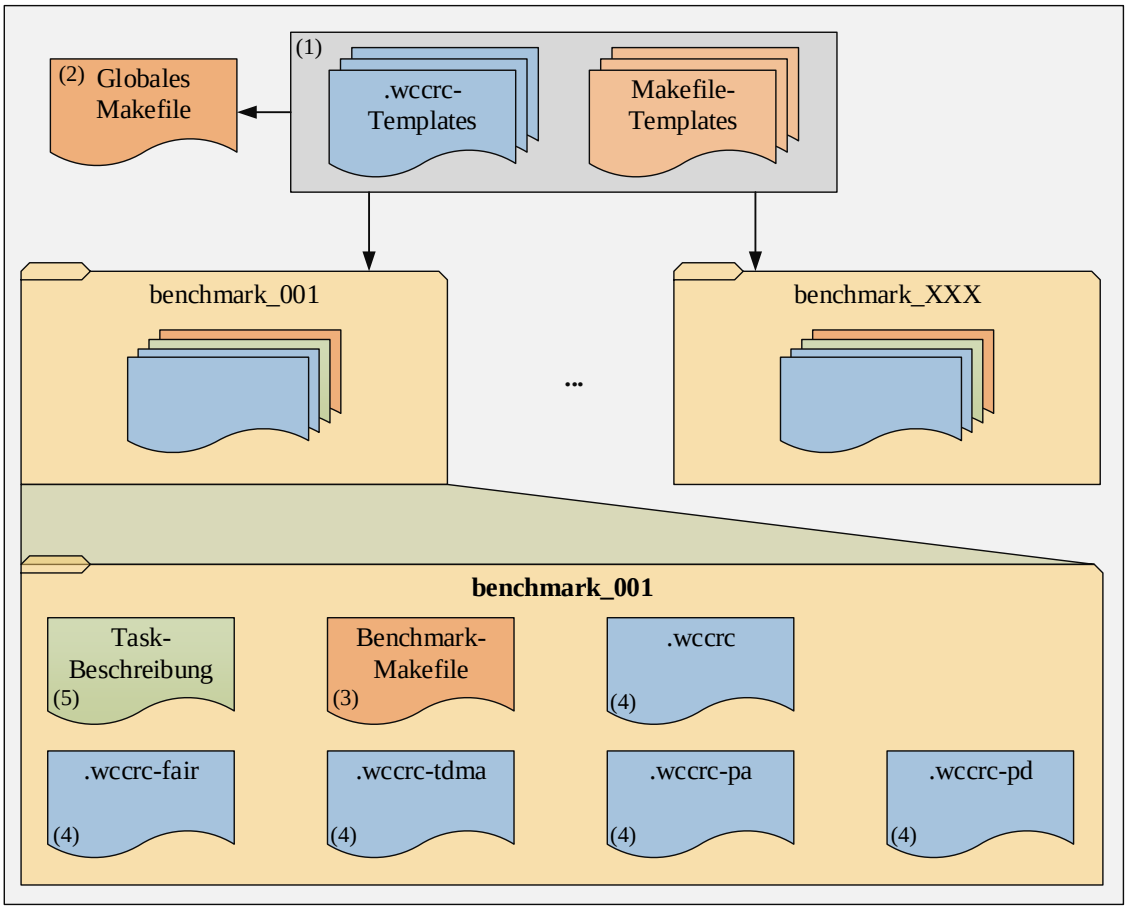


Abbildung 5.1: Generierung der Benchmark-Ordnerstruktur.

Im ersten Schritt wird die Benchmark-Ordnerstruktur generiert (siehe Abbildung 5.1). Als Basis hierfür dienen *.wccrc*-Templates (1) für die einzelnen Arbitrierungsverfahren (*.wccrc-fair*, *.wccrc-tdma*, *.wccrc-pa* und *.wccrc-pd*) sowie eine gemeinsame *.wccrc*, die alle Konfigurationsparameter für die jeweiligen Arbitrierungsverfahren oder allgemeingültige Konfigurationsparameter wie beispielsweise die Cachegrößen enthalten.

Das globale Makefile (2) wird in Abhängigkeit von der Anzahl der durchzuführenden Benchmarks generiert und dient dazu, die einzelnen Benchmarks systematisch aus-

zuführen. Die jeweiligen Benchmark-Makefiles (3) werden ebenfalls aus einem Template generiert. Diese Makefiles dienen zur Generierung des SREC-Binary aus der zum Benchmark gehörigen Task-Beschreibung (siehe Abbildung 4.1) und zum Start der eigentlichen Simulation. Die einzelnen *.wccrc*s für jeden Benchmark werden aus den entsprechenden Templates generiert.

Die Generierung der Task-Beschreibungen (5) für die Multicore-Plattformen ist in Abbildung 5.2 am Beispiel von vier Benchmarks (a,b,c und d) und der X2-Plattform als Zielplattform näher erläutert.

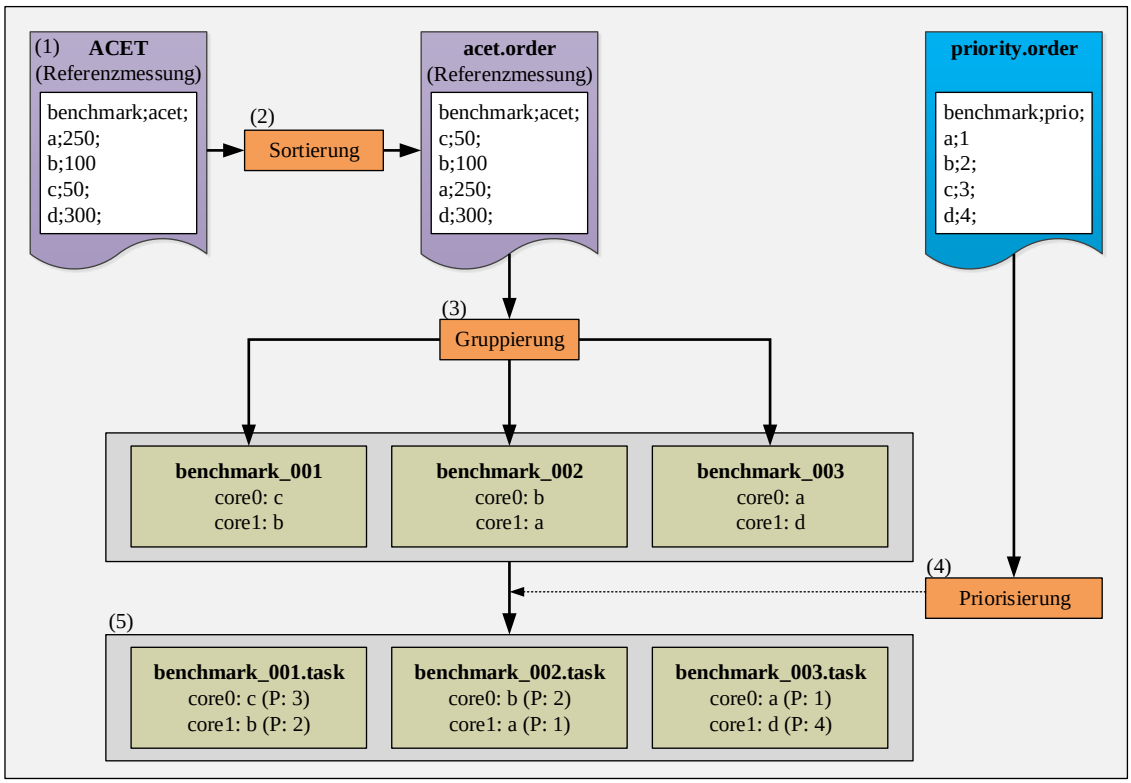


Abbildung 5.2: Generierung der Task-Beschreibungen.

Bei der Generierung der Multicore-Benchmarks wurden Referenzmessungen zur ACET der einzelnen Benchmarks auf der X1-Plattform zu Grunde gelegt (1).

Die einzelnen Singlecore-Benchmarks werden aufsteigend nach den gemessenen Ausführungszeiten sortiert (2) und durch die Sortierung übereinander stehende Benchmarks jeweils in Zweiergruppen (X2-Plattform) zusammengefasst (3). Diese Gruppierung dient dazu, um nach Möglichkeit Singlecore-Benchmarks mit ähnlicher Laufzeit gleichzeitig auf den Multicore-Plattformen zur Ausführung zu bringen. Dies ist notwendig, damit aussagekräftige Messungen zum Einfluss von Arbitrierungsverfahren auf das Verhalten in Multicore-Systemen überhaupt möglich sind.

Nach der Gruppierung der einzelnen Singlecore-Benchmarks erfolgt noch eine Priorisierung der einzelnen Core-Anwendungen (4) für Arbitrierungsverfahren PA und PD. Mögliche Kriterien für die Priorisierung sind beispielsweise die Busauslastung im Singlecore-Benchmark (Anwendungen mit geringer Bus-Auslastung kann eine höhere Priorität zugeteilt werden, damit diese die wenigen Speicherzugriffe zeitnah absetzen können und somit nicht unnötig verzögert werden).

Die so generierten Task-Beschreibungen (5) stellen die eigentlichen Multicore-Benchmarks dar.

2. **Simulation**

   Wie bereits angedeutet wurde, erfolgt der Start der Simulation der generierten Benchmarks durch das globale Makefile sowie die generierten Benchmark-Makefiles.

3. **Konsolidierung der Ergebnisse**

   Die Konsolidierung und das systematische Abspeichern der Ergebnisse ist ein wichtiges Thema, damit die immense Menge an gesammelten Daten überhaupt handhabbar bleibt. Die eigentlichen Messergebnisse und durch die Simulation generierten Leistungsmerkmale (siehe Abschnitt 3.2.3) werden für jeden Benchmark und jedes Arbitrierungsverfahren in einer separaten Log-Datei gespeichert.

   Um die einzelnen Messergebnisse zu konsolidieren wurde ein Skript geschrieben, dass die jeweiligen Kennziffern aus den einzelnen Log-Dateien extrahiert und systematisch in einer SQLite-Datenbank [22] ablegt - auf die Erläuterung der genauen Funktionsweise dieses Skripts soll an dieser Stelle aber verzichtet werden. Die in dieser Datenbank gesammelten Daten wurden dann für die eigentliche Auswertung der Ergebnisse verwendet.

5.2.2 Singlecore-Messungen (Referenz)

Ziel der Arbeit ist die Analyse von unterschiedlichen Arbitrierungsverfahren hinsichtlich ihres Einflusses auf die Ausführungszeiten von Anwendungen und die Auslastung der gemeinsamen Komponenten. Um die jeweiligen Referenzwerte der Ausführungszeiten und Auslastung zu ermitteln, wurden die einzelnen Benchmarks dabei zunächst auf der erstellten X1-Plattform ausgeführt. Die Ausführung auf der X1-Plattform ermöglicht dabei eine Messung der ACET und Auslastung ohne den potenziell störenden Einfluss weiterer Bus-Master.

5.2.3 Multicore-Messungen

Die Generierung der Multicore-Benchmarks wurde in Abschnitt 5.2.1 bereits ausführlich beschrieben. Neben der Gruppierung der einzelnen Singlecore-Benchmarks wurden zwei

unterschiedliche Priorisierungsverfahren verwendet, um die Prioritäten (für PA und PD) der einzelnen Core-Anwendungen festzulegen: Inverse Busauslastung (eng.: Inverse Utilization, kurz: IU) und $ACET + ACET \cdot UTILIZATION$ (kurz: AAU). Diese beiden Verfahren sollen zunächst definiert und kurz erläutert werden.
Sei B die Menge der Singlecore-Benchmarks und $M = \{b_1, b_2, \ldots, b_n\}$ mit $M \subset B$ und $n \in \{2, 4, 8\}$ ein Multicore-Benchmark für eine Plattform mit n Cores. Seien weiterhin $a : B \to \mathbb{N}$ und $u : B \to [0, 1]$ Funktionen, die die einzelnen Singlecore-Benchmarks $b \in B$ auf ihre ACET bzw. Busauslastung bei den Referenzmessungen abbilden. Aufgrund des Gruppierungsverfahrens der Benchmarks gilt: $a(b_1) \leq a(b_2) \leq \cdots \leq a(b_n)$. Sei $P = \{p_1, p_2, \ldots, p_n\}$ die Menge der Prioritäten mit $p_1 < p_2 < ... < p_n$.
Die beiden Priorisierungsverfahren sind dann definieren durch die bijektiven Funktionen t_{IU} und t_{AAU}. Die Bijektivität beruht auf der Tatsache, dass innerhalb jedes Multicore-Benchmarks eine eindeutige Zuordnung zwischen den unterschiedlichen Prioritäten P und der Menge der ausgeführten Singlecore-Benchmarks M existiert.

- **IU**: Dieses Verfahren wird definiert durch die Funktion $t_{IU} : M \to P$ mit der folgenden Eigenschaft:

  - $u(t_{IU}^{-1}(p_1)) \geq u(t_{IU}^{-1}(p_2)) \geq ... \geq u(t_{IU}^{-1}(p_n))$.

  Bei diesem Priorisierungsverfahren wird also die Gesamtauslastung des gemeinsamen Busses im Singlecore-Benchmark zu Grunde gelegt. Die Verteilung der Prioritäten erfolgt invers zur Busauslastung, Anwendungen mit einer höheren Busauslastung erhalten eine niedrigere Priorität, wohingegen Anwendungen mit niedriger Busauslastung eine höhere Priorität zugeordnet wird.

- **AAU**: Sei $f : B \to \mathbb{R}_+$ mit $f(b) = a(b) + a(b) \cdot u(b)$. Dieses Verfahren wird definiert durch die Funktion $t_{AAU} : M \to P$ mit der folgenden Eigenschaft:

  - $f(t_{AAU}^{-1}(p_1)) \geq f(t_{AAU}^{-1}(p_2)) \geq ... \geq f(t_{AAU}^{-1}(p_n))$.

  Die Grundlage für dieses Verfahren bildet die Belastungskennziffer f, die sich aus der ACET und Busauslastung der Core-Anwendung im korrespondierenden Singlecore-Benchmark zusammensetzt. Benchmarks mit hoher Belastungskennziffer erhalten dabei niedrigere Prioritäten als Benchmarks mit niedriger Belastungskennziffer.

Diese beiden Priorisierungsverfahren verfolgen zwei grundlegend verschiedene Ansätze bei der gleichen Zielsetzung: die unterschiedliche Laufzeit der gruppierten Core-Anwendungen so gut wie möglich auszugleichen. Welcher der beiden Ansätze sich im Falle der hier verwendeten Benchmarks besser eignet ist dabei im Vorhinein nicht absehbar und soll ebenfalls durch die Analyse und Auswertung der Messergebnisse bestimmt werden.

Kapitel 6

Auswertung

Im nachfolgenden Kapitel sollen die gesammelten Messergebnisse dargestellt und analysiert werden. Innerhalb dieses Kapitels erfolgt zunächst eine Betrachtung des Jitters, wie er bei der Gruppierung von einzelnen Singlecore-Anwendungen mit unterschiedlicher Laufzeit zu einem Multicore-Benchmark beobachtet werden kann. Anschließend werden die Gesamtauslastung des gemeinsamen Busses betrachtet und die Wartezeiten analysiert, die durch die unterschiedlichen Arbitrierungsverfahren beim Zugriff auf den gemeinsamen Bus verursacht werden. Abschließend erfolgt eine Auswertung zur ACET der unterschiedlichen Benchmarks. Die detaillierten Messergebnisse der Singlecore-Benchmarks befinden sich im Anhang A.1, die Messergebnisse der Multicore-Benchmarks sind im Anhang A.2 aufgeführt.

6.1 Referenzlaufzeiten und Jitter

Da es sich bei den ausgeführten Core-Anwendungen um unabhängige Singlecore-Benchmarks mit unterschiedlicher Laufzeit handelt, sind allgemeingültige Aussagen bezüglich ACET und Gesamtauslastung bei der Übertragung auf die Multicore-Plattformen schwierig. Durch die Gruppierung der Singlecore-Benchmarks nach Ausführungszeit und Verwendung von unterschiedlichen Priorisierungsverfahren wurde versucht, die Laufzeit der daraus resultierenden Multicore-Benchmarks weitestgehend anzugleichen.

Abbildung 6.1 zeigt zunächst die Verteilung der Laufzeiten der unterschiedlichen Singlecore-Benchmarks. Jeder Balken repräsentiert dabei einen einzelnen Benchmark, die unterschiedlichen Laufzeiten liegen zwischen $2{\cdot}10^2$ und $0.6{\cdot}10^9$ Zeiteinheiten. Diese extrem unterschiedlichen Laufzeiten verursachen bei der Gruppierung zu Multicore-Benchmarks Probleme, auf die im folgenden Abschnitt näher eingegangen werden soll.

Zunächst soll an dieser Stelle der Begriff des *Jitters* definiert und eine kurze Analyse des bei den einzelnen Multicore-Benchmarks gemessenen Jitters erfolgen. Abbildung 6.2 illustriert die grundlegende Idee dieser Kennziffer.

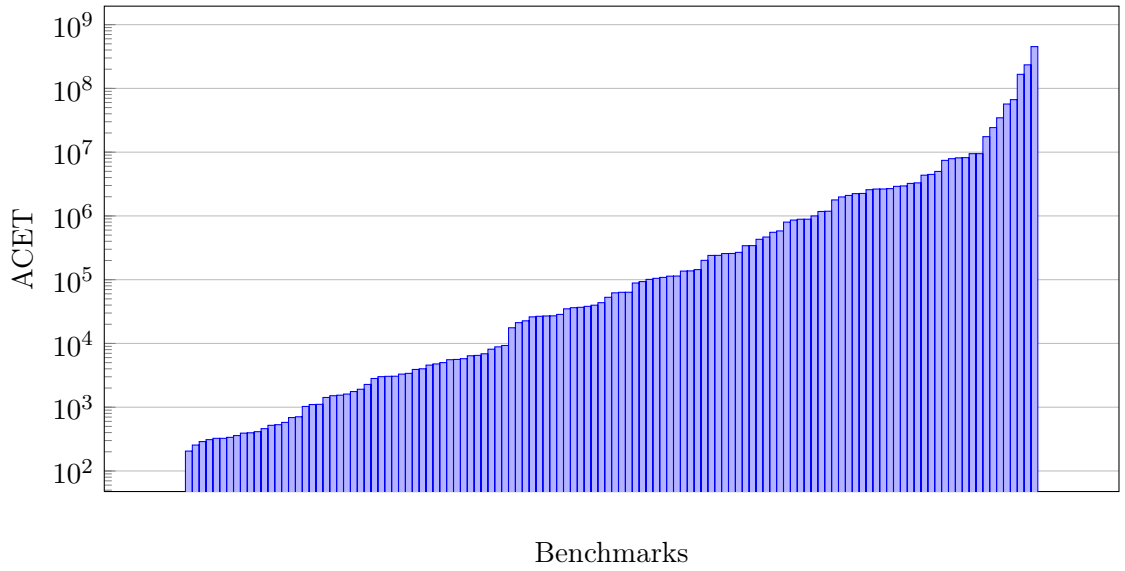


Abbildung 6.1: Laufzeiten der unterschiedlichen Singlecore-Benchmarks.

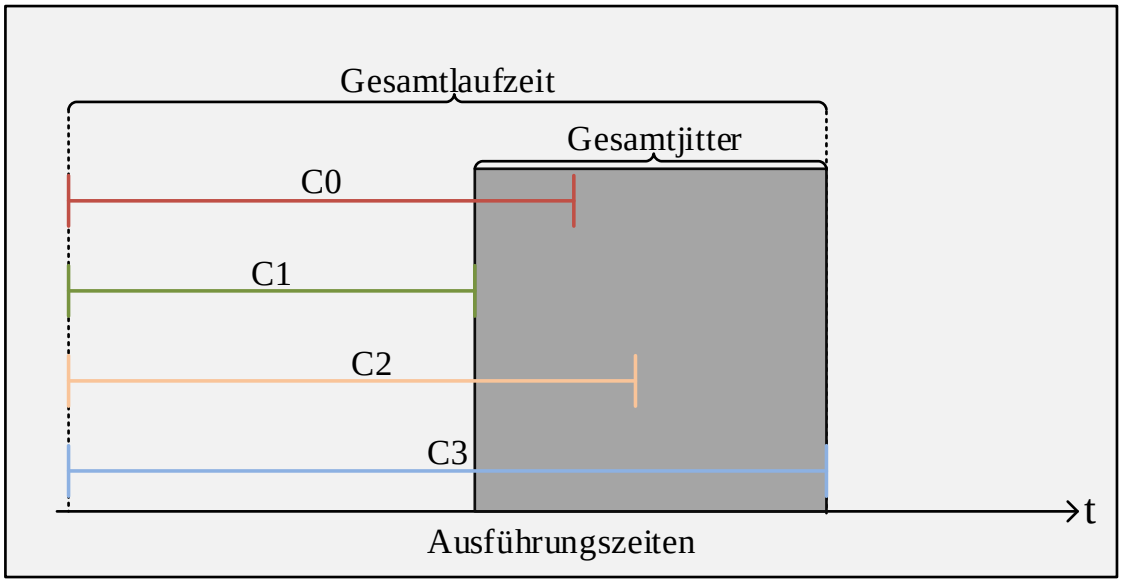


Abbildung 6.2: Veranschaulichung des Jitters.

Als *Jitter* wird die Differenz zwischen dem Endzeitpunkt der Anwendung, die ihre Ausführung innerhalb eines einzelnen Multicore-Benchmarks als letztes beendet hat und der Anwendung, die ihre Ausführung zuerst beendet hat, bezeichnet. Sei $t : C \rightarrow \mathbb{N}$ eine Funktion, die jedem Core $c \in C$ den Endzeitpunkt ihrer Ausführung innerhalb eines einzelnen Multicore-Benchmarks zuordnet. Der *Jitter* ist dann definiert als: $max_{\forall c \in C}(t(C)) - min_{\forall c \in C}(t(C))$. Im Prinzip beschreibt der Jitter das Zeitfenster innerhalb eines Multicore-Benchmarks, in dem nicht mehr alle Cores Transaktionen über den gemeinsamen Bus ab-

setzen. Die Bestimmung und Analyse des Jitters soll dabei helfen, die Ergebnisse bezüglich ACET und Gesamtauslastung des gemeinsamen Busses besser einordnen zu können.

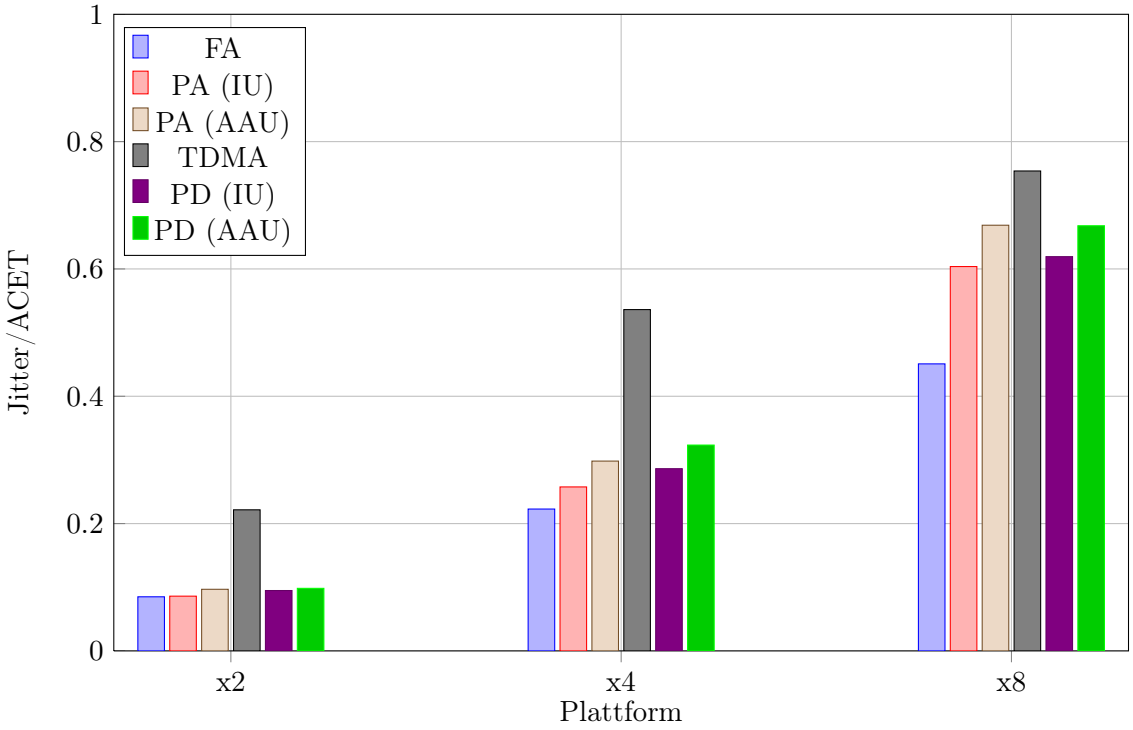


Abbildung 6.3: Durchschnittlicher Jitter der einzelnen Benchmarks (gruppiert nach Plattform).

Abbildung 6.3 zeigt den durchschnittlichen Jitter, der sich bei den einzelnen unterschiedlichen Multicore-Benchmarks angesammelt hat. Der Jitter wurde pro Benchmark relativ zur ACET des jeweils betrachteten Multicore-Benchmarks betrachtet und berechnet sich somit durch $\frac{max_{\forall c \in C}(t(C)) - min_{\forall c \in C}(t(C))}{ACET}$.

Es zeigt sich, dass der Jitter mit steigender Core-Anzahl deutlich wächst, was ein Verfahren wie TDMA bei den durchgeführten Messungen benachteiligt, da hier eindeutige Zuordnungen zwischen einzelnen Cores und den korrespondierenden TDMA-Slots existieren. Die Slots, die von bereits beendeten Anwendungen nicht mehr benötigt werden, bleiben frei und werden nie mit Transaktionen anderer Anwendungen aufgefüllt. Dies resultiert bei genügend großem Jitter vermutlich in einer deutlich sinkenden Auslastung und einer somit deutlich schlechteren ACET.

Andere Verfahren, bei denen eine solche explizite Zuordnung zwischen einzelnen Zeitschlitzen und Core-Anwendungen nicht existiert, werden von einem hohen Jitter voraussichtlich weniger stark beeinflusst. Insgesamt ist zu erwarten, dass die Auslastung des gemeinsamen Busses ebenfalls sinkt, da innerhalb des Zeitfensters, das durch den Jitter definiert wird, weniger Core-Anwendungen als vorher Anfragen an den gemeinsamen Speicher stellen und sich somit die Wahrscheinlichkeit erhöht, dass der gemeinsame Bus ungenutzt bleibt.

Des Weiteren ist anhand des Jitters eine Bewertung der unterschiedlichen Priorisierungsverfahren möglich, die bei PD und PA eingesetzt werden. In Abbildung 6.3 zeigt sich, dass in den X2-Benchmarks beide Priorisierungsverfahren annähernd gleiche Ergebnisse liefern, während in den X4- und X8-Benchmarks AAU durch IU klar dominiert wird.

6.2 Auslastung

Abbildung 6.4 zeigt die durchschnittliche Gesamtauslastung der Plattform. Bei der Betrachtung des Diagramms zeigt sich deutlich, dass die unterschiedlichen Arbitrierungsverfahren mit Ausnahme von TDMA mit zunehmender Core-Anzahl den Bus immer mehr auslasten, so dass der gemeinsame Bus bei den X8-Benchmarks durchschnittlich zu fast 70% ausgelastet ist.
Bei der Arbitrierung mit TDMA verhält sich die Gesamtauslastung gegenläufig, so dass diese mit zunehmender Core-Anzahl zunächst relativ konstant bei 20% bleibt und bei den X8-Benchmarks sogar noch hinter der Auslastung des X1-Benchmarks zurückbleibt. Diese Entwicklung ist vermutlich auf den großen Jitter bei der Verwendung von TDMA auf den unterschiedlichen Multicore-Plattformen zurückzuführen. Beim Einsatz von TDMA ist die möglichst gleichmäßige Balancierung der unterschiedlichen Benchmarks von zentraler Bedeutung, da nur so eine hohe Auslastung des gemeinsamen Busses erreicht werden kann.

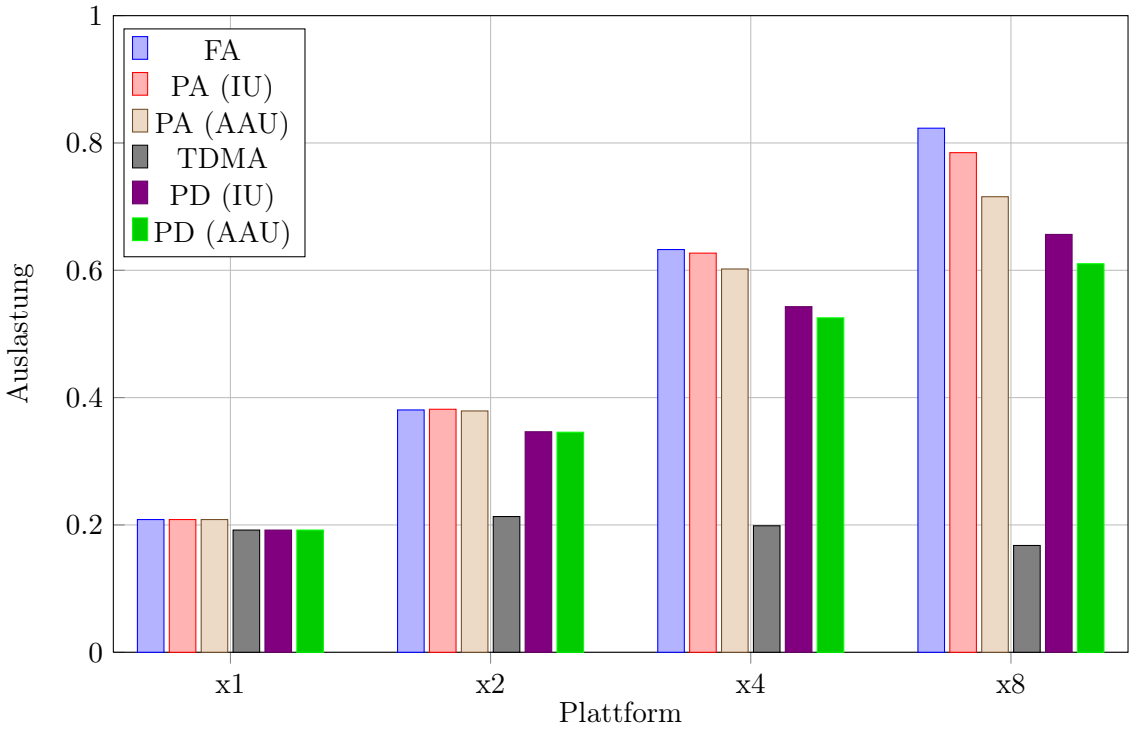


Abbildung 6.4: Vergleich der durchschnittlichen Gesamtauslastung des gemeinsamen Busses durch die einzelnen Benchmarks (gruppiert nach Plattform).

Die Abbildungen 6.5 - 6.8 zeigen neben der durchschnittlichen Auslastung des Busses ebenfalls die Minimal- bzw. Maximalauslastung durch die ausgeführten Benchmarks und ermöglichen einen direkten Vergleich pro Arbitrierungsverfahren zwischen den unterschiedlichen Plattformen.

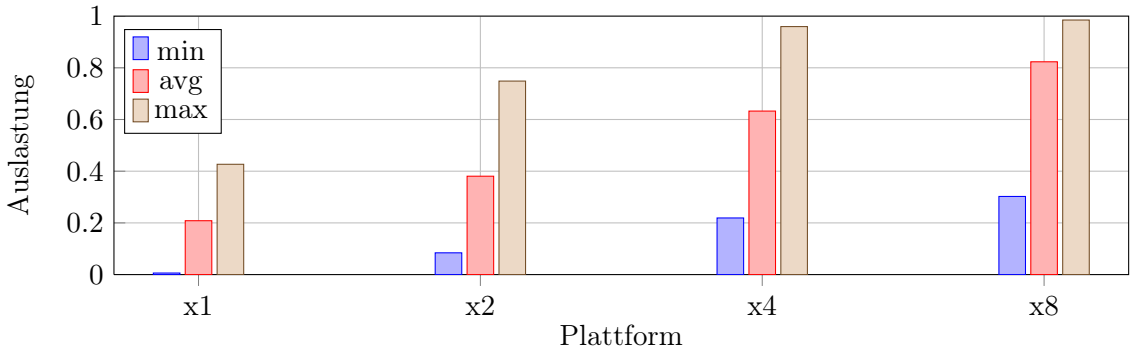


Abbildung 6.5: Gesamtauslastung des Busses bei Arbitrierung mit FA.

Bei der Verwendung von FA (siehe Abbildung 6.5) zeigt sich, dass sich die durchschnittliche Gesamtauslastung des gemeinsamen Busses mit zunehmender Core-Anzahl immer mehr an die Maximalauslastung - die in diesem Falle bei der X8-Plattform beinahe bei 100% liegt - annähert. Die durchschnittliche Auslastung liegt bei der X8-Plattform bei knapp über 80%. Dieses Verhalten ist darauf zurückzuführen, dass bei steigender Core-Anzahl mehr konkurrierende Zugriffe entstehen und somit immer ausreichend Transaktionen warten, um den Bus beinahe komplett auszulasten. Der zusammen mit der Core-Anzahl steigende Jitter bei den einzelnen Multicore-Benchmarks beeinflusst die durchschnittliche Gesamtauslastung des Busses negativ, so dass die Auslastung des Busses sinkt.

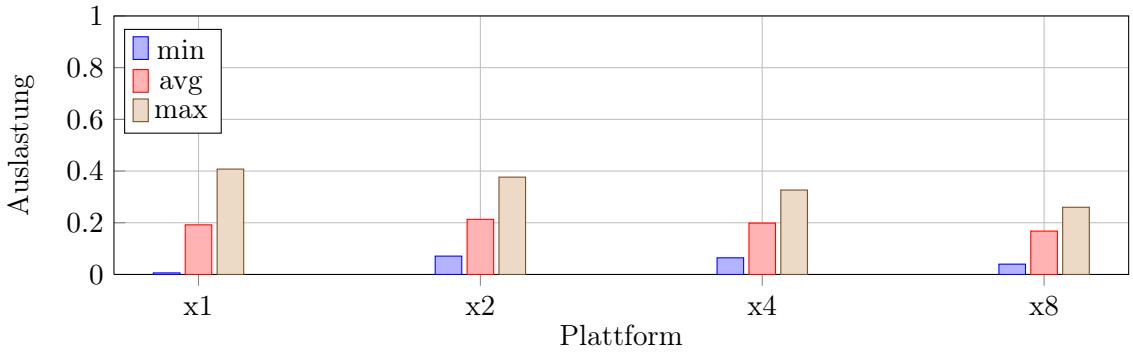
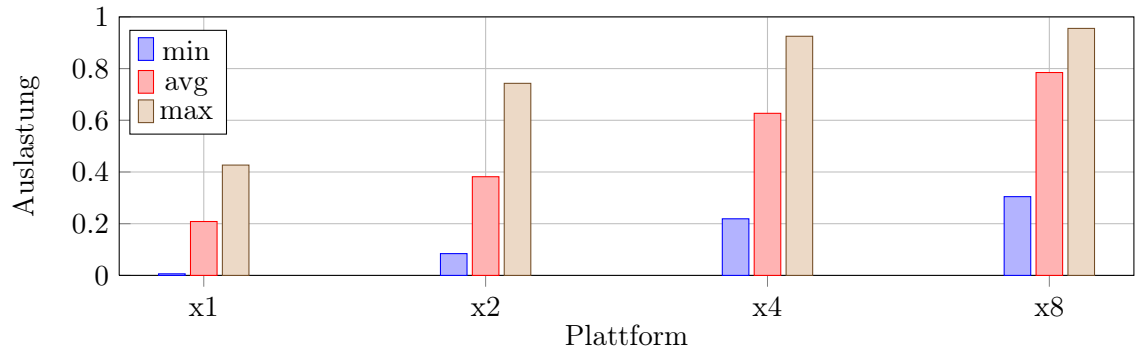
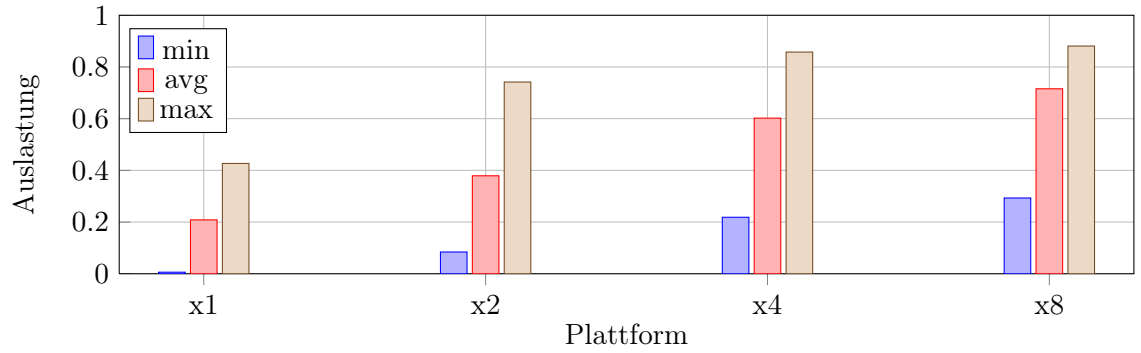


Abbildung 6.6: Gesamtauslastung des Busses bei Arbitrierung mit TDMA.

Bei der Verwendung von TDMA (siehe Abbildung 6.6) zeigt sich ein völlig konträres Verhalten. Die durchschnittliche Gesamtauslastung bleibt in allen Benchmarks relativ konstant, wohingegen die maximale Gesamtauslastung mit steigender Core-Anzahl stetig sinkt. Hier zeigt sich, dass die Verwendung von TDMA in Multicore-Systemen problematisch sein kann und trotz der extrem guten zeitlichen Vorhersagbarkeit nicht unbedingt geeignet ist.



(a) Gesamtauslastung PA mit IU.



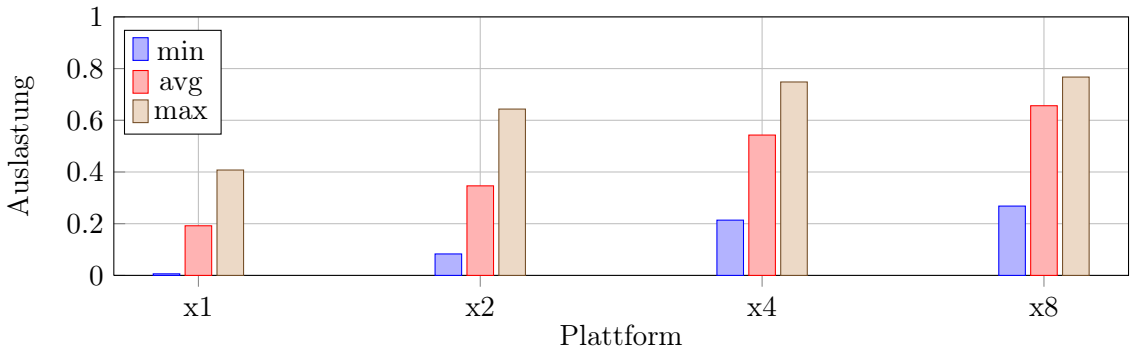
(b) Gesamtauslastung PA mit AAU.

Abbildung 6.7: Gesamtauslastung des Busses bei Arbitrierung mit PA.

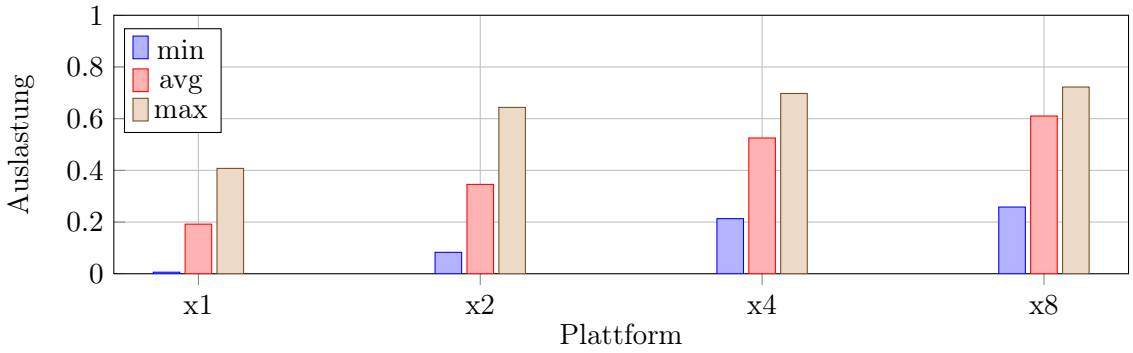
Abbildung 6.7 zeigt den Vergleich der Gesamtauslastung zwischen den unterschiedlichen Plattformen bei der Verwendung von PA. Während die unterschiedlichen Priorisierungsschemata bei Ausführung der Benchmarks auf der X2-Plattform noch relativ ähnliche Ergebnisse bezüglich der Auslastung produzieren, erzeugt IU bei steigender Core-Anzahl minimal bessere Ergebnisse, was sich an der steigenden durchschnittlichen und maximalen Gesamtauslastung ablesen lässt.

Hier zeigt sich - ähnlich wie bei der Verwendung von FA - erneut das Verhalten, dass sich die durchschnittliche Gesamtauslastung des gemeinsamen Busses mit steigender Core-Anzahl der maximalen Auslastung annähert und schließlich bei der X8-Plattform durchschnittlich eine fast 80%-ige Auslastung erreicht werden kann.

Bei der Verwendung von PD zeigt sich ein ähnliches Bild wie bei der Verwendung von PA. Abbildung 6.8 zeigt die Gesamtauslastung des gemeinsamen Busses bei der Verwendung von PD mit den unterschiedlichen Priorisierungsverfahren. Insgesamt ist der Einfluss des Priorisierungsschemas sichtbar, dieser ist aber weniger stark als bei PA. Die durchschnittliche Gesamtauslastung bleibt leicht hinter der von PA zurück. Dies beruht vermutlich auf der Tatsache, dass die einzelnen Slots wie bei TDMA häufig nicht komplett ausge-



(a) Gesamtauslastung PD mit IU.



(b) Gesamtauslastung PD mit AAU.

Abbildung 6.8: Gesamtauslastung des Busses bei Arbitrierung mit PD.

nutzt werden können, um die Ausführung der Anwendung im nachfolgenden Slot durch ein mögliches „Hineinragen“ von Transaktionen nicht zu beeinträchtigen.
Im Vergleich zu TDMA erzielt PD hingegen deutlich bessere Ergebnisse. Die Gesamtauslastung des Systems bleibt mit zunehmender Core-Anzahl nicht annähernd konstant, sondern steigt stattdessen deutlich. Insgesamt zeigt sich bei PD eine überraschend hohe Gesamtauslastung des gemeinsamen Busses.

6.3 Wartezeit

Abbildung 6.9 zeigt die durchschnittliche Wartezeit der einzelnen Cores bei der Ausführung der Multicore-Benchmarks. An dieser Stelle sei noch einmal daran erinnert, dass auf jedem Core pro Multicore-Benchmark genau eine Singlecore-Anwendung ausgeführt wird.
Die Wartezeit eines einzelnen Cores berechnet sich dabei als $\frac{delay-busy}{ACET}$ und gibt Aufschluss darüber, wie viel Prozent der ACET die einzelnen Cores durchschnittlich auf die Zuteilung des gemeinsamen Busses gewartet haben.
Hier zeigt sich erneut, dass die drei Arbitrierungsverfahren FA, PA und PD TDMA im Bezug auf die ACET und Auslastung deutlich überlegen sind. Die einzelnen Benchmarks

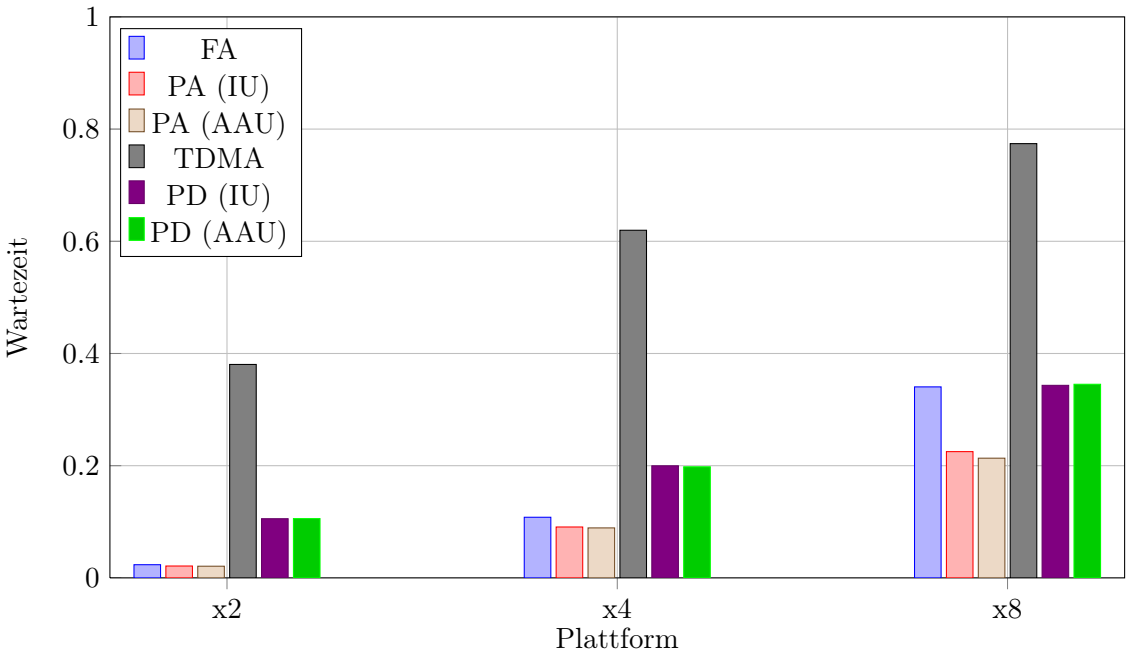


Abbildung 6.9: Durchschnittliche Wartezeit der einzelnen Cores im Verhältnis zur ACET (gruppiert nach Plattform).

warten bei der Verwendung von TDMA im Durchschnitt mehr als doppelt so lange auf die Zuteilung der gemeinsamen Ressource, auch wenn der extrem größere Jitter das Ergebnis an dieser Stelle zum Nachteil von TDMA verfälscht. Insgesamt ist allerdings absehbar, dass die Verwendung der anderen Verfahren eine deutlich geringere Wartezeit mit sich bringt. Die unterschiedlichen Priorisierungsverfahren haben auf allen Plattformen annähernd keinen Einfluss.

6.4 ACET

Innerhalb dieses Kapitels erfolgt eine Analyse der Auswirkung der unterschiedlichen Arbitrierungsverfahren auf die ACET. Abbildung 6.10 zeigt dabei die durchschnittliche ACET der Singlecore-Benchmarks in Abhängigkeit vom verwendeten Arbitrierungsverfahren und der verwendeten Ausführungsplattform.
Die durchschnittliche ACET pro Singlecore-Anwendung bei Ausführung auf der Multicore-Plattform wächst bei allen Arbitrierungsverfahren mit einer steigenden Anzahl von Cores. Diese Erhöhung der ACET hält sich bei FA, PA und PD in Grenzen, während sie sich bei der Verwendung von TDMA beim Vergleich von X2- und X4- bzw. X4- und X8-Plattform jeweils annähernd verdoppelt.
Ein interessantes Resultat ist auf den ersten Blick die Tatsache, dass auf der X4- und X8-Plattform trotz des kleinsten Jitters bei der Ausführung der Multicore-Benchmarks, der durchschnittlich höchsten Auslastung und der daraus resultierenden höheren durchschnitt-

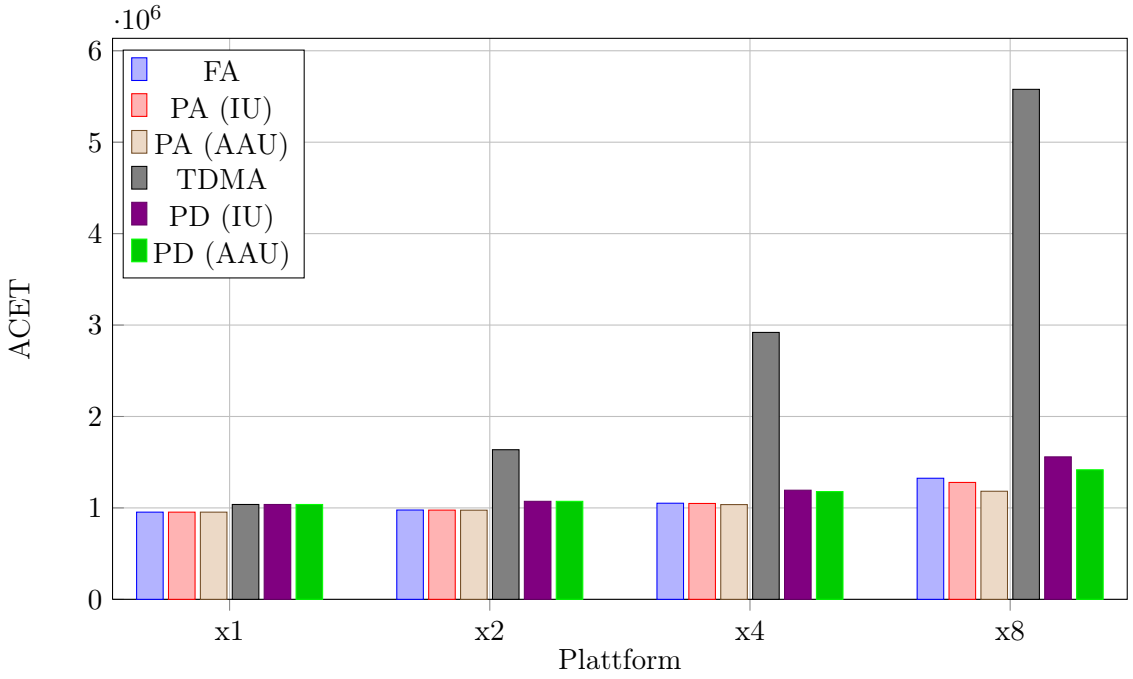


Abbildung 6.10: Durchschnittliche ACET der einzelnen Core-Anwendungen (gruppiert nach Plattform).

lichen Wartezeit der einzelnen Instruktionen die durchschnittliche ACET der Singlecore-Anwendungen bei FA minimal höher ist als bei der Verwendung von PA. Insgesamt wurde hier die Laufzeit der unterschiedlichen Benchmarks relativ gut aneinander angeglichen, was allerdings zu einer Verlängerung der durchschnittlichen ACET der einzelnen Core-Anwendungen geführt hat. Diese Verlängerung bei der Verwendung von FA kann dabei aber direkt durch den kleineren Jitter erklärt werden: Um den Jitter zu reduzieren, musste die Laufzeit von Anwendungen mit eigentlich kleinerer Ausführungszeit verlängert werden. Dies resultiert in der hier beobachteten Vergrößerung der durchschnittlichen ACET der einzelnen Core-Anwendungen.

Die unterschiedlichen Priorisierungsverfahren haben auf die durchschnittliche ACET der Core-Anwendungen einen deutlich geringeren Einfluss, als dies noch beim Vergleich der Gesamtauslastung und des Jitters zu beobachten war. Beim Vergleich der durchschnittlichen Ausführungszeiten auf der X4- bzw. X8-Plattform schneidet IU allerdings erneut besser ab als AAU, obwohl die Unterschiede in diesem Fall marginal sind.

Kapitel 7

Fazit und Ausblick

In dieser Arbeit wurde der Einfluss der unterschiedlichen Arbitrierungsverfahren auf die ACET und Gesamtauslastung einer gemeinsamen Komponente in eingebetteten Multicore-Systemen untersucht.

Zu Durchführung der notwendigen Messungen wurden dabei zunächst mehrere virtuelle Ausführungsplattformen entwickelt, die zur komfortablen Konfiguration und Simulation der Benchmarks in die bestehende Compiler-Infrastruktur des WCC integriert wurden. Die zentrale Komponente dieser Ausführungsplattform - die Arbitration Bridge - wurde dabei möglichst modular entwickelt, um in neuen Projekten wiederverwendet zu werden.

Die Erstellung der durchgeführten Benchmarks und Konsolidierung der Ergebnisse wurden weitestgehend automatisiert und eine Vielzahl an Benchmark-Läufen wurde durchgeführt.

Innerhalb des Analyseteils stellte sich heraus, dass die Verwendung von TDMA in Multicore-Systemen zumindest im Hinblick auf die Gesamtauslastung des Systems (durchschnittlich nur ca. 20%) und die ACET (durchschnittlich um Faktor 5 höher als bei den anderen untersuchten Verfahren) der jeweiligen Benchmarks problematisch ist. Diese Ergebnisse wurden aufgrund von teilweise extrem voneinander abweichenden Ausführungszeiten der einzelnen Singlecore-Anwendungen und dem daraus resultierenden *Jitter* bei der Ausführung auf der Multicore-Plattform verstärkt.

Des Weiteren zeigte sich, dass das Priorisierungsverfahren der *inversen Busauslastung* leicht bessere Ergebnisse bezüglich Gesamtauslastung des gemeinsamen Busses und Reduktion des Jitters bei der Verwendung von PA bzw. PD lieferte, in Bezug auf die durchschnittliche ACET der Singlecore-Anwendungen in den hier durchgeführten Messungen allerdings minimal hinter AAU zurückliegt.

In Bezug auf die durchschnittliche Wartezeit der Instruktionen beim Zugriff auf den gemeinsamen Bus wurden bei der Verwendung von TDMA erneut die schlechtesten Ergebnisse erzielt, wohingegen FA und PD sehr ähnliche Ergebnisse lieferten. PA erzielte in dieser Disziplin unabhängig vom verwendeten Priorisierungsverfahren die besten Ergebnisse.

Im Hinblick auf zukünftige Arbeiten in diesem Bereich wären Untersuchungen dahingehend denkbar, den Einfluss von Arbitrierungsverfahren auf die Auslastung des Systems unter der Verwendung von voneinander abhängigen Multitask-Anwendungen zu untersuchen. Des Weiteren wären komplexere Szenarien mit PD als Arbitrierungsverfahren interessant, in der die in der Bridge vorbereiteten Slot Modes variiert und eine Maximierung der Auslastung bzw. Minimierung der ACET bei den durchgeführten Benchmarks erreicht werden. Außerdem wäre eine Betrachtung der WCET sowie ein Vergleich zwischen ACET und WCET der einzelnen Benchmarks bei der Verwendung der unterschiedlichen Arbitrierungsverfahren bzw. unterschiedlichen Parametrisierungen interessant.
Zur Verbesserung der Bridge wäre die in dieser Arbeit nicht näher betrachtete Modellierung von Split Transactions eine mögliche Erweiterung, die die parallele Verwendung von mehreren unabhängigen Speichern oder anderen Peripheriegeräten hinter dem Bus ermöglicht, was die Flexibilität und Einsetzbarkeit der Bridge weiter erhöhen würde.

Anhang A

Weitere Informationen

A.1 Messergebnisse der Singlecore-Benchmarks (Referenz)

Innerhalb dieses Abschnittes finden sich die detaillierten Messergebnisse der Singlecore-Benchmarks. Die Ergebnisse sind nach Benchmark-Name alphabetisch sortiert und vergleichen die ACET, die Busy-Tickstamps, die Delay-Tickstamps und die Auslastung des gemeinsamen Busses miteinander.
Im Spezialfall der Ausführung auf der X1-Plattform sind die Ergebnisse der Arbitrierung mit FA/PA bzw. TDMA/PD identisch, aus diesem Grund wurden die Ergebnisse in jeweils einer Spalte konsolisiert.

	FA/PA				TDMA/PD			
Benchmark	ACET	Busy	Delay	Util.	ACET	Busy	Delay	Util.
adpcm	2.639.629	172.269	172.269	0,065	2.708.977	172.269	268.916	0,064
adpcm_decoder	887.330	42.937	42.937	0,048	920.671	42.937	77.076	0,047
adpcm_encoder	888.472	43.229	43.229	0,049	921.935	43.229	77.492	0,047
adpcm_g721_board_test	240.123	48.899	48.899	0,204	254.612	48.899	64.474	0,192
adpcm_g721_verify	240.646	49.093	49.093	0,204	255.260	49.093	64.793	0,192
ammunition	166.194.097	28.501.535	28.501.535	0,171	185.712.214	28.501.535	53.139.654	0,153
audiobeam	2.238.082	308.082	308.082	0,138	2.340.520	308.082	428.483	0,132
basicmath_small	7.440.343	299.211	299.211	0,040	7.552.321	299.211	455.714	0,040
binarysearch	325	103	103	0,317	354	103	138	0,291
bitcount	17.611	3.265	3.265	0,185	19.413	3.265	5.107	0,168
bitonic	39.798	9.263	9.263	0,233	44.159	9.263	14.011	0,210
bsort100	201.037	74.590	74.590	0,371	203.072	74.590	76.772	0,367
cjpeg_jpeg6b_transupp	4.477.085	1.591.204	1.591.204	0,355	4.931.962	1.591.204	2.051.661	0,323
cjpeg_jpeg6b_wrbmp	429.730	141.421	141.421	0,329	461.505	141.421	183.442	0,306
codecs_codrle1	114.089	30.633	30.633	0,269	125.380	30.633	45.389	0,244
codecs_dcodhuff	466.779	127.186	127.186	0,272	509.798	127.186	177.373	0,249
codecs_dcodrle1	63.417	17.096	17.096	0,270	66.458	17.096	20.237	0,257
complex_multiply_fixed	325	103	103	0,317	368	103	152	0,280
complex_multiply_float	578	123	123	0,213	629	123	180	0,196
complex_update_fixed	289	92	92	0,318	336	92	142	0,274
complex_update_float	688	132	132	0,192	734	132	178	0,180
compress	34.582.425	3.186.762	3.186.762	0,092	35.294.453	3.186.762	3.937.515	0,090
compressdata	2.276	617	617	0,271	2.418	617	867	0,255
convolution_fixed	413	122	122	0,295	448	122	162	0,272
convolution_float	3.014	249	249	0,083	3.078	249	353	0,081
countnegative	43.579	10.090	10.090	0,232	48.585	10.090	16.096	0,208
cover	5.616	43	43	0,008	5.633	43	62	0,008
crc	52.944	8.234	8.234	0,156	55.437	8.234	10.832	0,149
dijkstra	234.095.000	89.756.526	89.756.526	0,383	253.587.310	89.756.526	109.293.881	0,354
dot_product_fixed	255	81	81	0,318	294	81	123	0,276
dot_product_float	461	115	115	0,249	497	115	154	0,231

continued ...

	FA/PA				TDMA/PD			
Benchmark	ACET	Busy	Delay	Util.	ACET	Busy	Delay	Util.
duff	1.421	593	593	0,417	1.551	593	723	0,382
edge_detect	860.632	184.893	184.893	0,215	878.183	184.893	210.544	0,211
edn	88.693	13.230	13.230	0,149	105.612	13.230	30.516	0,125
epic	56.892.596	4.657.746	4.657.746	0,082	58.946.954	4.657.746	6.734.890	0,079
expint	6.405	39	39	0,006	6.432	39	66	0,006
fac	1.110	215	215	0,194	1.203	215	324	0,179
fft_1024_13	2.680.172	192.493	192.493	0,072	2.793.375	192.493	353.959	0,069
fft_1024	2.907.585	283.771	283.771	0,098	3.000.417	283.771	385.088	0,095
fft_1024_7	1.984.412	147.437	147.437	0,074	2.071.000	147.437	254.213	0,071
fft_16_13	36.884	2.103	2.103	0,057	37.782	2.103	3.531	0,056
fft_16_7	28.550	1.783	1.783	0,062	29.459	1.783	3.038	0,061
fft1	93.383	4.894	4.894	0,052	95.235	4.894	7.120	0,051
fft_256	581.630	58.729	58.729	0,101	600.475	58.729	79.418	0,098
fibcall	359	20	20	0,056	359	20	20	0,056
filterbank	24.265.251	1.144.933	1.144.933	0,047	24.794.792	1.144.933	2.412.491	0,046
fir_256_64	26.962	1.580	1.580	0,059	27.227	1.580	1.845	0,058
fir2dim_fixed	6.496	2.495	2.495	0,384	7.143	2.495	3.182	0,349
fir2dim_float	26.667	2.803	2.803	0,105	27.223	2.803	3.744	0,103
fir_32_1	3.306	234	234	0,071	3.357	234	285	0,070
fir	5.764	1.423	1.423	0,247	6.010	1.423	1.693	0,237
fir_fixed	1.102	430	430	0,390	1.236	430	567	0,348
fir_float	4.772	482	482	0,101	4.893	482	705	0,099
fmref	4.353.232	240.276	240.276	0,055	4.466.908	240.276	372.300	0,054
g721_encode	1.174.135	199.712	199.712	0,170	1.284.210	199.712	326.975	0,156
g721.marcuslee_decoder	137.301	43.350	43.350	0,316	144.536	43.350	50.585	0,300
g721.marcuslee_encoder	257.035	43.356	43.356	0,169	267.487	43.356	53.808	0,162
g723_encode	1.184.903	204.129	204.129	0,172	1.295.825	204.129	332.247	0,158
gsm	9.468.301	2.549.644	2.549.644	0,269	11.111.750	2.549.644	4.228.307	0,229
gsm_decode	2.951.840	355.790	355.790	0,121	3.069.864	355.790	499.483	0,116
gsm_encode	9.466.694	2.548.287	2.548.287	0,269	11.108.108	2.548.287	4.226.651	0,229
h263	4.985.308	1.422.639	1.422.639	0,285	5.641.139	1.422.639	2.120.144	0,252

continued ...

	FA/PA				TDMA/PD			
Benchmark	ACET	Busy	Delay	Util.	ACET	Busy	Delay	Util.
h264dec_ldecode_block	62.173	17.788	17.788	0,286	66.822	17.788	23.083	0,266
hamming_window	38.184	13.831	13.831	0,362	43.727	13.831	20.133	0,316
histogram	340.861	127.507	127.507	0,374	371.191	127.507	163.984	0,344
iir_1_1	520	76	76	0,146	548	76	109	0,139
iir_4_64	113.589	10.618	10.618	0,093	117.696	10.618	15.114	0,090
iir_biquad_N_sections_fixed	1.028	312	312	0,304	1.143	312	469	0,273
iir_biquad_N_sections_float	1.760	273	273	0,155	1.853	273	430	0,147
iir_biquad_one_section_fixed	337	103	103	0,306	378	103	159	0,272
iir_biquad_one_section_float	708	108	108	0,153	764	108	169	0,141
insertsort	3.061	1.212	1.212	0,396	3.167	1.212	1.320	0,383
janne_complex	310	20	20	0,065	320	20	30	0,063
jfdctint	5.558	1.245	1.245	0,224	5.851	1.245	1.655	0,213
jpeg	257.611	65.474	65.474	0,254	274.699	65.474	90.188	0,238
latnrm_32_64	799.930	51.251	51.251	0,064	814.426	51.251	71.518	0,063
lcdnum	532	140	140	0,263	565	140	173	0,248
lms	3.235.656	202.674	202.674	0,063	3.292.428	202.674	276.374	0,062
lmsfir_32_64	553.685	43.640	43.640	0,079	563.074	43.640	54.737	0,078
lmsfir_8_1	1.912	179	179	0,094	1.958	179	242	0,091
lms_fixed	1.514	566	566	0,374	1.718	566	779	0,329
lms_float	3.912	572	572	0,146	4.052	572	731	0,141
lpc	2.096.632	128.425	128.425	0,061	2.125.761	128.425	162.124	0,060
ludcmp	6.887	1.890	1.890	0,274	7.641	1.890	2.751	0,247
matmult	342.269	112.495	112.495	0,329	368.497	112.495	151.938	0,305
matrix1_fixed	36.342	14.481	14.481	0,398	41.674	14.481	20.232	0,347
matrix1_float	105.074	14.489	14.489	0,138	109.721	14.489	20.073	0,132
matrix1x3_fixed	392	149	149	0,380	433	149	191	0,344
matrix1x3_float	1.544	220	220	0,142	1.596	220	285	0,138
matrix2_fixed	34.915	13.881	13.881	0,398	40.213	13.881	19.543	0,345
matrix2_float	101.047	13.889	13.889	0,137	105.343	13.889	19.082	0,132
md5	17.507.802	3.844.528	3.844.528	0,220	19.659.494	3.844.528	6.973.578	0,196
minver	9.290	1.306	1.306	0,141	9.610	1.306	1.672	0,136

continued . . .

	FA/PA				TDMA/PD			
Benchmark	ACET	Busy	Delay	Util.	ACET	Busy	Delay	Util.
mpeg2	452.309.869	138.678.969	138.678.969	0,307	510.558.866	138.678.969	198.234.881	0,272
mult_10_10	108.647	6.343	6.343	0,058	110.251	6.343	8.144	0,058
n_complex_updates_fixed	3.395	1.359	1.359	0,400	3.917	1.359	1.896	0,347
n_complex_updates_float	8.122	1.365	1.365	0,168	8.430	1.365	1.673	0,162
ndes	136.363	42.201	42.201	0,309	153.858	42.201	62.869	0,274
n_real_updates_fixed	1.608	639	639	0,397	1.833	639	868	0,349
n_real_updates_float	2.822	643	643	0,228	3.054	643	881	0,211
petrinet	4.573	1.661	1.661	0,363	5.037	1.661	2.125	0,330
pm	66.639.593	5.676.645	5.676.645	0,085	68.008.685	5.676.645	7.398.594	0,083
prime	63.496	3.088	3.088	0,049	64.932	3.088	4.851	0,048
qmf_receive	2.577.575	914.268	914.268	0,355	2.790.090	914.268	1.133.538	0,328
qmf_transmit	2.643.336	950.774	950.774	0,360	2.865.088	950.774	1.182.531	0,332
qsort-exam	5.010	1.276	1.276	0,255	5.231	1.276	1.709	0,244
qurt	8.843	478	478	0,054	9.033	478	708	0,053
real_update_fixed	205	65	65	0,317	233	65	93	0,279
real_update_float	397	109	109	0,275	428	109	141	0,255
rijndael_decoder	8.200.503	2.806.654	2.806.654	0,342	9.466.339	2.806.654	4.087.047	0,296
rijndael_encoder	8.141.439	2.762.902	2.762.902	0,339	9.459.906	2.762.902	4.114.915	0,292
searchmultiarray	21.186	3.778	3.778	0,178	22.715	3.778	5.307	0,166
select	4.002	607	607	0,152	4.403	607	1.083	0,138
selection_sort	999.359	274.502	274.502	0,275	1.089.662	274.502	364.805	0,252
sha	3.298.164	596.107	596.107	0,181	3.482.700	596.107	860.014	0,171
spectral	2.251.130	230.499	230.499	0,102	2.345.344	230.499	333.507	0,098
sqrt	27.096	706	706	0,026	27.462	706	1.180	0,026
st	1.782.050	104.012	104.012	0,058	1.834.326	104.012	174.588	0,057
startup_fixed	3.050	798	798	0,262	3.276	798	1.065	0,244
statemate	268.200	114.453	114.453	0,427	280.778	114.453	127.331	0,408
v32.modem_achop	22.627	8.175	8.175	0,361	24.068	8.175	9.616	0,340
v32.modem_cnoise	143.944	45.003	45.003	0,313	154.136	45.003	56.740	0,292
v32.modem_ddecode	7.899.364	1.233.118	1.233.118	0,156	8.215.002	1.233.118	1.666.452	0,150
v32.modem_eglue	26.127	9.577	9.577	0,367	28.766	9.577	12.933	0,333

A.2 Messergebnisse der Multicore-Benchmarks

Innerhalb dieses Abschnittes finden sich die detaillierten Messergebnisse der Multicore-Benchmarks. Die Ergebnisse sind nach nach Benchmark-Name alphabetisch sortiert und vergleichen die durchscnittliche ACET und Busauslastung der einzelen Singlecore-Anwendungen bei Ausführung auf der Multicore-Plattform miteinander. Die Ergebnisse sind dabei pro Ausführungsplattform aufgeführt, um die unterschiedlichen Ausführungszeiten vergleichen zu können.
Innerhalb der durchgeführten Simulationsläufe kam es bei der Durchführung der Multicore-Benchmarks, die den *mpeg2*-Benchmark als Core-Anwendung enthielten, auf der X4- bzw. X8-Plattform unabhängig vom verwendeten Arbitrierungsverfahren zu Timeouts bei der Simulation. Die Ergebnisse dieses Benchmarks sind aus diesem Grund innerhalb der Tabelle genullt.

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
adpcm_decoder	x1	887.330	0,048	920.671	0,047	887.330	0,048	887.330	0,048	920.671	0,047	920.671	0,047
	x2	894.480	0,048	1.116.186	0,038	892.385	0,048	892.385	0,048	928.424	0,046	928.424	0,046
	x4	920.661	0,047	1.456.691	0,029	904.121	0,047	906.598	0,047	944.349	0,045	947.628	0,045
	x8	1.015.159	0,042	1.981.886	0,022	913.809	0,047	966.816	0,045	959.623	0,045	1.034.399	0,042
adpcm_encoder	x1	888.472	0,049	921.935	0,047	888.472	0,049	888.472	0,049	921.935	0,047	921.935	0,047
	x2	901.304	0,048	1.118.163	0,039	895.201	0,048	895.201	0,048	930.681	0,046	930.681	0,046
	x4	925.469	0,047	1.460.627	0,030	907.099	0,048	907.782	0,048	950.079	0,046	951.141	0,045
	x8	1.010.746	0,043	1.989.743	0,022	915.467	0,047	943.408	0,046	962.886	0,045	1.012.674	0,043
adpcm_g721_board_test	x1	240.123	0,204	254.612	0,192	240.123	0,204	240.123	0,204	254.612	0,192	254.612	0,192
	x2	246.557	0,198	397.639	0,123	244.559	0,200	246.670	0,198	266.616	0,183	266.563	0,183
	x4	278.219	0,176	735.485	0,066	258.121	0,189	278.828	0,177	296.044	0,165	321.214	0,153
	x8	420.304	0,118	1.377.731	0,035	265.116	0,184	354.123	0,143	317.232	0,154	435.452	0,116
adpcm_g721_verify	x1	240.646	0,204	255.260	0,192	240.646	0,204	240.646	0,204	255.260	0,192	255.260	0,192
	x2	245.566	0,200	398.924	0,123	246.720	0,199	245.101	0,200	265.290	0,185	265.339	0,185
	x4	279.177	0,176	738.079	0,067	267.990	0,183	274.692	0,180	309.267	0,159	318.445	0,155
	x8	434.080	0,114	1.382.865	0,036	277.268	0,177	358.946	0,143	352.899	0,140	447.305	0,115
adpcm	x1	2.639.629	0,065	2.708.977	0,064	2.639.629	0,065	2.639.629	0,065	2.708.977	0,064	2.708.977	0,064
	x2	2.669.836	0,065	3.265.049	0,053	2.661.730	0,065	2.661.730	0,065	2.762.323	0,062	2.762.323	0,062
	x4	2.749.628	0,063	4.683.764	0,037	2.682.469	0,064	2.695.782	0,064	2.801.156	0,062	2.821.727	0,061
	x8	2.967.952	0,058	7.926.709	0,022	2.701.309	0,064	2.757.216	0,063	2.827.633	0,061	2.948.029	0,059
ammunition	x1	166.194.097	0,171	185.712.214	0,153	166.194.097	0,171	166.194.097	0,171	185.712.214	0,153	185.712.214	0,153
	x2	173.896.789	0,164	326.399.049	0,087	171.480.380	0,166	171.480.380	0,166	194.818.192	0,147	194.818.192	0,147
	x4	117.791.859	0,108	435.184.740	0,029	116.462.094	0,109	116.452.898	0,109	131.514.733	0,096	131.491.498	0,096
	x8	120.803.235	0,105	870.355.619	0,015	117.488.848	0,108	121.334.061	0,104	133.426.273	0,095	137.849.435	0,092
audiobeam	x1	2.238.082	0,138	2.340.520	0,132	2.238.082	0,138	2.238.082	0,138	2.340.520	0,132	2.340.520	0,132
	x2	2.248.868	0,137	3.168.240	0,097	2.252.741	0,137	2.252.741	0,137	2.364.505	0,130	2.364.505	0,130
	x4	2.301.644	0,134	4.892.942	0,063	2.307.748	0,134	2.303.284	0,134	2.475.480	0,125	2.468.017	0,125
	x8	2.602.510	0,119	8.856.148	0,035	2.450.858	0,126	2.470.136	0,125	2.780.492	0,111	2.781.482	0,111
basicmath_small	x1	7.440.343	0,040	7.552.321	0,040	7.440.343	0,040	7.440.343	0,040	7.552.321	0,040	7.552.321	0,040
	x2	7.471.059	0,040	8.953.479	0,033	7.460.525	0,040	7.470.928	0,040	7.607.323	0,039	7.607.313	0,039
	x4	7.631.517	0,039	12.077.893	0,025	7.509.903	0,040	7.600.872	0,039	7.728.459	0,039	7.862.229	0,038
	x8	8.310.493	0,036	18.995.987	0,016	7.530.918	0,040	7.977.637	0,038	7.770.684	0,039	8.610.713	0,035

continued . . .

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
binarysearch	x1	325	0,317	354	0,291	325	0,317	325	0,317	354	0,291	354	0,291
	x2	329	0,313	670	0,154	330	0,312	330	0,312	361	0,286	361	0,286
	x4	366	0,281	1.335	0,077	354	0,291	364	0,283	435	0,238	422	0,245
	x8	626	0,165	2.683	0,038	496	0,218	403	0,259	671	0,164	528	0,200
bitcount	x1	17.611	0,185	19.413	0,168	17.611	0,185	17.611	0,185	19.413	0,168	19.413	0,168
	x2	18.123	0,180	29.345	0,111	18.155	0,180	17.988	0,182	19.854	0,164	19.892	0,164
	x4	20.359	0,162	53.322	0,061	19.634	0,167	19.089	0,171	22.569	0,145	21.861	0,150
	x8	26.366	0,125	103.425	0,032	22.735	0,144	21.650	0,151	28.091	0,117	25.824	0,127
bitonic	x1	39.798	0,233	44.159	0,210	39.798	0,233	39.798	0,233	44.159	0,210	44.159	0,210
	x2	42.275	0,219	71.321	0,130	42.162	0,220	41.610	0,223	48.327	0,192	48.305	0,192
	x4	50.063	0,185	142.460	0,065	48.274	0,192	44.713	0,207	58.898	0,157	54.470	0,170
	x8	78.303	0,120	284.788	0,033	55.278	0,168	50.827	0,184	73.267	0,127	65.116	0,144
bsort100	x1	201.037	0,371	203.072	0,367	201.037	0,371	201.037	0,371	203.072	0,367	203.072	0,367
	x2	205.470	0,363	398.323	0,187	207.366	0,360	205.780	0,362	230.454	0,324	230.427	0,324
	x4	237.480	0,314	796.634	0,094	269.131	0,278	240.745	0,313	318.786	0,234	283.825	0,265
	x8	373.473	0,204	1.593.278	0,047	431.697	0,175	298.433	0,258	526.437	0,144	373.923	0,205
cjpeg_jpeg6b_transupp	x1	4.477.085	0,355	4.931.962	0,323	4.477.085	0,355	4.477.085	0,355	4.931.962	0,323	4.931.962	0,323
	x2	4.557.258	0,349	8.850.709	0,180	4.607.776	0,345	4.537.966	0,351	5.090.591	0,313	5.090.602	0,313
	x4	4.687.929	0,340	17.594.779	0,090	4.862.401	0,328	4.658.758	0,342	5.537.981	0,288	5.274.579	0,302
	x8	5.818.778	0,276	35.091.354	0,045	8.343.018	0,215	5.091.325	0,314	10.143.630	0,176	6.053.683	0,264
cjpeg_jpeg6b_wrbmp	x1	429.730	0,329	461.505	0,306	429.730	0,329	429.730	0,329	461.505	0,306	461.505	0,306
	x2	455.541	0,310	799.845	0,177	459.146	0,308	451.192	0,314	505.691	0,280	505.828	0,280
	x4	554.845	0,257	1.599.418	0,088	548.949	0,258	551.225	0,263	673.441	0,211	663.799	0,218
	x8	801.553	0,179	3.198.559	0,044	785.727	0,181	714.668	0,208	892.891	0,122	914.347	0,162
codecs_codrle1	x1	114.089	0,269	125.380	0,244	114.089	0,269	114.089	0,269	125.380	0,244	125.380	0,244
	x2	118.098	0,259	248.310	0,123	117.346	0,261	117.346	0,261	134.351	0,228	134.351	0,228
	x4	139.585	0,222	496.603	0,062	126.968	0,241	126.827	0,242	153.800	0,200	153.778	0,200
	x8	227.633	0,147	993.250	0,031	157.591	0,195	145.819	0,211	214.294	0,147	187.245	0,164
codecs_dcodhuff	x1	466.779	0,272	509.798	0,249	466.779	0,272	466.779	0,272	509.798	0,249	509.798	0,249
	x2	478.343	0,266	929.797	0,137	475.869	0,267	480.156	0,265	532.769	0,239	532.692	0,239
	x4	531.563	0,240	1.859.567	0,068	506.061	0,251	556.733	0,233	598.305	0,213	650.575	0,200
	x8	766.825	0,168	3.719.181	0,034	609.265	0,209	751.600	0,180	708.240	0,138	930.224	0,145

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
codecs_dcodrle1	x1	63.417	0,270	66.458	0,257	63.417	0,270	63.417	0,270	66.458	0,257	66.458	0,257
	x2	65.695	0,261	130.520	0,131	65.000	0,263	65.981	0,259	74.055	0,231	74.058	0,231
	x4	70.940	0,242	261.027	0,065	70.805	0,242	72.815	0,237	83.636	0,205	84.231	0,205
	x8	96.965	0,178	522.054	0,033	96.057	0,178	94.601	0,186	123.038	0,139	117.195	0,150
complex_multiply_fixed	x1	325	0,317	368	0,280	325	0,317	325	0,317	368	0,280	368	0,280
	x2	349	0,295	677	0,152	348	0,296	344	0,300	400	0,258	404	0,255
	x4	394	0,262	1.337	0,077	397	0,260	382	0,270	492	0,210	488	0,212
	x8	738	0,140	2.720	0,038	660	0,165	473	0,225	873	0,121	644	0,166
complex_multiply_float	x1	578	0,213	629	0,196	578	0,213	578	0,213	629	0,196	629	0,196
	x2	593	0,208	987	0,125	592	0,208	592	0,208	648	0,190	654	0,188
	x4	660	0,187	1.725	0,071	649	0,190	646	0,191	761	0,162	747	0,165
	x8	1.103	0,113	3.261	0,038	737	0,168	765	0,167	945	0,131	1.035	0,124
complex_update_fixed	x1	289	0,318	336	0,274	289	0,318	289	0,318	336	0,274	336	0,274
	x2	297	0,310	644	0,143	299	0,308	299	0,308	344	0,267	344	0,267
	x4	344	0,268	1.270	0,072	364	0,253	341	0,270	452	0,204	424	0,217
	x8	636	0,145	2.548	0,036	551	0,187	374	0,248	791	0,118	457	0,204
complex_update_float	x1	688	0,192	734	0,180	688	0,192	688	0,192	734	0,180	734	0,180
	x2	701	0,188	1.118	0,118	699	0,189	701	0,188	763	0,173	755	0,175
	x4	778	0,170	1.906	0,069	732	0,180	772	0,171	843	0,157	876	0,151
	x8	1.173	0,114	3.340	0,040	742	0,178	899	0,149	909	0,146	1.196	0,114
compressdata	x1	2.276	0,271	2.418	0,255	2.276	0,271	2.276	0,271	2.418	0,255	2.418	0,255
	x2	2.288	0,270	3.807	0,162	2.302	0,268	2.298	0,268	2.460	0,251	2.465	0,250
	x4	2.392	0,258	7.596	0,081	2.440	0,253	2.391	0,258	2.742	0,225	2.609	0,237
	x8	3.310	0,187	15.198	0,041	3.046	0,204	2.837	0,222	4.004	0,156	3.597	0,180
compress	x1	34.582.425	0,092	35.294.453	0,090	34.582.425	0,092	34.582.425	0,092	35.294.453	0,090	35.294.453	0,090
	x2	34.632.175	0,092	42.080.921	0,076	34.668.040	0,092	34.625.378	0,092	35.427.283	0,090	35.427.283	0,090
	x4	34.812.908	0,092	59.527.825	0,054	34.847.703	0,091	34.840.548	0,091	35.977.046	0,089	35.941.730	0,089
	x8	30.295.860	0,073	79.560.118	0,028	29.297.032	0,076	33.424.645	0,067	30.832.780	0,072	34.957.274	0,064
convolution_fixed	x1	413	0,295	448	0,272	413	0,295	413	0,295	448	0,272	448	0,272
	x2	424	0,288	722	0,169	423	0,288	422	0,289	465	0,263	471	0,259
	x4	489	0,250	1.343	0,091	500	0,244	467	0,262	594	0,206	545	0,225
	x8	768	0,159	2.537	0,048	719	0,179	591	0,218	1.026	0,128	767	0,169

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
convolution_float	x1	3.014	0,083	3.078	0,081	3.014	0,083	3.014	0,083	3.078	0,081	3.078	0,081
	x2	3.033	0,082	3.628	0,069	3.030	0,082	3.030	0,082	3.125	0,080	3.125	0,080
	x4	3.125	0,080	4.688	0,053	3.070	0,081	3.079	0,081	3.196	0,078	3.185	0,078
	x8	3.573	0,070	6.704	0,037	3.091	0,081	3.389	0,075	3.236	0,077	3.688	0,069
countnegative	x1	43.579	0,232	48.585	0,208	43.579	0,232	43.579	0,232	48.585	0,208	48.585	0,208
	x2	44.260	0,228	65.484	0,154	44.246	0,228	44.385	0,227	48.972	0,206	49.015	0,206
	x4	49.767	0,203	129.428	0,078	47.392	0,213	49.617	0,204	54.821	0,184	57.708	0,175
	x8	73.549	0,139	258.851	0,039	51.098	0,198	68.925	0,157	64.133	0,158	84.284	0,128
cover	x1	5.616	0,008	5.633	0,008	5.616	0,008	5.616	0,008	5.633	0,008	5.633	0,008
	x2	5.629	0,008	5.758	0,007	5.625	0,008	5.625	0,008	5.655	0,008	5.655	0,008
	x4	5.658	0,008	6.053	0,007	5.633	0,008	5.631	0,008	5.684	0,008	5.673	0,008
	x8	5.948	0,007	6.718	0,006	5.646	0,008	5.708	0,008	5.699	0,008	5.784	0,007
crc	x1	52.944	0,156	55.437	0,149	52.944	0,156	52.944	0,156	55.437	0,149	55.437	0,149
	x2	54.200	0,152	79.425	0,104	53.828	0,153	54.116	0,152	56.802	0,145	56.791	0,145
	x4	57.885	0,142	137.191	0,060	55.270	0,149	58.041	0,142	60.597	0,136	63.891	0,129
	x8	73.901	0,112	233.429	0,035	57.934	0,142	66.791	0,126	65.505	0,126	85.626	0,101
dijkstra	x1	234.095.000	0,383	253.587.310	0,354	234.095.000	0,383	234.095.000	0,383	253.587.310	0,354	253.587.310	0,354
	x2	239.594.588	0,375	505.948.640	0,177	242.896.537	0,370	238.883.852	0,376	274.629.757	0,327	274.629.693	0,327
	x4	118.840.649	0,189	505.948.640	0,044	120.668.197	0,186	120.668.197	0,186	137.089.931	0,164	137.089.931	0,164
	x8	121.066.996	0,185	1.011.897.280	0,022	125.085.067	0,179	124.867.779	0,180	143.836.197	0,156	143.504.567	0,156
dot_product_fixed	x1	255	0,318	294	0,276	255	0,318	255	0,318	294	0,276	294	0,276
	x2	268	0,303	541	0,150	268	0,302	268	0,302	320	0,253	320	0,253
	x4	300	0,270	1.088	0,075	297	0,273	287	0,283	390	0,208	363	0,223
	x8	593	0,137	2.089	0,039	573	0,142	293	0,277	684	0,119	386	0,210
dot_product_float	x1	461	0,249	497	0,231	461	0,249	461	0,249	497	0,231	497	0,231
	x2	472	0,244	833	0,138	469	0,245	470	0,245	509	0,226	515	0,223
	x4	539	0,214	1.499	0,077	508	0,227	565	0,210	606	0,190	665	0,178
	x8	864	0,133	3.018	0,038	611	0,192	722	0,170	796	0,150	942	0,130
duff	x1	1.421	0,417	1.551	0,382	1.421	0,417	1.421	0,417	1.551	0,382	1.551	0,382
	x2	1.460	0,406	2.787	0,213	1.495	0,397	1.447	0,410	1.622	0,366	1.626	0,365
	x4	1.783	0,334	5.596	0,106	2.030	0,294	1.724	0,345	2.420	0,247	2.082	0,286
	x8	2.885	0,209	11.134	0,053	2.665	0,231	2.102	0,288	3.939	0,154	2.876	0,211

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
edge_detect	x1	860.632	0,215	878.183	0,211	860.632	0,215	860.632	0,215	878.183	0,211	878.183	0,211
	x2	862.152	0,214	1.313.320	0,141	863.562	0,214	863.562	0,214	887.870	0,208	887.870	0,208
	x4	869.432	0,213	2.410.382	0,077	874.600	0,211	874.474	0,211	926.687	0,200	926.654	0,200
	x8	959.797	0,193	4.634.777	0,040	948.931	0,195	1.011.085	0,186	1.093.907	0,170	1.153.718	0,164
edn	x1	88.693	0,149	105.612	0,125	88.693	0,149	88.693	0,149	105.612	0,125	105.612	0,125
	x2	89.071	0,149	207.239	0,064	89.186	0,148	89.186	0,148	105.769	0,125	105.769	0,125
	x4	94.839	0,140	414.392	0,032	94.314	0,140	95.450	0,139	110.386	0,120	110.337	0,120
	x8	118.622	0,112	828.622	0,016	109.656	0,121	118.332	0,114	132.039	0,101	140.772	0,096
epic	x1	56.892.596	0,082	58.946.954	0,079	56.892.596	0,082	56.892.596	0,082	58.946.954	0,079	58.946.954	0,079
	x2	57.173.712	0,081	77.499.794	0,060	57.047.308	0,082	57.124.082	0,082	59.351.736	0,078	59.351.709	0,078
	x4	58.122.441	0,080	116.248.112	0,040	57.404.366	0,081	57.604.914	0,081	60.117.277	0,077	60.372.846	0,077
	x8	48.009.550	0,062	141.952.955	0,021	46.109.493	0,065	50.328.690	0,059	48.413.399	0,062	52.916.040	0,056
expint	x1	6.405	0,006	6.432	0,006	6.405	0,006	6.405	0,006	6.432	0,006	6.432	0,006
	x2	6.414	0,006	6.565	0,006	6.410	0,006	6.410	0,006	6.446	0,006	6.446	0,006
	x4	6.444	0,006	6.846	0,006	6.423	0,006	6.424	0,006	6.468	0,006	6.467	0,006
	x8	6.652	0,006	7.514	0,005	6.433	0,006	6.460	0,006	6.476	0,006	6.586	0,006
fac	x1	1.110	0,194	1.203	0,179	1.110	0,194	1.110	0,194	1.203	0,179	1.203	0,179
	x2	1.247	0,173	2.094	0,103	1.187	0,181	1.187	0,181	1.393	0,154	1.393	0,154
	x4	1.703	0,128	4.156	0,052	1.317	0,163	1.289	0,167	1.711	0,126	1.715	0,126
	x8	2.724	0,081	8.287	0,026	1.488	0,145	1.427	0,152	1.972	0,110	1.901	0,114
fft_1024_13	x1	2.680.172	0,072	2.793.375	0,069	2.680.172	0,072	2.680.172	0,072	2.793.375	0,069	2.793.375	0,069
	x2	2.719.697	0,071	3.453.200	0,056	2.708.326	0,071	2.708.326	0,071	2.848.560	0,068	2.848.560	0,068
	x4	2.826.140	0,068	4.983.504	0,039	2.757.724	0,070	2.753.105	0,070	2.944.462	0,065	2.936.999	0,066
	x8	3.299.965	0,058	8.482.872	0,023	2.821.019	0,068	2.823.680	0,068	3.067.480	0,063	3.066.597	0,063
fft_1024_7	x1	1.984.412	0,074	2.071.000	0,071	1.984.412	0,074	1.984.412	0,074	2.071.000	0,071	2.071.000	0,071
	x2	1.993.118	0,074	2.577.281	0,057	1.994.714	0,074	1.991.707	0,074	2.080.595	0,071	2.080.600	0,071
	x4	2.020.125	0,073	3.753.995	0,039	2.013.461	0,073	2.011.385	0,073	2.119.814	0,070	2.118.228	0,070
	x8	2.230.909	0,066	6.351.510	0,023	2.072.942	0,071	2.098.919	0,070	2.237.731	0,066	2.277.275	0,065
fft_1024	x1	2.907.585	0,098	3.000.417	0,095	2.907.585	0,098	2.907.585	0,098	3.000.417	0,095	3.000.417	0,095
	x2	2.922.772	0,097	3.779.909	0,075	2.920.798	0,097	2.920.798	0,097	3.040.067	0,093	3.040.067	0,093
	x4	2.995.565	0,095	5.118.451	0,055	2.971.139	0,096	2.964.953	0,096	3.145.300	0,090	3.135.118	0,091
	x8	3.478.707	0,082	9.998.353	0,028	3.062.773	0,093	3.065.470	0,093	3.349.978	0,085	3.341.668	0,085

continued . . .

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
fft_16_13	x1	36.884	0,057	37.782	0,056	36.884	0,057	36.884	0,057	37.782	0,056	37.782	0,056
	x2	37.716	0,056	44.962	0,047	37.460	0,056	37.460	0,056	39.130	0,054	39.130	0,054
	x4	41.295	0,051	59.898	0,035	38.129	0,055	38.148	0,055	40.836	0,052	40.985	0,051
	x8	49.767	0,042	96.484	0,022	38.067	0,055	38.581	0,055	40.504	0,052	41.595	0,051
fft_16_7	x1	28.550	0,062	29.459	0,061	28.550	0,062	28.550	0,062	29.459	0,061	29.459	0,061
	x2	28.994	0,062	34.999	0,051	28.815	0,062	28.815	0,062	30.044	0,059	30.044	0,059
	x4	30.591	0,058	48.130	0,037	29.424	0,061	29.425	0,061	31.163	0,057	31.143	0,057
	x8	37.584	0,048	78.065	0,023	29.908	0,060	30.243	0,059	32.202	0,055	32.902	0,054
fft1	x1	93.383	0,052	95.235	0,051	93.383	0,052	93.383	0,052	95.235	0,051	95.235	0,051
	x2	93.792	0,052	107.573	0,045	93.583	0,052	93.583	0,052	95.913	0,051	95.913	0,051
	x4	94.963	0,052	145.575	0,034	94.113	0,052	94.303	0,052	97.033	0,050	97.224	0,050
	x8	103.419	0,047	228.326	0,021	95.430	0,051	99.990	0,049	99.779	0,049	107.878	0,046
fft_256	x1	581.630	0,101	600.475	0,098	581.630	0,101	581.630	0,101	600.475	0,098	600.475	0,098
	x2	584.593	0,100	767.556	0,077	585.116	0,100	584.372	0,101	607.184	0,097	607.213	0,097
	x4	602.570	0,097	1.045.814	0,056	596.489	0,098	605.707	0,097	633.874	0,093	645.568	0,091
	x8	725.032	0,081	2.051.486	0,029	616.085	0,095	742.900	0,082	596.981	0,075	872.580	0,071
fibcall	x1	359	0,056	359	0,056	359	0,056	359	0,056	359	0,056	359	0,056
	x2	366	0,055	433	0,046	362	0,055	362	0,055	374	0,053	374	0,053
	x4	375	0,053	619	0,032	365	0,055	364	0,055	384	0,052	383	0,052
	x8	471	0,042	959	0,021	368	0,054	372	0,054	388	0,052	400	0,050
filterbank	x1	24.265.251	0,047	24.794.792	0,046	24.265.251	0,047	24.265.251	0,047	24.794.792	0,046	24.794.792	0,046
	x2	24.317.076	0,047	25.936.828	0,044	24.284.825	0,047	24.317.894	0,047	24.860.275	0,046	24.860.270	0,046
	x4	24.455.675	0,047	31.853.904	0,036	24.327.964	0,047	24.585.825	0,047	24.944.279	0,046	25.321.677	0,045
	x8	22.265.321	0,038	36.368.767	0,023	20.936.968	0,040	25.369.067	0,034	21.592.931	0,039	25.785.485	0,033
fir_256_64	x1	26.962	0,059	27.227	0,058	26.962	0,059	26.962	0,059	27.227	0,058	27.227	0,058
	x2	26.995	0,059	31.035	0,051	26.990	0,059	26.990	0,059	27.316	0,058	27.316	0,058
	x4	27.375	0,058	41.604	0,038	27.203	0,058	27.197	0,058	27.713	0,057	27.728	0,057
	x8	29.883	0,053	42.391	0,037	27.458	0,058	27.631	0,057	28.117	0,056	28.829	0,055
fir2dim_fixed	x1	6.496	0,384	7.143	0,349	6.496	0,384	6.496	0,384	7.143	0,349	7.143	0,349
	x2	6.629	0,377	12.765	0,195	6.629	0,377	6.629	0,377	7.460	0,335	7.460	0,335
	x4	6.971	0,358	25.519	0,098	7.127	0,351	7.003	0,357	8.627	0,291	8.339	0,300
	x8	9.818	0,256	51.034	0,049	11.885	0,213	9.036	0,284	14.988	0,170	11.509	0,221

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
fir2dim_float	x1	26.667	0,105	27.223	0,103	26.667	0,105	26.667	0,105	27.223	0,103	27.223	0,103
	x2	26.788	0,105	32.076	0,087	26.778	0,105	26.778	0,105	27.541	0,102	27.541	0,102
	x4	27.380	0,102	43.438	0,065	27.048	0,104	27.079	0,104	28.072	0,100	28.170	0,100
	x8	30.587	0,092	81.906	0,034	27.471	0,102	27.984	0,100	28.912	0,097	30.249	0,093
fir_32_1	x1	3.306	0,071	3.357	0,070	3.306	0,071	3.306	0,071	3.357	0,070	3.357	0,070
	x2	3.317	0,071	3.983	0,059	3.314	0,071	3.314	0,071	3.425	0,068	3.425	0,068
	x4	3.477	0,067	5.421	0,043	3.383	0,069	3.385	0,069	3.514	0,067	3.515	0,067
	x8	4.095	0,057	6.369	0,037	3.381	0,069	3.501	0,067	3.539	0,066	3.835	0,061
fir_fixed	x1	1.102	0,390	1.236	0,348	1.102	0,390	1.102	0,390	1.236	0,348	1.236	0,348
	x2	1.137	0,379	2.200	0,195	1.145	0,376	1.145	0,376	1.276	0,337	1.276	0,337
	x4	1.346	0,322	4.386	0,098	1.351	0,319	1.316	0,327	1.672	0,259	1.586	0,271
	x8	2.145	0,207	8.785	0,049	1.953	0,225	1.565	0,279	2.621	0,168	2.092	0,208
fir_float	x1	4.772	0,101	4.893	0,099	4.772	0,101	4.772	0,101	4.893	0,099	4.893	0,099
	x2	4.850	0,099	5.842	0,083	4.829	0,100	4.829	0,100	5.002	0,096	5.002	0,096
	x4	5.006	0,096	7.823	0,062	4.847	0,099	4.882	0,099	5.059	0,095	5.127	0,094
	x8	5.895	0,082	12.620	0,038	4.889	0,099	5.652	0,088	5.145	0,094	6.171	0,080
fir	x1	5.764	0,247	6.010	0,237	5.764	0,247	5.764	0,247	6.010	0,237	6.010	0,237
	x2	5.765	0,247	10.362	0,137	5.768	0,247	5.768	0,247	6.003	0,237	6.003	0,237
	x4	6.138	0,232	19.917	0,071	6.084	0,234	6.125	0,233	6.602	0,216	6.683	0,213
	x8	8.050	0,177	39.171	0,036	6.960	0,205	7.722	0,189	8.646	0,165	9.242	0,158
fmref	x1	4.353.232	0,055	4.466.908	0,054	4.353.232	0,055	4.353.232	0,055	4.466.908	0,054	4.466.908	0,054
	x2	4.401.000	0,055	5.223.829	0,046	4.384.200	0,055	4.392.494	0,055	4.525.640	0,053	4.525.601	0,053
	x4	4.511.700	0,053	6.895.265	0,035	4.427.867	0,054	4.460.582	0,054	4.614.376	0,052	4.661.163	0,052
	x8	5.014.302	0,048	10.344.804	0,023	4.456.763	0,054	4.689.987	0,051	4.663.299	0,052	5.056.723	0,048
g721_encode	x1	1.174.135	0,170	1.284.210	0,156	1.174.135	0,170	1.174.135	0,170	1.284.210	0,156	1.284.210	0,156
	x2	1.207.971	0,165	1.988.125	0,100	1.195.904	0,167	1.209.338	0,165	1.329.158	0,150	1.329.147	0,150
	x4	1.249.381	0,160	3.710.036	0,054	1.226.568	0,163	1.242.834	0,161	1.374.743	0,145	1.390.108	0,144
	x8	1.431.453	0,140	7.408.540	0,027	1.352.582	0,148	1.374.670	0,146	1.613.742	0,124	1.619.020	0,124
g721.marcuslee_decoder	x1	137.301	0,316	144.536	0,300	137.301	0,316	137.301	0,316	144.536	0,300	144.536	0,300
	x2	140.473	0,309	192.764	0,225	142.948	0,303	141.622	0,306	153.444	0,283	153.459	0,282
	x4	165.401	0,263	385.519	0,112	172.413	0,252	157.510	0,275	206.841	0,211	188.512	0,230
	x8	243.243	0,184	771.028	0,056	292.399	0,155	178.329	0,246	363.607	0,126	229.453	0,191

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
g721.marcuslee_encoder	x1	257.035	0,169	267.487	0,162	257.035	0,169	257.035	0,169	267.487	0,162	267.487	0,162
	x2	260.327	0,167	362.236	0,120	259.326	0,167	260.543	0,166	273.390	0,159	273.413	0,159
	x4	289.412	0,150	581.500	0,075	266.520	0,163	280.585	0,155	288.804	0,150	308.930	0,141
	x8	399.920	0,109	1.156.299	0,037	268.063	0,162	361.210	0,126	296.781	0,146	429.754	0,106
g723_encode	x1	1.184.903	0,172	1.295.825	0,158	1.184.903	0,172	1.184.903	0,172	1.295.825	0,158	1.295.825	0,158
	x2	1.198.764	0,170	2.013.778	0,101	1.203.014	0,170	1.199.724	0,170	1.313.644	0,155	1.313.656	0,155
	x4	1.246.155	0,164	3.761.336	0,054	1.251.992	0,163	1.258.436	0,162	1.405.924	0,145	1.410.117	0,145
	x8	1.451.475	0,141	7.512.528	0,027	1.459.558	0,140	1.416.627	0,145	1.799.015	0,115	1.686.658	0,122
gsm_decode	x1	2.951.840	0,121	3.069.864	0,116	2.951.840	0,121	2.951.840	0,121	3.069.864	0,116	3.069.864	0,116
	x2	2.962.557	0,120	3.897.098	0,091	2.965.727	0,120	2.964.567	0,120	3.077.935	0,116	3.080.210	0,116
	x4	2.999.674	0,119	5.740.137	0,062	3.003.904	0,118	2.994.106	0,119	3.156.935	0,113	3.143.028	0,113
	x8	3.460.291	0,103	9.489.419	0,037	3.136.050	0,113	3.108.059	0,115	3.466.925	0,103	3.387.028	0,105
gsm_encode	x1	9.466.694	0,269	11.108.108	0,229	9.466.694	0,269	9.466.694	0,269	11.108.108	0,229	11.108.108	0,229
	x2	9.942.378	0,256	20.274.269	0,126	9.762.457	0,261	9.945.988	0,256	11.495.156	0,222	11.495.213	0,222
	x4	11.384.184	0,224	38.951.994	0,065	10.453.000	0,244	11.287.865	0,228	12.800.609	0,199	13.745.963	0,188
	x8	15.451.068	0,166	77.881.952	0,033	12.114.694	0,212	13.604.385	0,193	16.245.420	0,159	17.499.142	0,150
gsm	x1	9.468.301	0,269	11.111.750	0,229	9.468.301	0,269	9.468.301	0,269	11.111.750	0,229	11.111.750	0,229
	x2	9.824.106	0,260	20.286.041	0,126	9.829.478	0,259	9.706.784	0,263	11.313.597	0,225	11.316.204	0,225
	x4	11.020.647	0,233	38.989.114	0,065	10.951.842	0,233	11.200.751	0,232	13.461.765	0,190	13.575.586	0,192
	x8	15.624.833	0,164	77.949.272	0,033	14.545.490	0,178	14.366.962	0,186	20.037.696	0,130	18.478.654	0,144
h263	x1	4.985.308	0,285	5.641.139	0,252	4.985.308	0,285	4.985.308	0,285	5.641.139	0,252	5.641.139	0,252
	x2	5.147.314	0,277	10.093.243	0,141	5.092.166	0,279	5.170.714	0,275	5.828.942	0,244	5.828.945	0,244
	x4	5.479.742	0,260	20.186.460	0,070	5.415.059	0,263	5.490.732	0,259	6.331.342	0,225	6.420.051	0,222
	x8	7.336.321	0,198	40.372.954	0,035	6.724.520	0,216	6.230.692	0,230	8.813.694	0,167	7.793.209	0,184
h264dec_ldecode_block	x1	62.173	0,286	66.822	0,266	62.173	0,286	62.173	0,286	66.822	0,266	66.822	0,266
	x2	62.672	0,284	120.394	0,148	63.320	0,281	62.678	0,284	69.120	0,257	69.146	0,257
	x4	65.529	0,272	240.458	0,074	68.723	0,259	65.979	0,270	77.430	0,230	74.622	0,239
	x8	83.237	0,216	480.580	0,037	92.037	0,194	80.301	0,228	114.706	0,156	102.163	0,182
hamming_window	x1	38.184	0,362	43.727	0,316	38.184	0,362	38.184	0,362	43.727	0,316	43.727	0,316
	x2	38.392	0,360	77.034	0,180	38.530	0,359	38.530	0,359	44.472	0,311	44.472	0,311
	x4	42.778	0,324	154.049	0,090	43.199	0,320	44.403	0,315	50.199	0,276	53.393	0,263
	x8	60.417	0,232	308.146	0,045	65.022	0,230	58.837	0,247	81.948	0,184	74.109	0,196

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
histogram	x1	340.861	0,374	371.191	0,344	340.861	0,374	340.861	0,374	371.191	0,344	371.191	0,344
	x2	345.501	0,369	655.136	0,195	346.506	0,368	349.544	0,365	376.128	0,339	376.056	0,339
	x4	426.406	0,300	1.285.814	0,099	431.054	0,299	439.230	0,294	547.466	0,235	549.065	0,234
	x8	641.255	0,199	2.530.598	0,050	717.037	0,179	576.930	0,230	795.012	0,124	770.539	0,170
iir_1_1	x1	520	0,146	548	0,139	520	0,146	520	0,146	548	0,139	548	0,139
	x2	530	0,143	770	0,099	528	0,144	529	0,144	571	0,133	567	0,134
	x4	592	0,128	1.256	0,061	540	0,141	591	0,130	605	0,126	658	0,117
	x8	837	0,091	2.239	0,034	549	0,138	697	0,114	623	0,122	930	0,087
iir_4_64	x1	113.589	0,093	117.696	0,090	113.589	0,093	113.589	0,093	117.696	0,090	117.696	0,090
	x2	114.547	0,093	151.119	0,070	114.247	0,093	114.247	0,093	119.544	0,089	119.544	0,089
	x4	119.659	0,089	226.367	0,047	116.139	0,091	117.111	0,091	123.883	0,086	125.423	0,085
	x8	146.573	0,074	370.862	0,029	118.033	0,090	124.652	0,085	128.427	0,083	140.872	0,076
iir_biquad_N_sections_fixed	x1	1.028	0,304	1.143	0,273	1.028	0,304	1.028	0,304	1.143	0,273	1.143	0,273
	x2	1.054	0,296	2.244	0,139	1.052	0,297	1.052	0,297	1.201	0,260	1.201	0,260
	x4	1.165	0,269	4.481	0,070	1.133	0,275	1.136	0,275	1.335	0,234	1.347	0,232
	x8	1.890	0,171	8.926	0,035	1.420	0,221	1.314	0,240	1.889	0,165	1.699	0,186
iir_biquad_N_sections_float	x1	1.760	0,155	1.853	0,147	1.760	0,155	1.760	0,155	1.853	0,147	1.853	0,147
	x2	1.779	0,153	2.758	0,099	1.777	0,154	1.773	0,154	1.890	0,144	1.884	0,145
	x4	1.890	0,145	4.783	0,057	1.836	0,149	1.359	0,113	2.055	0,133	1.509	0,102
	x8	2.870	0,095	9.568	0,029	1.899	0,144	2.153	0,132	2.208	0,124	2.590	0,111
iir_biquad_one_section_fixed	x1	337	0,306	378	0,272	337	0,306	337	0,306	378	0,272	378	0,272
	x2	346	0,298	726	0,142	344	0,299	347	0,297	408	0,254	399	0,259
	x4	410	0,252	1.421	0,073	383	0,270	413	0,250	465	0,222	503	0,205
	x8	751	0,138	2.872	0,036	548	0,203	511	0,208	750	0,152	692	0,154
iir_biquad_one_section_float	x1	708	0,153	764	0,141	708	0,153	708	0,153	764	0,141	764	0,141
	x2	728	0,148	1.111	0,097	722	0,150	722	0,150	785	0,138	785	0,138
	x4	817	0,133	1.891	0,057	748	0,144	757	0,143	849	0,127	855	0,126
	x8	1.225	0,089	3.192	0,034	748	0,144	883	0,126	872	0,124	1.038	0,107
insertsort	x1	3.061	0,396	3.167	0,383	3.061	0,396	3.061	0,396	3.167	0,383	3.167	0,383
	x2	3.134	0,387	6.040	0,201	3.180	0,381	3.180	0,381	3.449	0,352	3.449	0,352
	x4	3.534	0,344	12.060	0,100	3.547	0,342	3.526	0,344	4.192	0,290	4.137	0,293
	x8	5.498	0,223	24.144	0,050	5.635	0,217	4.324	0,288	7.130	0,172	5.577	0,223

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
janne_complex	x1	310	0,065	320	0,063	310	0,065	310	0,065	320	0,063	320	0,063
	x2	313	0,064	387	0,052	313	0,064	313	0,064	330	0,061	330	0,061
	x4	330	0,061	526	0,038	318	0,063	319	0,063	333	0,060	339	0,059
	x8	422	0,047	948	0,021	321	0,062	321	0,062	341	0,059	343	0,058
jfdctint	x1	5.558	0,224	5.851	0,213	5.558	0,224	5.558	0,224	5.851	0,213	5.851	0,213
	x2	5.598	0,222	9.317	0,134	5.585	0,223	5.620	0,222	5.971	0,209	5.959	0,209
	x4	5.808	0,214	17.343	0,072	5.746	0,217	5.913	0,211	6.247	0,199	6.506	0,192
	x8	7.494	0,166	32.314	0,039	6.105	0,204	7.301	0,175	6.988	0,178	8.790	0,147
jpeg	x1	257.611	0,254	274.699	0,238	257.611	0,254	257.611	0,254	274.699	0,238	274.699	0,238
	x2	264.933	0,247	493.484	0,133	263.709	0,248	263.709	0,248	291.078	0,225	291.078	0,225
	x4	325.639	0,203	967.012	0,068	293.590	0,223	293.324	0,223	353.582	0,185	353.355	0,185
	x8	555.646	0,119	1.908.369	0,034	322.820	0,204	391.508	0,177	445.288	0,149	507.342	0,136
latnrm_32_64	x1	799.930	0,064	814.426	0,063	799.930	0,064	799.930	0,064	814.426	0,063	814.426	0,063
	x2	804.101	0,064	951.367	0,054	802.173	0,064	802.817	0,064	820.863	0,062	820.868	0,062
	x4	810.772	0,063	1.289.979	0,040	805.792	0,064	809.376	0,063	827.284	0,062	833.174	0,062
	x8	884.801	0,058	1.629.739	0,031	816.054	0,063	891.170	0,058	747.215	0,053	975.736	0,053
lcdnum	x1	532	0,263	565	0,248	532	0,263	532	0,263	565	0,248	565	0,248
	x2	545	0,257	1.075	0,130	551	0,254	543	0,258	591	0,237	601	0,233
	x4	609	0,230	2.144	0,065	640	0,219	601	0,234	758	0,185	701	0,201
	x8	981	0,144	4.300	0,033	864	0,165	716	0,205	1.112	0,127	988	0,150
lmsfir_32_64	x1	553.685	0,079	563.074	0,078	553.685	0,079	553.685	0,079	563.074	0,078	563.074	0,078
	x2	555.209	0,079	619.532	0,070	554.673	0,079	556.170	0,078	566.599	0,077	566.427	0,077
	x4	565.471	0,077	745.259	0,059	559.454	0,078	572.992	0,076	576.115	0,076	593.817	0,074
	x8	631.303	0,069	1.005.328	0,043	566.244	0,077	727.575	0,063	516.584	0,065	760.181	0,059
lmsfir_8_1	x1	1.912	0,094	1.958	0,091	1.912	0,094	1.912	0,094	1.958	0,091	1.958	0,091
	x2	1.930	0,093	2.402	0,075	1.924	0,093	1.932	0,093	2.004	0,089	2.003	0,089
	x4	1.998	0,090	3.476	0,051	1.955	0,092	1.469	0,069	2.055	0,087	1.564	0,064
	x8	2.562	0,070	4.564	0,039	1.981	0,090	2.415	0,077	2.122	0,084	2.714	0,069
lms_fixed	x1	1.514	0,374	1.718	0,329	1.514	0,374	1.514	0,374	1.718	0,329	1.718	0,329
	x2	1.561	0,363	3.039	0,186	1.545	0,366	1.597	0,355	1.803	0,314	1.796	0,315
	x4	1.907	0,298	6.062	0,093	1.749	0,324	1.988	0,287	2.149	0,264	2.433	0,234
	x8	3.129	0,183	12.142	0,047	2.393	0,240	2.645	0,221	3.298	0,175	3.498	0,165

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
lms_float	x1	3.912	0,146	4.052	0,141	3.912	0,146	3.912	0,146	4.052	0,141	4.052	0,141
	x2	3.945	0,145	5.243	0,109	3.940	0,145	3.940	0,145	4.124	0,139	4.124	0,139
	x4	4.170	0,137	8.061	0,071	4.038	0,142	4.049	0,141	4.338	0,132	4.360	0,131
	x8	5.663	0,101	13.601	0,042	4.088	0,140	4.596	0,128	4.527	0,126	5.442	0,109
lms	x1	3.235.656	0,063	3.292.428	0,062	3.235.656	0,063	3.235.656	0,063	3.292.428	0,062	3.292.428	0,062
	x2	3.246.966	0,062	3.730.791	0,054	3.241.480	0,063	3.244.785	0,062	3.322.384	0,061	3.321.833	0,061
	x4	3.280.820	0,062	4.695.090	0,043	3.258.946	0,062	3.273.431	0,062	3.357.025	0,060	3.377.712	0,060
	x8	3.624.972	0,056	6.540.768	0,031	3.301.089	0,061	3.361.025	0,060	3.436.241	0,059	3.570.107	0,057
lpc	x1	2.096.632	0,061	2.125.761	0,060	2.096.632	0,061	2.096.632	0,061	2.125.761	0,060	2.125.761	0,060
	x2	2.100.676	0,061	2.344.920	0,055	2.099.370	0,061	2.100.973	0,061	2.137.786	0,060	2.137.766	0,060
	x4	2.121.611	0,061	3.026.135	0,042	2.109.890	0,061	2.114.662	0,061	2.162.021	0,059	2.169.902	0,059
	x8	2.242.356	0,057	3.829.420	0,034	2.129.675	0,060	2.169.162	0,059	2.202.731	0,058	2.288.884	0,056
ludcmp	x1	6.887	0,274	7.641	0,247	6.887	0,274	6.887	0,274	7.641	0,247	7.641	0,247
	x2	7.014	0,270	13.596	0,139	7.012	0,270	6.989	0,270	7.865	0,240	7.844	0,241
	x4	7.359	0,257	26.624	0,071	7.370	0,256	7.242	0,261	8.482	0,223	8.239	0,230
	x8	9.806	0,194	53.249	0,035	9.834	0,193	8.077	0,236	12.448	0,153	9.579	0,201
matmult	x1	342.269	0,329	368.497	0,305	342.269	0,329	342.269	0,329	368.497	0,305	368.497	0,305
	x2	357.165	0,315	586.392	0,192	355.547	0,317	355.547	0,317	395.627	0,284	395.627	0,284
	x4	418.284	0,270	1.169.732	0,096	381.077	0,295	386.859	0,292	444.009	0,254	456.620	0,249
	x8	612.101	0,186	2.339.506	0,048	487.142	0,232	481.556	0,245	580.434	0,149	600.282	0,197
matrix1_fixed	x1	36.342	0,398	41.674	0,347	36.342	0,398	36.342	0,398	41.674	0,347	41.674	0,347
	x2	37.148	0,390	76.400	0,190	37.398	0,387	37.398	0,387	44.410	0,327	44.410	0,327
	x4	40.491	0,359	152.771	0,095	43.494	0,336	40.702	0,356	53.706	0,274	49.975	0,291
	x8	60.396	0,245	305.562	0,047	77.652	0,195	50.167	0,293	94.997	0,159	66.716	0,220
matrix1_float	x1	105.074	0,138	109.721	0,132	105.074	0,138	105.074	0,138	109.721	0,132	109.721	0,132
	x2	105.409	0,137	140.440	0,103	105.470	0,137	105.470	0,137	110.475	0,131	110.475	0,131
	x4	106.274	0,136	201.672	0,072	106.736	0,136	106.615	0,136	113.153	0,128	113.193	0,128
	x8	120.370	0,121	385.724	0,038	109.711	0,132	116.407	0,125	121.770	0,119	130.374	0,112
matrix1x3_fixed	x1	392	0,380	433	0,344	392	0,380	392	0,380	433	0,344	433	0,344
	x2	396	0,377	798	0,187	402	0,371	402	0,371	451	0,331	451	0,331
	x4	456	0,327	1.599	0,093	498	0,303	462	0,323	604	0,250	583	0,257
	x8	748	0,200	3.187	0,047	642	0,247	659	0,240	1.041	0,145	892	0,169

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
matrix1x3_float	x1	1.544	0,142	1.596	0,138	1.544	0,142	1.544	0,142	1.596	0,138	1.596	0,138
	x2	1.566	0,140	2.060	0,107	1.563	0,141	1.563	0,141	1.627	0,135	1.627	0,135
	x4	1.724	0,128	3.117	0,071	1.600	0,138	1.215	0,102	1.740	0,126	1.318	0,094
	x8	2.262	0,097	5.748	0,038	1.613	0,136	1.771	0,126	1.768	0,125	2.069	0,109
matrix2_fixed	x1	34.915	0,398	40.213	0,345	34.915	0,398	34.915	0,398	40.213	0,345	40.213	0,345
	x2	35.583	0,390	72.788	0,191	35.657	0,389	35.657	0,389	43.445	0,321	43.445	0,321
	x4	38.108	0,366	145.565	0,095	38.029	0,367	37.281	0,373	45.890	0,305	44.215	0,315
	x8	56.159	0,253	291.102	0,048	57.953	0,250	43.588	0,323	74.285	0,196	53.261	0,265
matrix2_float	x1	101.047	0,137	105.343	0,132	101.047	0,137	101.047	0,137	105.343	0,132	105.343	0,132
	x2	101.319	0,137	132.848	0,105	101.267	0,137	101.267	0,137	106.084	0,131	106.084	0,131
	x4	102.065	0,136	193.680	0,072	102.001	0,136	102.144	0,136	107.410	0,129	107.621	0,129
	x8	113.077	0,123	369.755	0,038	103.998	0,134	110.980	0,126	112.157	0,124	122.014	0,115
md5	x1	17.507.802	0,220	19.659.494	0,196	17.507.802	0,220	17.507.802	0,220	19.659.494	0,196	19.659.494	0,196
	x2	17.675.370	0,218	36.840.435	0,104	17.639.200	0,218	17.748.954	0,217	20.053.169	0,192	20.053.704	0,192
	x4	18.616.847	0,207	73.680.866	0,052	18.269.738	0,210	19.078.475	0,202	21.311.222	0,180	22.153.434	0,174
	x8	21.062.327	0,140	128.941.531	0,023	18.133.147	0,163	21.145.091	0,142	23.406.727	0,127	25.956.796	0,115
minver	x1	9.290	0,141	9.610	0,136	9.290	0,141	9.290	0,141	9.610	0,136	9.610	0,136
	x2	9.320	0,140	13.084	0,100	9.312	0,140	9.312	0,140	9.725	0,134	9.725	0,134
	x4	9.658	0,135	20.692	0,063	9.487	0,138	9.573	0,136	10.150	0,129	10.284	0,127
	x8	12.078	0,108	36.956	0,035	9.787	0,133	11.458	0,117	10.812	0,121	13.442	0,100
mpeg2	x1	452.309.869	0,307	510.558.866	0,272	452.309.869	0,307	452.309.869	0,307	510.558.866	0,272	510.558.866	0,272
	x2	467.660.039	0,297	951.781.207	0,146	461.584.868	0,300	473.411.504	0,293	528.202.327	0,263	528.202.150	0,263
	x4	0	0,000	0	0,000	0	0,000	0	0,000	0	0,000	0	0,000
	x8	0	0,000	0	0,000	0	0,000	0	0,000	0	0,000	0	0,000
mult_10_10	x1	108.647	0,058	110.251	0,058	108.647	0,058	108.647	0,058	110.251	0,058	110.251	0,058
	x2	108.837	0,058	120.363	0,053	108.815	0,058	108.815	0,058	110.466	0,057	110.466	0,057
	x4	109.958	0,058	152.703	0,042	109.408	0,058	109.492	0,058	111.495	0,057	111.968	0,057
	x8	117.773	0,054	161.434	0,039	110.020	0,058	112.056	0,057	112.983	0,056	117.941	0,054
n_complex_updates_fixed	x1	3.395	0,400	3.917	0,347	3.395	0,400	3.395	0,400	3.917	0,347	3.917	0,347
	x2	3.449	0,394	7.080	0,192	3.449	0,394	3.449	0,394	3.952	0,344	3.952	0,344
	x4	3.831	0,355	14.156	0,096	4.173	0,327	4.008	0,340	5.017	0,273	4.824	0,283
	x8	6.282	0,218	28.313	0,048	7.126	0,194	5.431	0,258	9.155	0,150	7.152	0,196

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
n_complex_updates_float	x1	8.122	0,168	8.430	0,162	8.122	0,168	8.122	0,168	8.430	0,162	8.430	0,162
	x2	8.189	0,167	10.996	0,124	8.175	0,167	8.207	0,166	8.584	0,159	8.562	0,159
	x4	8.474	0,161	16.871	0,081	8.404	0,162	8.593	0,159	9.103	0,150	9.495	0,144
	x8	10.606	0,129	28.478	0,048	8.721	0,157	10.659	0,132	9.854	0,139	12.814	0,110
ndes	x1	136.363	0,309	153.858	0,274	136.363	0,309	136.363	0,309	153.858	0,274	153.858	0,274
	x2	143.372	0,295	298.125	0,142	142.230	0,297	142.230	0,297	162.715	0,259	162.715	0,259
	x4	173.974	0,247	596.207	0,071	158.560	0,266	158.350	0,267	194.030	0,219	192.849	0,220
	x8	271.807	0,166	1.192.480	0,035	217.360	0,198	181.553	0,234	292.080	0,151	232.765	0,182
n_real_updates_fixed	x1	1.608	0,397	1.833	0,349	1.608	0,397	1.608	0,397	1.833	0,349	1.833	0,349
	x2	1.673	0,382	3.165	0,202	1.685	0,379	1.685	0,379	1.878	0,340	1.878	0,340
	x4	1.902	0,337	6.314	0,101	1.958	0,327	1.565	0,232	2.390	0,268	1.883	0,193
	x8	3.253	0,197	12.642	0,051	3.410	0,189	2.419	0,275	4.469	0,145	3.763	0,176
n_real_updates_float	x1	2.822	0,228	3.054	0,211	2.822	0,228	2.822	0,228	3.054	0,211	3.054	0,211
	x2	2.838	0,227	4.359	0,148	2.832	0,227	2.835	0,227	3.104	0,207	3.104	0,207
	x4	2.977	0,216	7.422	0,087	2.947	0,218	2.991	0,215	3.311	0,194	3.353	0,192
	x8	4.204	0,154	14.700	0,044	3.242	0,198	3.548	0,185	3.888	0,166	4.723	0,141
petrinet	x1	4.573	0,363	5.037	0,330	4.573	0,363	4.573	0,363	5.037	0,330	5.037	0,330
	x2	4.607	0,361	10.045	0,165	4.644	0,358	4.644	0,358	5.225	0,318	5.225	0,318
	x4	4.971	0,334	20.074	0,083	5.232	0,318	5.118	0,325	6.269	0,266	6.059	0,275
	x8	7.323	0,227	40.145	0,041	7.298	0,229	6.462	0,264	9.206	0,182	8.376	0,206
pm	x1	66.639.593	0,085	68.008.685	0,083	66.639.593	0,085	66.639.593	0,085	68.008.685	0,083	68.008.685	0,083
	x2	66.761.537	0,085	81.535.009	0,070	66.756.766	0,085	66.756.766	0,085	68.333.994	0,083	68.333.994	0,083
	x4	50.643.229	0,063	80.758.902	0,040	50.439.429	0,063	50.490.032	0,063	51.965.593	0,061	52.054.266	0,061
	x8	51.694.590	0,062	125.140.262	0,026	50.557.213	0,063	53.761.751	0,059	52.339.271	0,061	56.270.087	0,057
prime	x1	63.496	0,049	64.932	0,048	63.496	0,049	63.496	0,049	64.932	0,048	64.932	0,048
	x2	63.888	0,048	72.073	0,043	63.650	0,049	63.650	0,049	66.447	0,046	66.447	0,046
	x4	65.939	0,047	104.610	0,030	64.528	0,048	64.636	0,048	68.033	0,045	68.192	0,045
	x8	75.436	0,041	140.480	0,022	65.430	0,047	70.489	0,044	69.828	0,044	77.979	0,040
qmf_receive	x1	2.577.575	0,355	2.790.090	0,328	2.577.575	0,355	2.577.575	0,355	2.790.090	0,328	2.790.090	0,328
	x2	2.607.546	0,351	5.170.120	0,177	2.616.779	0,349	2.616.779	0,349	2.881.447	0,317	2.881.447	0,317
	x4	2.769.634	0,330	10.340.221	0,088	2.788.852	0,328	2.788.722	0,328	3.219.418	0,284	3.219.298	0,284
	x8	3.326.198	0,276	20.680.457	0,044	3.349.885	0,273	3.309.780	0,277	4.106.560	0,223	4.045.251	0,226

continued ...

Benchmark	Arch.	FA ∅ACET	FA ∅Util.	TDMA ∅ACET	TDMA ∅Util.	PA (IU) ∅ACET	PA (IU) ∅Util.	PA (AAU) ∅ACET	PA (AAU) ∅Util.	PD (IU) ∅ACET	PD (IU) ∅Util.	PD (AAU) ∅ACET	PD (AAU) ∅Util.
qmf_transmit	x1	2.643.336	0,360	2.865.088	0,332	2.643.336	0,360	2.643.336	0,360	2.865.088	0,332	2.865.088	0,332
	x2	2.660.506	0,357	5.180.196	0,184	2.670.932	0,356	2.670.932	0,356	2.930.798	0,324	2.930.798	0,324
	x4	2.827.125	0,337	10.360.393	0,092	2.923.043	0,326	2.923.177	0,326	3.394.762	0,281	3.395.361	0,281
	x8	3.489.914	0,273	20.720.787	0,046	4.011.720	0,238	3.728.412	0,257	4.915.803	0,195	4.563.770	0,210
qsort-exam	x1	5.010	0,255	5.231	0,244	5.010	0,255	5.010	0,255	5.231	0,244	5.231	0,244
	x2	5.068	0,252	7.637	0,167	5.088	0,251	5.050	0,253	5.366	0,238	5.378	0,237
	x4	5.318	0,240	15.268	0,084	5.331	0,239	5.327	0,240	6.004	0,213	5.974	0,215
	x8	7.405	0,173	30.544	0,042	6.446	0,199	6.675	0,197	8.182	0,157	8.170	0,163
qurt	x1	8.843	0,054	9.033	0,053	8.843	0,054	8.843	0,054	9.033	0,053	9.033	0,053
	x2	8.890	0,054	10.177	0,047	8.881	0,054	8.881	0,054	9.115	0,052	9.115	0,052
	x4	9.214	0,052	12.773	0,037	8.994	0,053	9.112	0,052	9.368	0,051	9.576	0,050
	x8	10.849	0,044	18.567	0,026	9.123	0,052	9.912	0,048	9.636	0,050	11.081	0,044
real_update_fixed	x1	205	0,317	233	0,279	205	0,317	205	0,317	233	0,279	233	0,279
	x2	229	0,284	447	0,145	213	0,305	213	0,305	247	0,263	247	0,263
	x4	244	0,267	878	0,074	215	0,302	221	0,294	295	0,220	297	0,219
	x8	493	0,132	1.752	0,037	500	0,130	217	0,300	686	0,095	280	0,232
real_update_float	x1	397	0,275	428	0,255	397	0,275	397	0,275	428	0,255	428	0,255
	x2	418	0,261	718	0,152	411	0,265	411	0,265	479	0,228	479	0,228
	x4	500	0,219	1.326	0,082	457	0,238	444	0,246	551	0,198	536	0,204
	x8	796	0,137	2.694	0,040	601	0,192	567	0,204	797	0,148	709	0,162
rijndael_decoder	x1	8.200.503	0,342	9.466.339	0,296	8.200.503	0,342	8.200.503	0,342	9.466.339	0,296	9.466.339	0,296
	x2	8.724.998	0,322	18.603.023	0,151	8.931.060	0,314	8.745.344	0,321	10.476.681	0,268	10.476.763	0,268
	x4	10.148.727	0,277	37.206.046	0,075	11.665.122	0,243	9.658.955	0,292	14.182.451	0,200	12.103.293	0,233
	x8	14.136.492	0,199	74.412.074	0,038	18.106.150	0,159	11.657.728	0,247	23.202.337	0,125	15.260.167	0,187
rijndael_encoder	x1	8.141.439	0,339	9.459.906	0,292	8.141.439	0,339	8.141.439	0,339	9.459.906	0,292	9.459.906	0,292
	x2	8.630.515	0,320	18.525.204	0,149	8.542.236	0,323	8.542.236	0,323	10.280.424	0,269	10.280.424	0,269
	x4	9.890.411	0,281	37.050.424	0,075	9.807.151	0,283	9.203.410	0,300	12.281.073	0,227	11.423.832	0,242
	x8	13.899.352	0,201	74.100.853	0,037	14.496.786	0,196	10.877.070	0,258	19.236.665	0,149	14.014.338	0,200
searchmultiarray	x1	21.186	0,178	22.715	0,166	21.186	0,178	21.186	0,178	22.715	0,166	22.715	0,166
	x2	21.419	0,176	26.236	0,144	21.421	0,176	21.509	0,176	22.748	0,166	22.761	0,166
	x4	22.620	0,167	50.454	0,075	21.923	0,172	22.129	0,171	23.953	0,158	24.178	0,156
	x8	25.524	0,148	100.944	0,037	22.781	0,166	23.801	0,159	26.195	0,144	27.050	0,141

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
selection_sort	x1	999.359	0,275	1.089.662	0,252	999.359	0,275	999.359	0,275	1.089.662	0,252	1.089.662	0,252
	x2	1.001.917	0,274	1.818.120	0,151	1.008.684	0,272	1.002.385	0,274	1.092.745	0,251	1.092.760	0,251
	x4	1.022.835	0,268	3.636.232	0,075	1.042.952	0,263	1.015.682	0,270	1.168.777	0,235	1.128.924	0,243
	x8	1.113.672	0,247	7.272.475	0,038	1.272.964	0,216	1.087.184	0,253	1.574.921	0,175	1.260.986	0,219
select	x1	4.002	0,152	4.403	0,138	4.002	0,152	4.002	0,152	4.403	0,138	4.403	0,138
	x2	4.160	0,146	6.523	0,093	4.127	0,147	4.127	0,147	4.568	0,133	4.568	0,133
	x4	4.776	0,127	13.018	0,047	4.505	0,135	4.452	0,136	5.209	0,117	5.108	0,119
	x8	7.205	0,085	26.061	0,023	4.788	0,127	5.302	0,117	5.983	0,102	6.571	0,095
sha	x1	3.298.164	0,181	3.482.700	0,171	3.298.164	0,181	3.298.164	0,181	3.482.700	0,171	3.482.700	0,171
	x2	3.322.082	0,179	5.510.923	0,108	3.324.960	0,179	3.317.439	0,180	3.535.982	0,169	3.535.968	0,169
	x4	3.629.340	0,165	10.776.063	0,055	3.541.903	0,168	3.515.464	0,170	4.031.588	0,148	3.983.551	0,150
	x8	4.975.975	0,122	21.552.130	0,028	4.064.870	0,147	4.192.739	0,145	5.026.405	0,119	5.043.427	0,120
spectral	x1	2.251.130	0,102	2.345.344	0,098	2.251.130	0,102	2.251.130	0,102	2.345.344	0,098	2.345.344	0,098
	x2	2.283.579	0,101	2.956.319	0,078	2.272.536	0,101	2.272.536	0,101	2.398.782	0,096	2.398.782	0,096
	x4	2.368.836	0,097	4.449.109	0,052	2.317.503	0,099	2.309.821	0,100	2.473.606	0,093	2.462.494	0,094
	x8	2.648.526	0,087	7.738.701	0,030	2.395.507	0,096	2.388.264	0,097	2.635.367	0,088	2.605.419	0,089
sqrt	x1	27.096	0,026	27.462	0,026	27.096	0,026	27.096	0,026	27.462	0,026	27.462	0,026
	x2	27.126	0,026	29.238	0,024	27.118	0,026	27.118	0,026	27.519	0,026	27.519	0,026
	x4	27.827	0,025	33.087	0,021	27.405	0,026	27.404	0,026	28.058	0,025	28.045	0,025
	x8	31.038	0,023	41.912	0,017	27.603	0,026	27.790	0,025	28.457	0,025	28.763	0,025
startup_fixed	x1	3.050	0,262	3.276	0,244	3.050	0,262	3.050	0,262	3.276	0,244	3.276	0,244
	x2	3.173	0,252	5.554	0,144	3.115	0,256	3.115	0,256	3.466	0,231	3.466	0,231
	x4	3.550	0,225	11.108	0,072	3.405	0,234	3.457	0,231	4.035	0,198	4.083	0,196
	x8	5.454	0,148	22.219	0,036	4.117	0,194	4.261	0,191	5.346	0,150	5.341	0,152
statemate	x1	268.200	0,427	280.778	0,408	268.200	0,427	268.200	0,427	280.778	0,408	280.778	0,408
	x2	283.485	0,404	541.522	0,211	285.649	0,401	280.881	0,407	315.026	0,363	315.080	0,363
	x4	362.385	0,318	1.083.033	0,106	429.806	0,274	324.880	0,353	516.230	0,228	404.732	0,283
	x8	597.160	0,193	2.166.068	0,053	742.761	0,157	424.773	0,283	786.959	0,114	561.421	0,213
st	x1	1.782.050	0,058	1.834.326	0,057	1.782.050	0,058	1.782.050	0,058	1.834.326	0,057	1.834.326	0,057
	x2	1.784.948	0,058	2.134.449	0,049	1.783.384	0,058	1.785.526	0,058	1.837.927	0,057	1.837.914	0,057
	x4	1.800.219	0,058	2.315.508	0,045	1.790.420	0,058	1.805.005	0,058	1.847.887	0,056	1.866.899	0,056
	x8	1.910.209	0,055	3.896.792	0,027	1.811.675	0,057	1.881.137	0,055	1.881.556	0,055	2.012.678	0,052

continued ...

Benchmark	Arch.	**FA** ∅ACET	**FA** ∅Util.	**TDMA** ∅ACET	**TDMA** ∅Util.	**PA** (IU) ∅ACET	**PA** (IU) ∅Util.	**PA** (AAU) ∅ACET	**PA** (AAU) ∅Util.	**PD** (IU) ∅ACET	**PD** (IU) ∅Util.	**PD** (AAU) ∅ACET	**PD** (AAU) ∅Util.
v32.modem_achop	x1	22.627	0,361	24.068	0,340	22.627	0,361	22.627	0,361	24.068	0,340	24.068	0,340
	x2	23.784	0,344	48.127	0,170	23.477	0,348	23.477	0,348	27.468	0,298	27.468	0,298
	x4	26.655	0,307	96.221	0,085	26.035	0,314	26.069	0,314	31.314	0,262	31.312	0,261
	x8	35.163	0,237	192.457	0,042	31.161	0,263	31.142	0,263	38.919	0,210	39.064	0,209
v32.modem_cnoise	x1	143.944	0,313	154.136	0,292	143.944	0,313	143.944	0,313	154.136	0,292	154.136	0,292
	x2	151.836	0,296	250.490	0,180	148.714	0,303	150.561	0,299	170.132	0,265	170.123	0,265
	x4	188.937	0,239	500.707	0,090	175.465	0,257	181.729	0,250	220.210	0,204	223.353	0,203
	x8	308.257	0,151	1.001.279	0,045	268.734	0,172	218.637	0,211	361.007	0,129	289.291	0,159
v32.modem_ddecode	x1	7.899.364	0,156	8.215.002	0,150	7.899.364	0,156	7.899.364	0,156	8.215.002	0,150	8.215.002	0,150
	x2	7.979.033	0,155	11.441.480	0,108	7.967.399	0,155	7.967.399	0,155	8.450.483	0,146	8.450.483	0,146
	x4	8.438.152	0,146	18.573.332	0,066	8.140.826	0,151	8.271.007	0,149	8.832.024	0,140	9.012.767	0,137
	x8	10.064.509	0,123	34.250.686	0,036	8.267.659	0,149	9.002.835	0,138	9.117.580	0,135	10.413.971	0,120
v32.modem_eglue	x1	26.127	0,367	28.766	0,333	26.127	0,367	26.127	0,367	28.766	0,333	28.766	0,333
	x2	27.052	0,354	57.480	0,167	27.295	0,351	27.295	0,351	29.096	0,329	29.096	0,329
	x4	29.657	0,324	114.960	0,083	31.223	0,308	31.255	0,308	36.488	0,265	36.658	0,263
	x8	39.768	0,245	229.901	0,042	38.644	0,249	38.438	0,250	47.557	0,202	47.257	0,203

Abbildungsverzeichnis

Literaturverzeichnis

[1] AbsInt: *aiT Worst-Case Execution Time Analyzers*, 2013. `http://www.absint.com/ait/index.htm`.

[2] ARM Ltd: *ARM7TDMI Technical Reference Manual (Rev r4p1)*, 2001. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf`.

[3] Bakhkhat, Sidi et al.: *Eingebettete Systeme – Ein strategisches Wachstumsfeld für Deutschland*, 2010. `http://www.bitkom.org/files/documents/EingebetteteSysteme_web.pdf`.

[4] Berns, Karsten, Bernd Schürmann und Mario Trapp: *Eingebettete Systeme - Systemgrundlagen und Entwicklung eingebetteter Software*. Springer DE, Berlin, 2010.

[5] Falk, Heiko und Helena Kotthaus: *WCET-driven Cache-aware Code Positioning*. In: *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, Seiten 145–154, Taipeh, Taiwan, Oktober 2011.

[6] Falk, Heiko und Paul Lokuciejewski: *A compiler framework for the reduction of worst-case execution times*. Journal on Real-Time Systems, 46(2):251–300, Oktober 2010.

[7] Falk, Heiko, Norman Schmitz und Florian Schmoll: *WCET-aware Register Allocation based on Integer-Linear Programming*. In: *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, Seiten 13–22, Porto, Portugal, Juli 2011.

[8] Guthaus, M. R., J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge und R. B. Brown: *MiBench: A Free, Commercially Representative Embedded Benchmark Suite*. In: *Proceedings of the Workload Characterization 2001 IEEE International Workshop*, Seiten 3–14, Washington, DC, USA, 2001. IEEE Computer Society.

[9] Hennessy, John L. und David A. Patterson: *Computer Architecture - A Quantitative Approach*. Elsevier, Amsterdam, 5. Auflage, 2011.

[10] Lee, Chunho, Miodrag Potkonjak und William H. Mangione-smith: *MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems*. In: *International Symposium on Microarchitecture*, Seiten 330–335, 1997.

[11] Lee, Corinna G.: *UTDSP Benchmark Suite.*, 2013. `http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html`.

[12] Lokuciejewski, Paul, Daniel Cordes, Heiko Falk und Peter Marwedel: *A Fast and Precise Static Loop Analysis based on Abstract Interpretation, Program Slicing and Polytope Models*. In: *International Symposium on Code Generation and Optimization (CGO)*, Seiten 136–146, Seattle, USA, März 2009.

[13] Marwedel, Peter: *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Embedded Systems. Springer London Ltd., 2011.

[14] Memik, Gokhan, William H. Mangione-smith und Wendong Hu: *NetBench: A Benchmarking Suite for Network Processors*. In: *International Conference on Computer-Aided Design (ICCAD)*, Seiten 39–42, 2001.

[15] Metzlaff, Stefan, Jörg Mische und Theo Ungerer: *A Real-Time Capable Many-Core Model*. In: *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Wien, Österreich, 29. November - 2. Dezember, 2011*. IEEE Computer Society, 2011.

[16] MOTOROLA, Inc.: *MOTOROLA M68000 FAMILY - Programmer's Reference Manual*, 1992. `http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf`.

[17] Mälardalen WCET Research Group: *MRTC Benchmarks*, 2013. `http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`.

[18] Pinedo, Michael L.: *Scheduling - Theory, Algorithms, and Systems*. Springer DE, Berlin, Heidelberg, 3. Auflage, 2008.

[19] Reineke, Jan, Björn Wachter, Stefan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger und Bernd Becker: *A Definition and Classification of Timing Anomalies*. In: Mueller, Frank (Herausgeber): *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Deutschland, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Deutschland.

[20] Schranzhofer, Andreas, Jian-Jia Chen und Lothar Thiele: *Timing Analysis for TDMA Arbitration in Resource Sharing Systems*. In: *Proceedings of the 2010 16th*

IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS '10, Seiten 215–224, Washington, DC, USA, 2010. IEEE Computer Society.

[21] Shah, Hardik, Andreas Raabe und Alois Knoll: *Priority division: A high-speed shared-memory bus arbitration with bounded latency.* In: *Design, Automation & Test in Europe (DATE)*, Seiten 1497–1500. IEEE, 2011.

[22] SQLite: *SQLite - A self-contained, serverless, zero-configuration, transactional SQL database engine*, 2013. `http://www.sqlite.org/`.

[23] Stream It Developers: *Stream It Testbench*, 2013. `http://groups.csail.mit.edu/cag/streamit/shtml/benchmarks.shtml`.

[24] Synopsys: *Vast CoMET*, 2013. `http://www.synopsys.com/Systems/VirtualPrototyping/Pages/CoMET-METeor.aspx`.

[25] Živojnović, Vojin, Juan M. Velarde, Christian Schläger und Heinrich Meyr: *DSPSTONE: A DSP-oriented Benchmarking Methodology.* In: *Proceedings of the International Conference on Signal Processing and Technology*, 1994.

[26] Wilhelm, Reinhard, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat und Per Stenström: *The worst-case execution-time problem—overview of methods and survey of tools.* ACM Trans. Embed. Comput. Syst., 7(3):36:1–36:53, Mai 2008.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 18. Februar 2013

Tim Harde