Zur Homepage der Dissertation

CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Effizienter Entwurf verteilter eingebetteter Echtzeit-Systeme

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

von

Dipl.-Inform. Alexander Metzner

Gutachter:

Prof. Dr. W. Damm
Prof. Dr. F. Rammig

Tag der Disputation: 19.5.2006

Zusammenfassung

Eingebettete Systeme sind in immer größer werdendem Maße Teil moderner technischer Geräte. Viele Teilfunktionen dieser Systeme sind üblicherweise als sicherheitskritisch einzustufen und müssen daher innerhalb eines gegebenen Zeitintervalls funktional korrekte Ergebnisse produzieren. In komplexen technischen Systemen, wie sie beispielsweise in der Automobilindustrie entwickelt werden, ist dabei eine zunehmende Vernetzung früher unabhängiger Steuergeräte zu beobachten. Der jährlich um bis zu 25% steigende Bedarf an neuen Software-basierten Lösungen im Automobil führt aufgrund der traditionellen Zulieferer-Hersteller-Beziehung auf Basis ganzer Steuergeräte zu einer linearen Abhängigkeit der Anzahl der Steuergeräte von der Menge der Funktionen. So sind bereits in heutigen Oberklasse-Fahrzeugen bis zu 80 Steuergeräte untereinander über mindestens 5 unterschiedliche Bussysteme vernetzt, Tendenz steigend. Diese kostenintensive Abhängigkeit aufzulösen, ist ein starkes Interesse der Automobil- und Zulieferindustrie, welches sich beispielsweise in der Bildung des AUTOSAR-Konsortiums niederschlägt. Ein wesentliches Ziel ist hierbei die Loslösung der Software-Komponenten von proprietären Steuergeräte-Architekturen.

Gegenstand der hier vorgestellten Arbeit ist die Unterstützung dieses Prozesses unter dem Blickwinkel der Echtzeitfähigkeit. Dazu werden formale Analysemethoden verwendet, die ein verteiltes Echtzeitsystem ganzheitlich, d.h. sowohl hinsichtlich der Software-Ausführungszeiten als auch der Nachrichtenübertragungszeiten auf unterschiedlichen Kommunikationsmedien, betrachten. Wohldefinierte Schnittstellen bzgl. des Zeitverhaltens erlauben eine Integration komplexer Echtzeitsysteme im Sinne eines komponenten-basierten Entwurfsprozesses. Ein auf SAT-Checking basierender Verteilungsansatz sorgt für eine optimale Zuordnung der Software-Komponenten zu Steuergeräten in einer heterogenen, hierarchischen Steuergeräte-Architektur. Die Leistungsfähigkeit und die Grenzen dieser Methodik werden an einigen ausgewählten Beispielen aufgezeigt und mit anderen Ansätzen verglichen.
Festgelegte Schnittstellen bzgl. des zeitlichen Verhaltens von Software-Komponenten bedingen jedoch potentiell eine geringere Effizienz. Aufgrund der üblicherweise sehr engen Kostenmargen in eingebetteten Systemen für einen Massenmarkt, ist diesem Effekt durch Optimierungen entgegenzuwirken. Daher werden im zweiten Teil der Arbeit Methoden zur Effizienzsteigerung von Steuergeräten vorgestellt, die direkt in das Verteilungsverfahren und damit den Entwurf der Steuergeräte-Architektur integrierbar sind. Hierzu wird eine effizienzsteigernde Architekturerweiterung für Prozessoren von eingebetteten Systemen diskutiert, die es ermöglicht, die durch ein Echtzeitbetriebssystem bedingten Zeitverluste drastisch zu erniedrigen. Gleichzeitig kann dieses auf Multithreading beruhende Verfahren die Vorhersagbarkeit von Software-Ausführungszeiten unter Berücksichtigung von Preemptionen deutlich erhöhen. Das Konzept wird an einer beispielhaften Prozessorarchitektur illustriert und evaluiert, die bis hinunter zur ASIC-Fertigung durchgeführt wurde.

Abstract

Embedded systems became an increasing part of modern complex technical systems. Typically, most subfunctions of such systems are working in a safety critical environment and therefor have to react within given time intervals. One current trend in developing complex systems, like e.g., in the automotive industry, is the increasing interconnection between former independant electronic control units. The annual growth rate of up to 25% for software based solutions within cars leads to a nearly linear dependency of the number of embedded control units to the number of software based functions. The reason for this is the traditionally installed supplier-manufacturer relationship on the basis of complete electronic control units. The result of this dependency can be seen in looking at today's car electronic system: In premium cars up to 80 ECUs are connected with each other by using at most 5 different bus systems. In order to overcome these high costly dependency, the car manufacturer and supplier industrie installed the so-called AUTOSAR consortium, which goal is to develop software functions independently from proprietary control unit architectures.
The contribution of this thesis is the support of such a process with regard to real-time properties. For this purpose we use formal methods that are able to consider real-time systems by regarding software runtimes as well as message broadcast latencies on different communication media. Well defined real-time interfaces are used to enable an integration of complex embedded real-time systems in the sense of a component based development process. Optimal allocations of software functions to electronic control units in a heterogeneous and hierarchical organized architecture are provided by using an optimisation technique which is based on satisfiability checking. We will show the efficiency of this technique for some benchmark systems and, furthermore, show comparisons to other optimisation methods.
However, one drawback of fixed real-time interfaces for components is the potentially lower efficiency. Due to typically very small cost ranges for production costs of systems delivered to the mass market, we have to counteract this by using optimisations. Therefore, in the second part of this thesis we present methods to improve the efficiency of electronic control units and these techniques are integrated in the allocation optimisation of the first part of this thesis. We propose an extension of the instruction set architecture of processors used in embedded real-time systems which enables a dramatic reduction of time loss resulting from using a real-time operating system. At the same time this technique, which is based on multithreaded architectures, improves the predictability of software runtimes considering the cost inferred by preemptions. The approach is evaluated and illustrated by an examplary development of a processor architecture which was implemented as an ASIC.

Inhaltsverzeichnis

Kapitel 1

Eingebettete Systeme

1.1 Vorbemerkung

Bevor wir mit der Definition eingebetteter Systeme starten, hier zunächst eine Bemerkung zur Wahl der Darstellungsgranularität des Umfeldes dieser Arbeit: Im folgenden Abschnitt wird dies mit einer, für den Typ einer Arbeit wie der hier vorliegenden eher unüblichen, ausführlichen Darstellung und Einordnung eingebetteter Systeme in den gesamtgesellschaftlichen und wirtschaftlichen Kontext beginnen. Diese ausführliche Darstellung wurde gewählt, weil sich hieraus wesentliche Einflüsse auf die Art und Weise der Entwicklung eingebetteter Systeme ableiten lassen, dessen Verständnis wichtig ist, um die generelle Einbettung und Motivation der in dieser Arbeit vorgestellten Methodiken einordnen zu können. Gleichzeitig ergeben sich daraus wesentliche Konsequenzen bzgl. des Entwurfsprozesses, der bei der Entwicklung eingesetzt werden sollte und auch dies zieht Konsequenzen nach sich, die in der hier vorliegenden Arbeit in der Wahl der Methodik reflektiert sind. Folglich ist auch für den Bereich der Entwurfsprozesse eine etwas ausführlichere Beschreibung gewählt worden. Nach der Beschreibung dieses Hintergrundes folgt anschließend eine Einordnung der Beiträge dieser Arbeit in eben jenes Umfeld, gefolgt von einer Übersicht über die einzelnen Kapitel, bevor die eigentliche Kernthematik erörtert wird.

1.2 Bedeutung in der Industrie-Gesellschaft

Als *Eingebettete Systeme* bezeichnet man ein in ein umgebendes technisches System eingebettetes und mit diesem wechselwirkendes Computersystem, das komplexe Steuerungs-, Regelungs- und Datenverarbeitungsaufgaben übernimmt. Ein wesentliches Merkmal dieser Systeme ist die typischerweise fehlende Mensch-Maschine-Interaktion, an deren Stelle Schnittstellen zum umgebenden technischen System mittels Sensoren, die Umgebungseigenschaften dem eingebetteten System bekanntmachen, und Aktoren, die Einfluss auf die Umgebung nehmen können, treten. Sensorik besteht üblicherweise aus Messwertgebern, die bestimmte Umgebungsgrößen messen und mit einer bestimmten Genauigkeit (Werte- und zeitlicher Auflösung) an das eingebettete System weiterleiten. Die Aktuatorik kann aus beliebigen steuerbaren Stellgliedern bestehen, beispielsweise elektro-mechanische Einheiten, wie Magnetventile, Schrittmotoren oder Ultraschallpulsgeber. Diese Art der Ein-/Ausgabeschnittstellen

hat weitreichende Konsequenzen, auch und gerade für die Entwicklung eingebetteter Systeme, da anstelle der sehr differenzierten Benutzerinteraktion der allgemeinen Datentechnik durch menschliche Interaktionen die Notwendigkeit tritt, auf weit weniger differenzierte Messwerte reagieren zu müssen. In der Folge bedeutet dies, dass das eingebettete System ein weit reichhaltigeres Modell seiner Umgebung besitzen muss, als dies in anderen Datenverarbeitungstechnik-Bereichen erforderlich ist. Die im Vergleich zu der flexiblen menschlichen Reaktionsfähigkeit eher beschränkten möglichen Interaktionen mit dem technischen System bedingen zudem, dass eingebettete Systeme auch auf unerwünschtes Verhalten, wie etwa Fehlermeldungen, Ausfall von Teilsystemen oder verspäteten Systemantworten reagieren können müssen. Alle diese Faktoren beeinflussen maßgeblich den Entwicklungsprozess dieser Systeme, und wir werden dies in Kapitel 1.3 noch eingehender diskutieren.
Insbesondere die fehlende Interaktion mit menschlichen Benutzern, aber auch die Anordnung eingebetteter Systeme in der Nähe der von ihnen gesteuerten technischen Systeme[1], führen zu einer kaum vorhandenen Wahrnehmung dieser Systeme in der Öffentlichkeit. Gleichwohl stellen sie eine der wesentlichen Stützen industrieller Wertschöpfungsprozesse dar. Schätzungen von Fachleuten besagen, dass die Zahl der eingebetteten Steuerungssysteme die Zahl der Menschen auf dieser Welt übertreffen. Zwar hat ein Großteil der Weltbevölkerung keinen oder kaum Zugang zu derartigen Systemen, in den hochindustrialisierten Zonen der Welt hingegen hat nahezu jedes Alltagsgerät derartige eingebettete Systeme integriert, mitunter, wie etwa im Automobil, bis hin zu 100 Steuergeräten in einem einzigen technischen Gerät. Die New York Times schätzte 1998 bereits, dass der Durchschnittsamerikaner täglich mit etwa 60 bis 100 eingebetteten Systemen in Berührung gerät. Diese Zahl dürfte sich im Verlaufe der letzten Jahre sicher noch vergrößert haben. Ein wesentlicher Grund für diese hohen Zahlen liegt in der Anwendungsvielfalt; So werden eingebettete Systeme in der Verkehrstechnik (Automobil, Schienenverkehr, Flugzeuge, Schiffe, aber auch Verkehrsleitsysteme wie Ampelanlagen, Verkehrserfassung, Radaranlagen, Funkpeilanlagen, etc.), in der Kommunikationstechnik (Mobiltelefone, Festnetztelefone, Fax, Verbindungsnetzwerke), in Haushaltsgeräten (Mikrowellen, Waschmaschinen, Fernbedienungen, Fernsehgeräte, Wecker, Fotoapperate, ja sogar Toaster), in der Gebäudetechnik (Heizungssteuerung, Schließsysteme, Alarmsysteme), in der Umweltechnik (Kraftwerksteuerung,Emissionskontrolle) und nicht zuletzt massiv in der Medizintechnik (Herzschrittmacher, Hörgeräte, Prothesen, Beatmungsgeräte, Defibrillatoren, Diagnosesysteme, Kernspin-Tomograph, Bestrahlungsgeräte, etc.) eingesetzt. Diese — sicher nicht vollständige — Liste macht eindrucksvoll deutlich, dass es bei der Produktion eingebetteter Systeme um die Bedienung eines Massenmarktes mit entsprechender wirtschaftlicher Bedeutung geht. Die Wertschöpfung liegt dabei nicht ausschließlich bei Produktion und Vertrieb eingebetteter Systeme, sondern sie bestimmen aufgrund der durch sie zusätzlich möglichen Funktionen zunehmend die Konkurrenzfähigkeit und das Image von Produkten. Ein sehr bekanntes Beispiel in diesem Kontext ist das sogenannte Elchtest-Debakel der Mercedes A-Klasse, nach dessen Bekanntwerden dieses Fahrzeug nahezu unverkäuflich war. Erst die Ent-

[1] So sind beispielsweise in der Autmobiltechnik Steuergeräte direkt in der sie beeinflussenden Umgebung eingebaut, beispielsweise die Getriebesteuerung innerhalb der Ölwanne des Getriebeöls.

wicklung und Integration eines Steuergerätes zur Stabilisierung der Lenkausschläge führte in der Öffentlichkeit wieder zu einem positiven Image dieses Fahrzeuges und verbesserte entscheidend die Konkurrenzfähigkeit. Wenige Jahre später ist diese spezielle Fahrer-Assistenz-Funktion aufgrund des Marktdruckes in nahezu jedem Neufahrzeug zu finden, gehört mittlerweile zur Standardausstattung und wird demzufolge in der Öffentlichkeit kaum noch wahrgenommen.
Im Rahmen dieser Arbeit werden wir uns im wesentlichen auf eine Anwendungsdomäne, nämlich die Automobilindustrie, konzentrieren, wobei jedoch die gemachten Aussagen und Beobachtungen so oder so ähnlich auch für alle anderen oben skizzierten Anwendungsfelder getroffen werden können. So lässt sich feststellen, dass im Bereich des Fahrzeugbaus der Anteil der eingebetteten Systeme über die letzten 30 Jahre erheblich gestiegen ist: von ursprünglich ca. 8% der Herstellungskosten eines Fahrzeugs Ende der Sechziger Jahre bis hin zu über 37% in modernen Oberklasse-Fahrzeugen[25] im Jahre 1998. Durch den Trend, in nachfolgenden Fahrzeuggenerationen des niedrigeren Preissegmentes die Errungenschaften und Funktionen von Vorgängermodellen höherer Preisklassen zu übernehmen, breitet sich dieses Phänomen auch im Massenmarkt mit etwas Verspätung aus. Beachtet man gleichzeitig, dass die Kosten für die elektronischen Komponenten wesentlich stärker gefallen sind, als dies bei mechanischen Anteilen eines Fahrzeugs der Fall ist, wird deutlich, dass die Steigerung der Komplexität von eingebetteten Systemen noch wesentlich höher liegt als es der reine Anteil an den Herstellungskosten ausdrückt. Insgesamt ist in den letzten Jahren eine Steigerung der Nachfrage nach zusätzlichen, durch eingebettete Systeme zur Verfügung gestellten, Funktionen von jährlich ca 25% zu verzeichnen[25]. Analysten gehen davon aus, dass diese Zahlen in den nächsten Jahren noch weiter ansteigen werden[40].
Die Folge davon ist eine zunehmende Komplexität der Fahrzeugentwicklung, die letztlich dazu führt, dass beispielsweise in einem einzigen Oberklasse-Fahrzeug aus dem Jahr 1998 bereits über 80 Steuergeräte installiert waren, die über eine Menge an verschiedenartigen Kommunikationskanälen miteinander interagieren konnten. An letzterem Punkt wird auch ein wesentlicher weiterer Trend sichtbar, der erheblich zur Steigerung der Komplexität des Entwurfs beiträgt: Die zunehmende Anzahl an Assistenzsystemen im Automobil verlangt nach einem viel höheren Interaktionsgrad der elektronischen Systeme miteinander. Alle die hier erwähnten Punkte können nicht spurlos am Entwurfsprozess eingebetteter Systeme (aber auch des technischen Systems insgesamt) vorrübergehen. Wir werden uns daher im folgenden zunächst mit dem typischen Entwurfsprozess dieser Systeme auseinandersetzen und mit den sogenannten Rich Component Models eine zukunftsweisende Technik sehr kurz darstellen. Anschließend wird das Thema “Entwurf eingebetteter Systeme” vertiefend unter dem Begriff der Echtzeit betrachtet und leitet damit zum eigentlichen Schwerpunkt der hier vorliegenden Arbeit über, in der anhand eben dieses Schwerpunktes gezeigt wird, welche Methodiken im Entwurf eingesetzt werden können, um derart komplexe Systeme korrekt und im Sinne einer vernünftigen Wirtschaftlichkeit zu entwickeln. Das nun folgende Kapitel über den Entwurfsprozess eingebetteter Systeme im besonderen Hinblick auf die betrachtete Anwendungsdomäne Automobilindustrie bietet hierbei den thematischen Hintergrund für die in den dann folgenden Kapitel dieser Arbeit vorgestellten Techniken.

1.3 Entwurfsprozess eingebetteter Systeme

Der zunehmende Einsatz von Fahrerassistenzsystemen (ABS, ESP, aber auch beispielsweise Autopiloten im Luftfahrtbereich) führen zwingenderweise zu einer erhöhten Kritikalität der eingebetteten Systeme, da sie aktiv Funktionen des menschlichen Benutzer übernehmen oder korrigieren. Ein Versagen dieser Systeme ist aufgrund der damit typischerweise einhergehenden verheerenden Schäden nicht tolerierbar, so dass hier ein besonders sorgfältiger Entwurfsstil vorrausgesetzt werden muss. Entwürfe für derartig komplexe und in punkto Sicherheit besonders anspruchsvolle Systeme werden daher in gewisser Weise standardisiert in Form von sogenannten Vorgehensmodellen, die beschreiben, in welcher Art und Weise ein technisches Gerät entwickelt werden sollte, welche Dokumentationen angefertigt werden müssen und wie die zeitlichen Abläufe der Entwicklung organisiert werden müssen. Eines der bekanntesten und im Bereich des Entwurfs eingebetteter Systeme wohl am weiten verbreiteste Vorgehensmodell ist das sogenannte V-Modell[49], wie es in Abbildung 1.1 dargestellt ist.

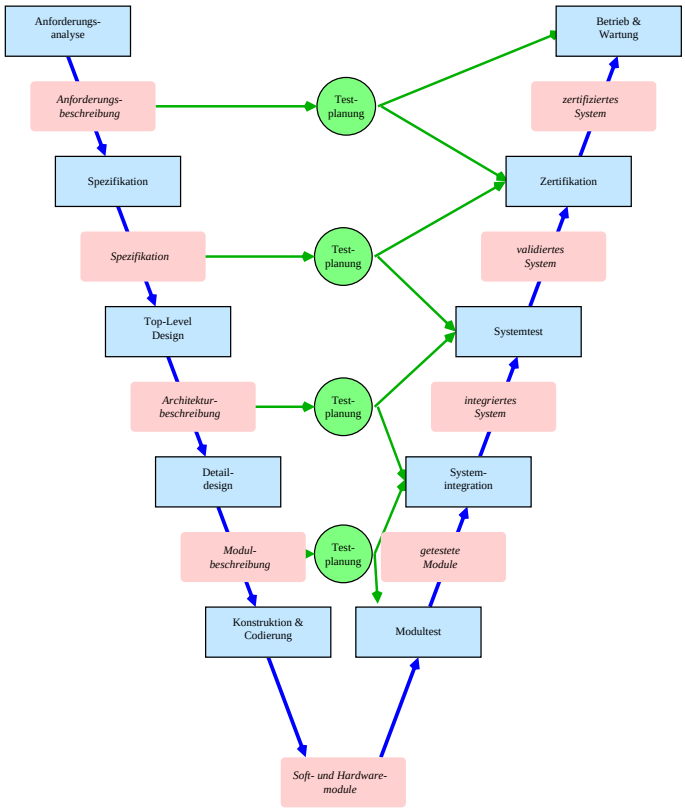


Abbildung 1.1: Das V-Modell mit seinem beiden Ästen: Links die Verfeinerung des Entwurfes, rechts die aufsteigende Integration der Teilkomponenten, sowie die zugehörigen Testszenarien.

Das V-Modell gliedert die Entwicklungsphasen, beginnend mit der Anforderungsanalyse und endend beim Betrieb des Systems, in zwei übergeordnete Phasen:

- eine von der konstruktiven Tätigkeit dominierte Modularisierungs- und Verfeinerungsphase, in der die Systembeschreibung sukkzessive detaillierter wird,

und

- eine von prüfenden Tätigkeiten dominierte Integrationsphase, in der die Teilprodukte zum betriebsfertigen Gesamtsystem integriert werden.

Diese beiden Phasen werden grafisch als die ab- und die aufsteigende Flanke eines "V" dargestellt, so dass in der grafischen Darstellung der Detaillierungsgrad der Betrachtungsweise nach unten hin zunehmend auf der y-Achse und der zeitliche Verlauf nach rechts zunehmend auf der x-Achse aufgetragen sind. Mit den konstruktiven Tätigkeiten einhergehen sollte die Planung von auf der jeweiligen Betrachtungsebene durchführbaren Validations- und Verifikationsmaßnahmen, z.B. Testfällen, die dann in der Integrationsphase auf der korrespondierenden Betrachtungsebene verwendet werden. Dieser Querbezug ist in Abbildung 1.1 durch "Brücken" zwischen den beiden Flanken dargestellt.
Eine besondere Betonung liegt im V-Modell auf der Dokumentation der ersten Phasen des konstruktiven Astes. Dies unterscheidet dieses Modell von anderen Vorgehensmodellen, wie beispielsweise Wasserfall-Modell oder Spiralmodell (vgl. [137]) und trägt der Beobachtung Rechnung, dass Änderungen am System in frühen Phasen des Entwurfes deutlich kostengünstiger sowohl vom Umfang der Änderung als auch bzgl. der Integration sind (siehe beispielsweise [180]).

Für eine Anwendung dieses Vorgehensmodelles auf komplexe technische Systeme, wie etwa Automobile oder Flugzeuge, reicht diese vereinfachende Darstellung aus Abbildung 1.1 allerdings nicht aus: Neben zahlreichen sogenannten transversen Prozessen (vgl. [77]) ist zudem zu berücksichtigen, dass die Entwicklung der Komponenten einer verfeinerten Spezifikation des Systems ebenfalls, parallel zum generellen Ablauf des V-Modells, einen Durchlauf durch eben dieses Modell vollführen. In bestimmten Fällen sind dies eigenständige Entwurfszyklen, die innerhalb des globalen Durchlaufes durchgeführt werden müssen[180]. Zudem ist oftmals die Entwicklung sogenannter Enabling Products (beispielsweise Teststände) notwendig, die mit ihrem Ablaufprozess zwar mehr oder weniger quer zum Ablauf der Systementwicklung an sich liegen, aber fest definierte Anknüpfungspunkte haben (vgl. wiederum [180] bzw. [77]). Diesen Umständen ist insbesondere in stark von Zulieferern geprägten Industrien Rechnung zu tragen, wie es die Automobilindustrie und in noch höherem Maße die Luftfahrtindustrie ist. Typischerweise werden in derartigen Konstellationen Anforderungsspezifikationen an einen Zulieferer übermittelt, der aufgrund dieser Spezifikationen den Implementierungsprozess gemäß obigen Modell durchführt. Dafür ist ein mühsamer Konsultationsprozess zur Validierung der Anforderungen notwendig, der mittlerweile nach und nach durch ausführbare Spezifikationen in einem modell-basierten Entwicklungsprozess[191, 25] ersetzt wird. Diese Kontrakte werden üblicherweise mit einer großen Anzahl unterschiedlicher Zulieferer für die verschiedenen Funktionalitäten des Systems durchgeführt, so dass in der Folge eine massive Parallelentwicklung stattfindet (sogenanntes Concurrent Engineering) und der Fahrzeughersteller erst in der Integrationsphase die Teilkomponenten geliefert bekommt. Aufgrund der mühsamen Absprachen zwecks Validation der Anforderungen gegenüber dem Fahrzeughersteller als primären Stakeholder, ist im Zuge des vermehrten Aufkommens ausführbarer Spezifikationen die Bemühung der Fahr-

zeughersteller, dem Zulieferer derartige Spezifikationen zu liefern und damit die Verantwortung seitens der Fahrzeughersteller nach unten auf dem linken Ast des V-Modells zu verschieben[59]. Gleichzeitig wird die Eigenentwicklung von Teilsystemen zunehmend ausgelagert, was dazu führt, dass Fahrzeughersteller "nur" noch Systemintegration übernehmen; Ein Trend, den die Luftfahrtindustrie bereits gesetzt hat. So ist zu erwarten, dass der Anteil der Zulieferer an der Konstruktion eines Automobils von heute bereits ca. 65% in den nächsten 10 Jahren auf über 75% ansteigen wird (vgl. [40]). Aus der Sicht der Fahrzeughersteller wird damit die Integrationsphase eine zunehmend starke Bedeutung erfahren, die sich unter anderem in klar definierten Schnittstellen, insbesondere in den Spezifikationen der Teilsysteme, niederschlägt, die gleichzeitig eine gewisse Allgemeingültigkeit zur Erleichterung der Integrationsphase haben müssen. In diesem Sinne ist auch die Einrichtung des AUTOSAR[69] Konsortiums zu sehen, in dem eine Standardisierung von Komponenten eingebetteter Systeme erfolgt ist, die eben jene allgemeingültigeren Schnittstellen durch eine Kapselung von Standardfunktionen (etwa Steuergeräte-Architekturen, Zugriffe auf typischen Sensorik und Aktuatorik, etc.) zum Ziel hat. Dies ermöglicht auf der einen Seite eine deutlich einfachere Kommunikation mit Zulieferern und beschränkt auf der anderen Seite die Konkurrenz der Fahrzeughersteller nur noch auf den angebotene Funktionalitätsumfang und technische Innovationen. Gleichzeitig erleichtert es die Integration und die Exploration von Designalternativen. Dies bildet zugleich den Hintergrund der in dieser Arbeit vorgestellten Methodik und wir werden dies daher noch etwas detaillierter in Kapitel 3 bzw. eingehender in Kapitel 6 diskutieren.

Wie es durch den Verfeinerungsansatz des V-Modells bereits angedeutet ist, erfolgt die Implementierung eingebetteter Systeme zunehmend komponentenbasiert. Dies ist aufgrund hoher Margen von Wiederverwendung ein durchaus erwünschter Effekt, der zu mehr Wirtschaftlichkeit beitragen soll. So ist bereits heute ca. 80% der Fahrzeugelektronik aus Vorgängermodellen wiederverwendbar[40] und Standardisierungen, wie beispielsweise durch das AUTOSAR-Konsortium vorgenommen, werden die Effektivität des Re-Uses noch steigern. Gleichzeitig muss man sich in diesem Zusammenhang aber mit der möglichen Interaktionen von Komponenten beschäftigen (beispielsweise könnten zwei Komponenten gleiche Aktuatoren benutzen, so dass die Komposition dieser beiden Komponenten eingehend analysiert und ggf. um Synchronisationspunkte erweitert werden muss). Da der Entwurf eingebetteter Systeme sich aufgrund deren Sicherheitskritikalität längst nicht mehr nur auf funktionale Anteile beschränkt, sind hier auch Interaktionen bzgl. anderer Sichten zu betrachten. Beispielhaft für unterschiedliche Sichten sind hier funktionale Korrektheit, zeitliche Korrektheit, Ausfallsicherheit, Verfügbarkeit, Resourcenverbrauch, etc. zu nennen, deren Berücksichtigung in die Implementierung bzw. Integration durch die sogenannten Design-for-X Techniken abgebildet werden. Komponentenmodelle müssen all diesen X-Sichten Rechnung tragen können. Ein vielversprechender Ansatz, dies zu gewährleisten, stellen die sogenannten Rich Component Modelle[37, 39] (kurz RCM) dar. Sie bieten zugleich ein Framework im Rahmen des V-Modells, in dem die Exploration des Entwurfsraumes enthalten ist. Insbesondere gliedern sich die im Rahmen dieser Arbeit entstandenen Techniken direkt in dieses Framework ein, so dass es im Folgenden kurz beschrieben werden sollen.

Rich Component Modelle sind Komponentenmodelle, die um zusätzliche Sichten angereichert sind, die jeweils den Design-for-X Klassifikationen entsprechen können. Dabei garantiert jede Komponente an ihren Schnittstellen den mit ihr komponierten anderen Komponenten ein spezifischen Verhalten (Promise), solange bestimmte Vorraussetzungen eingehalten werden. Diese Vorraussetzungen werden typischerweise ebenfalls in Schnittstellen zu anderen Komponenten definiert und stellen eine Erwartung an das Verhalten der Umgebung dar (Assume). Zusammen stellen diese beiden Ansätze einen sogenannten Assume-Guarantee Mechanismus zur Verfügung, der eine einfache, formal nachweisbar kompatible Komposition von Komponenten möglich machen soll. In Kapitel 6 wird deutlich werden, wie eine derartige Schnittstelle beispielsweise im Bereich der Echtzeit-Sicht ausgestaltet werden kann. Die — analog zum V-Modell — sukkzessive Dekomposition von zu entwerfenden Systemen erfolgt dabei in zwei unterschiedlichen Schritten: Im übergeordneten Schritt wird der Entwurfsraum exploriert, in dem ein Schichtenmodell mit unterschiedlichen Abstraktionsniveaus existiert. In jeder Schicht (die sogenannten Layer) wird die Spezifikation des Systems in der Granularität des gewählten Layers durchgeführt, wobei dies im Einzelnen sind: System Layer beschreibt das Zusammenspiel der technischen Systeme untereinander (üblicherweise Protokollverhalten, kooperatives Verhalten, etc.), Function Layer beschreibt die funktionale Ausgestaltung des technischen Systems an sich (abstrakte Spezifikationen des funktionalen und nicht-funktionalen Verhaltens), ECU Layer beschreibt die Implementierung von ausführbaren Einheiten (Code-Segmente, abstraktere Netzwerk-Topologien, etc.) und der Hardware-Layer schließlich beschreibt die physikalische Implementierung elektronischer Komponenten (ECUs, Prozessoren, Bussysteme, etc.). In jedem Layer-Übergang befindet sich eine sogenannte Mapping-Funktion, die die Komponentenmodelle des übergeordneten Layers den Komponenten auf dem darunter liegenden Layer zuordnet. Der Übergang zwischen den Layern ist gleichzeitig wiederum mit einem Assume-Guarantee Mechanismus ausgestattet, der es erlaubt, auf dem höheren Layer beispielsweise Reaktionszeiten zu versprechen, genau dann wenn der darunter liegende Layer bestimmte Laufzeiten von Codesegmenten garantiert. Auf diese Art und Weise kann neben der kompositionellen Entwurfsmethodik auch eine virtuelle Integration bereits in sehr frühen Entwurfsphasen erfolgen, wobei die Mapping-Funktionen hier den klassischen Part der Design-Space-Exploration übernehmen.
Die zweite Art der Dekomposition ist die Verfeinerung von Komponenten auf einem Layer. Auch hierzu wird ein Schichtenmodell, die sogenannten Level, eingeführt, dass allerdings in seiner Tiefe nicht beschränkt ist und der reinen Dekomposition im klassischen Sinne von hierarchischen Entwurfsstilen entspricht. Auch hier finden sich wiederum vertikale Schnittstellen im Sinne des Assume-Guarantee Mechanismus, allerdings existieren keine Mapping-Funktionen, da sich die Spezifikation auf dem gleichen Abstraktionsniveau befinden.
Diese sehr komprimierte Vorstellung der RCMs soll an dieser Stelle ausreichen, insbesondere da wir uns im Kontext dieser Arbeit lediglich mit den Mapping-Funktionen zwischen Function Layer und ECU Layer, sowie mit der Komposition von Komponenten auf dem ECU Layer beschäftigen werden. Für weitergehende Informationen sei hier auf [37] verwiesen.

Nach der Beschreibung des generellen Hintergrundes kann nun die Fokussierung auf eine spezielle Eigenschaft, nämlich die der Echtzeit als eine der wesentlichen, für die Sicherheitskritikalität fundamentalen Eigenschaften, erfolgen. Anschließend wird der Beitrag, den diese Arbeit liefern soll, in den bisher erörterten Zusammenhang gestellt und die Organisation der hier vorliegenden Arbeit beschrieben.

1.4 Sicherheit und Echtzeit

Funktionale Korrektheit innerhalb des Entwurfes eingebetteter Systeme ist, wie oben bereits festgestellt, nur ein Aspekt unter vielen. Ein weiterer, ebenso wesentlicher, nicht-funktionaler Aspekt betrifft die Rechtzeitigkeit von Reaktionen des eingebetteten Systems auf ankommende Ereignisse. Rechtzeitigkeit meint im Allgemeinen, dass die Reaktion des Systems (sichtbar an neuen Stellgrößen für Aktuatoren oder aber Erneuerung des internen Systemzustands) auf eine Zustandsänderung innerhalb definierter Zeitspannen erfolgt. Eingebettete Systeme, die diese Eigenschaften garantieren können, werden Echtzeitsysteme (engl. Real-Time Systems) genannt. Echtzeitreaktionen verlangen dabei nicht die schnellst mögliche Reaktion, sondern lediglich diejenige Reaktionsgeschwindigkeit, die notwendig ist, die zeitlichen Anforderungen zu erfüllen. In der Praxis wird daher unter Kostengesichtspunkten üblicherweise die geringst mögliche Reaktionszeit durch die Wahl entsprechend leistungsarmer, aber preiswerter Bausteine anvisiert, die aber noch innerhalb der Anforderungen liegt. Gleichzeitig ist es inakzeptabel, wenn die Reaktionszeit zeitweise über das notwendige Maß hinaus verlängert wird, wie dies beispielsweise Standard-Betriebssysteme bei der Abarbeitung interner Prozesse (etwa beim Starten des Speicherverwaltungsmangement) für mehrere Zehntelsekunden üblicherweise tun. Stattdessen ist in Echtzeitsystemen eine *Garantie* der Einhaltung von Zeitschranken gefordert.
Als illustrierendes Beispiel soll hier ein Anti-Blockier-System betrachtet werden, dessen Arbeitsweise in 3 Phasen untergliedert ist:

1. Es wird Druck im den Bremsbelag anpressenden Bremskolben aufgebaut, bis der vom Fahrer gewählte Bremsdruck erreicht wird.

2. Der Druck wird gehalten, bis durch einen Vergleich mit den Umdrehungsgeschwindigkeiten der anderen Räder Schlupf am gesteuerten Reifen festgestellt wird.

3. Der Bremsdruck wird abgebaut, bis durch den Vergleich mit den Umdrehungsgeschwindigkeiten der anderen Räder die Wiederherstellung des freien Radlaufes festgestellt wird.

Die verschiedenen Phasen werden vom zugehörigen ABS-Steuergerät durch Steuerung eines Ventils in der zum jeweiligen Bremskolben gehörigen Bremsleitung eingeleitet. Offensichtlich muss die Umsteuerung der Ventile zeitnah zum jeweiligen Phasenübergang erfolgen, da ansonsten unnötig lange Blockierungsphasen (Übergang Phase 2 zu Phase 3) oder sogar der zeitweilige Verlust der Bremsen (Übergang Phase 3 zu Phase 1) auftreten können.
Ursachen für die Verletzung von Zeitschranken können jedoch mitunter diffizil zu

entdecken sein. So beschreibt die NASA ein Problem des Mars-Landers Sojourner, der zunächst unerklärlicherweise ab und zu einen vollständigen Neustart durchführte, der zum Totalverlust der an diesem Tag gesammelten Daten führte. Die Ursache war letztlich die Blockierung eines Speicherbereichs durch eine unwichtige, aber langsame Funktion. Parallel ablaufende Systeme reagierten auf diese Blockierung planungsgemäß mit einem Neustart in Erwartung eines größeren Fehlverhaltens des Systems. Durch eine einfache Umstellung des auf dem Sojourner befindlichen Echtzeitbetriebssytems konnte dieser Fehler zwar behoben werden, aber das Auffinden eines solchen Fehlers gestaltet sich mitunter als äußerst schwierig, insbesondere wenn er nur sporadisch und unter bestimmten, nicht erwarteten Zuständen eintritt (siehe unten).

Das zunehmende Aufkommen von Assistenzsystemen, die direkt regulierend in die vom menschlichen Bediener des Systems getroffenen Entscheidungen eingreifen, steigert zwar die Sicherheit bei der Benutzung des Systems an sich, führt aber auch zu höheren Qualitätsanforderungen. Insbesondere die Verdrängung mechanischer Kopplungen zugunsten elektronischer, wie es beispielsweise durch die sogenannten X-by-Wire-Techniken[1] eingeführt wird, erhöhen den Druck auf die Industrie, aber auch auf Genehmigungsbehörden, für eine hohe Qualität der Systeme zu sorgen. Im Bereich der Luftfahrt hat dies schon eine lange Historie in Form von beispielsweise JAA,EASA und Luftfahrtbundesamt, die für eine Zertifizierung von allen Systemteilen zuständig sind und dabei insbesondere ein starkes Augenmerk auch auf die Entwicklungsprozesse legen. Mit dem vermehrten Aufkommen von X-by-Wire-Techniken könnten ähnliche Zertifizierungsprozesse auch im Bereich der Automobilindustrie durch die Technischen Überwachungsvereine installiert werden. Die jeweils geforderten Entwurfsprozesse definieren üblicherweise auch, wie bestimmte Eigenschaften, hierunter fallen insbesondere auch Echtzeiteigenschaften, nachzuprüfen sind. So gibt es beispielsweise im V-Modell Methodensammlungen, die zu jedem Entwurfsschritt bestimmte Verifikations- und Validationsmethoden vorschreiben, die allerdings im Allgemeinen recht oberflächlich gehalten sind. So sind gemäß [50], Activity SD 5.2-SW (Analysis of Resource and Time Requirements) für den Bereich der Echtzeitsysteme simulationsbasierte Methoden und sogenannte "Analysis of Covert Channels" vorgeschrieben. Für letztere schreibt der Standard bzgl. Echtzeitanalysen für Kommunikationskanäle in [50]: "[...] Up to now there are no formal methods for the analysis of time channels. [...]". Stattdessen wird die Verwendung der sogenannten Shared Resource Methodology (SRM) vorgeschlagen, die im wesentlichen einer Kommunkationsmatrix entspricht und damit nur einfache, statische Analysen ermöglicht, nicht jedoch komplexes Zeitverhalten abbilden kann. Im Bereich der Quality Assurance werden die Methoden Review, Simulation und Test vorgeschrieben. Das Fehlen eines formalen Vorgehens zur Verifikation des Echtzeitverhaltens führt häufig dazu, dass die Stakeholder nur Systeme akzeptieren, deren allgemeine Charakteristika unter bestimmten Grenzen liegen und daher mit hoher Wahrscheinlichkeit ein Fehlverhalten, welches durch die obigen Methoden nicht aufgedeckt wurde, nicht vorkommt (Das Fehlverhalten des oben erwähnten

[1] Beispielsweise gibt es dann keine mechanische Verbindung mehr vom Lenkrad zur Vorderachse, sondern die Stellung des Lenkrades wird mittels Sensorik ausgelesen und elektronisch an ein die Lenkung steuerndes Steuergerät übermittelt.

Mars-Landers zeigt übrigens, dass dies so nicht immer zutreffen muss, sondern eher eine subjektive Einschätzung der Stakeholder bzw. Entwickler darstellt). Die Aussage eines Integrations-verantwortlichen Ingenieurs auf einem Workshop zum Thema Systems Engineering mit dem Inhalt: "[...]Wir akzeptieren nur Steuergeräte, bei denen die Auslastung auf den Prozessoren unter 50% liegen.[...]" verdeutlicht dieses Verhalten, d.h. das Fehlen formaler Analysen verursacht die Einführung von Sicherheitsmargen, die allerdings von Unternehmen zu Unternehmen unterschiedlich sind, nirgendwo standardisiert werden und generell keine Garantie der zeitlichen Rechtzeitigkeit darstellen können.
Die Gründe für diese relativ groben Sicherheitsmargen im Zeitverhalten liegen offensichtlich darin, dass simulative oder messungsbasierte Methoden zum Nachweis von Echtzeiteigenschaften zwei schwerwiegende Nachteile haben: Zum Einen bieten sie keine Garantie des zeitlichen Verhaltens, da nur bestimmte, vom jeweiligen Tester ausgewählte Szenarien einer Überprüfung unterzogen werden können, die nicht vollständig sein können und die zudem der Gefahr unterliegen, dass eher weniger naheliegende Testszenarien nicht abgeprüft werden, die möglicherweise aber ein nicht der Anforderung entsprechendes zeitliches Verhalten aufweisen. Beispielsweise hätte die simulative Erkennung des oben beschriebenen Fehlers im Mars Sojourner die Simulationszeitspanne eines realen Tages für ein konkretes Testszenario bedurft, was üblicherweise mit heutigen Simulationssystemen nicht in adequater Laufzeit bewerkstelligt werden kann[1]. Zum Anderen sind derartige Methoden nur sehr schwer in der Explorationsphase des Entwurfsraumes einzusetzen, da sie in der Regel manueller Interaktion bedürfen und außerdem sehr umfangreiche Komplexitäten und Laufzeiten besitzen.
Die Benutzung formaler Analysemethoden bietet hingegen sowohl eine Garantie des zeitlichen Verhaltens als auch eine einfache Integration in einen automatisierten Prozess zur Exploration des Entwurfsraumes. Die Kosten hierfür sind allerdings nicht zu vernachlässigen, da formale Analysemethoden jeden denkbaren Fall berücksichtigen müssen, also im Allgemeinen mit einer überapproximierten Sicht auf das eingebettete System arbeiten. Hier muss also das Bestreben im Vordergrund stehen, neben der formalen Absicherung der Analysen an sich, eine möglichst akkurate und wenig approximierende Modellierung des zu analysierenden Systems abbilden zu können. Die grundlegenden Techniken hierfür werden in Kapitel 2.3 sowie in Kapitel 2.4 dargestellt und insbesondere bezüglich der Analysen aus Kapitel 2.3 im Rahmen dieser Arbeit verfeinert.

1.5 Stellung dieser Arbeit im Themenumfeld

Natürlich kann eine Arbeit wie die hier vorliegende nicht alle in den vorangegangenen Kapiteln angesprochenen Probleme lösen. Stattdessen soll hier eine Auswahl von einigen wesentlichen Punkten innerhalb des gesamten Prozesses zur Entwicklung komplexer eingebetteter Echtzeitsysteme getroffen werden, an denen substan-

[1] Modernste Simulationstechniken können im Verhältnis 1:10 bis 1:100 simulieren. Nur mit dem Einsatz sehr teuerer und komplexer Hardware-Simulatoren können Simulationen bis hinunter zum 1:1 Verhältnis durchgeführt werden.

tielle Verbesserungen des Stands der Technik vorgenommen werden sollen. Der Titel dieser Arbeit ist mit "Effizienter Entwurf verteilter eingebetteter Echtzeit-Systeme" bewusst mehrdeutig gewählt worden und kann hier in beiden Bindungsvarianten des Wortes "effizient" betrachtet werden: Wir werden in den folgenden Kapitel sowohl einen **effizienten Entwurf** verteilter eingebetteter Echtzeitsysteme betrachten, als auch einen Entwurf **effizienter verteilter eingebetteter Echtzeit-Systeme**. Ersteres wird durch eine automatische Unterstützung der Exploration des Entwurfsraumes, sowie durch eine aus den typischen Entwurfsprozessen der betrachteten Anwendungsdomänen getriebenen komponenten-basierte Integration im Concurrent Engineering zur Verfügung gestellt. Zweiteres erfolgt durch exakte Analysen insbesondere des Zeitverhaltens von Kommunikationen sowie durch die Bereitstellung spezifischer Architekturmerkmale. Insgesamt werden die folgenden Schwerpunkte innerhalb dieser Arbeit bearbeitet:

A Formale Analysen des Zeitverhaltens von Funktionsnetzwerken, insbesondere mit einem Schwerpunkt auf der Betrachtung von Kommunikationen auf unterschiedlichen Bussystemen

B Formale Analysen des Zeitverhaltens in gemischt Ereignis- und Zeit-basierten Systemen

C Automatische Platzierung von Ausführungseinheiten (Tasks) auf heterogene verteilte Systeme mittels eines optimalen Suchverfahrens

D Unterstützung von heterogenen, hierarchisch organisierten Bus-Topologien innerhalb der automatischen Platzierung

E Definition und Synthese von Echtzeitschnittstellen für eine komponenten-basierte inkrementelle Integration

F Berücksichtigung des Zeitverhaltens von Laufzeitumgebungen innerhalb der formalen Analyse von Funktionsnetzwerken

G Architekturerweiterungen von Prozessoren eingbetteter Systeme zur Steigerung der Performanz von Echtzeitsystemen und zur gleichzeitigen Steigerungen der Vorhersagegenauigkeit von Laufzeiten

Dieses Themenspektrum gliedert sich auf unterschiedlichste Weise in die verschiedenen Aspekte des Entwurfes von eingebetteten Echtzeitsystemen ein und ist insbesondere auch innerhalb komplexer komponenten-basierter Prozesse, wie etwa den schon vorgestellten Rich Component Modellen, einsetzbar. Im Einzelnen sollen die hier vorgestellten Techniken die folgenden Aspekte im Entwurf unterstützen:

- Generelle Aspekte der Sicherheit und Echtzeit, wie sie in Kapitel 1.4 beschrieben sind, verlangen die formale und sichere Betrachtung des Zeitverhaltens von verteilten Echtzeitsystemen. Dies wird hier durch die Techniken A, B, D, E, F und G unterstützt.

- Entwurfsprozesse, insbesondere Concurrent Engineering ganzer Systeme, bei denen die Echtzeitversprechungen ein Teil sind, werden durch die Techniken C, D und E abgedeckt.

- Die Forderung nach Verkürzung der Entwurfszeiten (Time-to-Market) kann durch die Techniken A bis G insgesamt unterstützt werden, wobei hier sicherlich ein Schwerpunkt in den Techniken C und E zu sehen ist, allerdings können auch formale Analysen wesentlich zur Verkürzung der Testzeit betragen.

- Vermehrter Einsatz von Re-Use und komponenten-basierten Entwurfsstilen mit womöglich frühen prototypischen Abschätzungen des Zeitverhaltens wird durch die Techniken A, B, C, D und E unterstützt. Hier ist zudem die Kompatibilität zu den AUTOSAR-Ansätzen zu beachten, die durch die Techniken C, D und E erfolgt.

- Die Reduzierung der Steuergeräte-Anzahl (Teil der Wirtschaftlichkeit und Entwurfsprozessdimension) kann durch die Techniken C, F und G unterstützt werden.

Generell liefern die verschiedenen Techniken vor allen Dingen Beiträge in den Bereichen der Vorhersagegenauigkeit von Echtzeitsystemen (Techniken A, B, F und G), in der Exploration des Entwurfsraumes (Techniken C, D und G), sowie in der Unterstützung von Entwurfsprozessen (Techniken C und E).

1.6 Aufbau der Arbeit

Die hier vorliegende Arbeit ist, nach den einführenden Kapiteln über die Problemstellung und die generelle Einführung in die Thematik "Echtzeitsysteme", in zwei Teile gegliedert. Während sich der erste Teil vornehmlich der Frage der Platzierung von Ausführungseinheiten auf heterogene, verteilte Architekturen und dessen Integration in einen Entwurfsprozess widmet, liegt der Schwerpunkt des zweiten Teils in der Optimierung des Laufzeitverhaltens von Ausführungseinheiten durch Erweiterungen der Architektur von in eingebetteten Echtzeitsystemen eingesetzten Prozessoren. Aufgrund der Vielschichtigkeit der Thematik sind die Betrachtungen des Stands der Technik für die jeweiligen Teil-Thematiken in den entsprechenden Kapiteln angesiedelt. Ebenso werden mögliche zukünftige Weiterentwicklungen jeweils im Anschluss an die Darstellung der Themen diskutiert. Im Einzelnen gibt die nun folgende Aufstellung Auskunft über die Inhalte der einzelnen Kapitel dieser Arbeit.

In Kapitel 2 wird die formale Grundlage für Zeitanforderungen an ein eingebettetes System und die für den Nachweis dieser Bedingungen notwendigen Methodiken dargestellt. Dabei wird zunächst von klassischen Anforderungen im Entwurf eines eingebetteten Echtzeitsystems ausgegangen, und anhand der spezifischen Requirements werden Techniken und Begriffsbildungen für den Umgang mit Echtzeitanforderungen motiviert. Dies umfasst neben der Klassifikation typischer Ausführungsszenarien auch die Implikationen auf entsprechend zu wählende Laufzeitumgebungen sowie die Analyse-Methodiken zur Sicherstellung der Einhaltung angegebener Echtzeitbedingungen.
Kapitel 3 führt anschließend in die Problematik im Entwurf verteilter eingebetteter Echtzeitsysteme ein und motiviert stellvertretend aus der Anwendungsdomäne der

Automobiltechnik heraus die Herausforderungen für den Systementwurf derartiger eingebetteter Echtzeitsysteme. Die Charakteristika eines verteilten Systems werden hierbei unter dem besonderen Blickwinkel der Echtzeitfähigkeit, aber auch unter Beachtung adequater Entwurfsprozesse, betrachtet und führen schließlich zu der Frage der Verteilung von Ausführungseinheiten auf die verteilte Architektur.
Basierend auf diesen Beobachtungen werden in Kapitel 4 das Platzierungsproblem sowie die dazu notwendigen Echtzeitanalysen formalisiert. Ein besonderer Schwerpunkt liegt hierbei in der Analyse des Zeitverbrauches von Nachrichtenübertragungen in vernetzten Systemen, wobei erstmals Systeme analysiert werden können, die gemischte Varianten der prinzipiell unterschiedlichen typischen Aktivierungsszenarien von Ausführungseinheiten, sogenannte Ereignis-basierte und Zeit-basierte Aktivierungen, verwenden. Ferner werden Einflüsse der Topologie verteilter Architekturen auf die Analysetechnik diskutiert und daraus unter dem Gesichtspunkt eines typischen Entwurfsprozesses obere Schranken für diese Einflüsse definiert. Eine Betrachtung des Optimierungspotentials der Echtzeitanalysen unter Ausnutzung von Präzedenzbedingungen schließen dieses Kapitel ab.
Kapitel 5 widmet sich der Auswahl und Modellierung eines adequaten Optimierungsverfahrens, dass in der Lage ist, eine automatische Zuweisung von Ausführungseinheiten auf eine gegebene, hierarchische und verteilte Architektur durchzuführen. Hier wird, unter Berücksichtigung der bekannten Optimierungsverfahren, die Wahl eines Erfüllbarkeits-Entscheidungsverfahren als zentrales Optimierungsverfahren motiviert und die Modellierung der in Kapitel 4 vorgestellten Formalismen angegeben. Im Zuge dieser Modellierung wird ebenfalls erklärt, wie eine Platzierung der Ausführungseinheiten und Nachrichten auf einer hierarchischen Topologie durchgeführt werden kann. Es wird gezeigt, wie ein Entscheidungsverfahren für Erfüllbarkeit als Optimierungsmethodik eingesetzt werden kann und welche möglichen Erweiterungen vorstellbar sind. Anschließend daran findet sich eine Diskussion zur Übertragung der Modellierung auf andere Optimierungsverfahren. Diverse Evaluationen bewerten schließlich den vorgestellten Ansatz auch im Vergleich zu bekannten anderen Optimierungsverfahren.
Die Integration von Platzierungsverfahren in industrielle Entwurfsprozesse für eingebettete Echtzeitsysteme ist der Schwerpunkt der Ausführungen in Kapitel 6. Hier wird zunächst die generelle Methodologie eines Komponenten-basierten Entwicklungsprozesses beschrieben, der im Kontext dieser Arbeit vorgeschlagen wird. Darauf aufbauend wird der Begriff der inkrementellen Integration erläutert und damit zusammenhängende Echtzeit-Schnittstellen für Komponenten abgeleitet. Syntheseverfahren für diese Schnittstellen werden angegeben und mit anderen bekannten Verfahren verglichen. Es folgen Messungen für die inkrementelle Integration anhand zweier Beispielssysteme. Zwecks Integration des Platzierungsprozesses in den Gesamtzusammenhang eines durchgängigen Entwicklungsprozesses werden zum Einen beispielhaft mehrfach-parametrisierte Optimierungen mit Hilfe des in Kapitel 5 vorgestellten Verfahrens gezeigt. Zum Anderen erfolgt eine Diskussion über virtuelle Integration mit einem Beispiel zur Platzierungsoptimierung unter Berücksichtigung variabler Echtzeitattribute, die motiviert, in welcher Art und Weise die in dieser Arbeit vorgestellte Methodik zur Exploration des Entwurfsraumes, womöglich auch in frühen Phasen des Designs, eingesetzt werden könnte.

Kapitel 7 greift die in Kapitel 4 formalisierte Analysetechnik zur Bestimmung des Zeitverbrauches nochmals auf und erweitert dies durch Berücksichtigung von Störungen durch die Laufzeitumgebungen. An einem Beispiel wird eindrucksvoll deutlich, welche Größenordnungen dieser Einfluss annehmen kann und es wird motiviert an welchen Stellen durch welche Technik hier signifikante Verbesserungen möglich werden.
Eine dieser Techniken ist die Verwendung von Multithreaded Architekturen in den Prozessorknoten eines verteilten eingebetteten Systems und dementsprechend wird diese Technik in Kapitel 8 generell vorgestellt und ergänzt durch die Beschreibung einer im Rahmen dieser Arbeit entstandenen Prozessorimplementierung. Die Vorteile dieser Architektur werden anschließend in die Echtzeit-Analyse-Methodik integriert und es werden verschiedene Varianten der Optimierung der Ausnutzung dieser Technik diskutiert sowie umfangreich evaluiert. Abschließend erfolgt eine Integration der adaptierten Analyseverfahren in das in Kapitel 5 vorgestellte Analyse- und Optimierungsverfahren.
Kapitel 9 verfeinert die Technik des Multithreading auf speziell für eingebettete Echtzeitsysteme zugeschnittene Varianten, die aus der Beobachtung typischer Ausführungen heraus motiviert sind und evaluiert diese Verfeinerung.
Eine Zusammenfassung der Ergebnisse dieser Arbeit ist schließlich in Kapitel 10 zu finden. In den diversen Anhängen finden sich nützliche Daten zu den verwendeten Beispielapplikationen und der entstandenen Werkzeugkette, sowie andere, generelle Informationen.

Kapitel 2

Analyse der Echtzeiteigenschaften

2.1 Zeitanforderungen und Nebenläufigkeit

Komplexe eingebettete Systeme beinhalten, wie in Kapitel 1.4 schon dargestellt, neben den funktionalen Anforderungen zusätzlich diverse nichtfunktionale Anforderungen, die ein erwartetes zeitliches Verhalten des eingebetteten Systems in der Interaktion mit einer physikalischen Umwelt sicherstellen sollen. Geht man von einer formalen Anforderungsdefinition, wie sie im Systems- und Software-Engineering Bereich in vielen Entwicklungsprozessen vorgeschrieben ist, aus, so wird man zunächst eine große Sammlung an Anforderungen finden, die implizite oder explizite Zeitanforderungen enthalten. Die folgenden vier Anforderungen sollen beispielhaft darstellen, welche Notwendigkeiten des zeitlichen Verhaltens in der Praxis auftreten können und wie aus ihnen die typischen Echtzeitkonstrukte abgeleitet werden können. Gleichzeitig dienen sie der Einführung der üblicherweise verwendeten Nomenklatur und der generellen Eigenschaften von Echtzeitsystemen. Als Beispielapplikation für ein komplexes eingebettetes System soll hier ein fiktiver, stark vereinfachter Ausschnitt von Service- und Sicherheitseigenschaften in der Kfz-Elektronik betrachtet werden.

1. spätestens $5ms$ nach detektieren eines Front-Crashes sollen die Zündkapseln am Fahrer- und Beifahrer-Airbag das Signal zur Zündung erhalten

2. In festen Abschnitten von jeweils 100 ± 10 Umdrehungen pro Minute soll der Einspritzzeitpunkt für Kraftstoff in den Zylinder gemäß einem gegebenen Kennfeld aktualisiert werden.

3. Befindet sich das Fahrzeug im Tempomat-Zustand, soll der angegebene Geschwindigkeitswert mit einer maximalen Variation von 3% eingehalten werden.

4. Alle $0.5s$ sollte die elektronische Geschwindigkeitsanzeige mit einer maximalen Variation von 5% aktualisiert werden

Diese vier zunächst einmal unabhängigen Anforderungen schreiben ein komplexes Regelverhalten innerhalb der Elektronik des Fahrzeugs vor, das zwar an manchen Stellen durchaus exaktes zeitliches Verhalten fordert, im allgemeinen aber nur implizit die Notwendigkeit der Betrachtung von zeitlichem Verhalten ersichtlich macht. Als Beispiel für eine explizite Zeitanforderung kann die Anforderung 1. angesehen

werden, in der ein Zeitintervall von $5ms$ festgelegt wird, in dem die Fahrzeugelektronik in einer bestimmten Art und Weise auf Veränderungen in der physikalischen Umwelt reagieren **muss**. Eine weitere explizite Zeitanforderung ist in Anforderung 4. zu finden. Allerdings hat diese Anforderung qualitativ einen anderen Grad als Anforderung 1., da durch das Verletzen der Anforderung kein nennenswerter Schaden aus Sicht eines Nutzers/Betreibers entsteht. Dies führt zu der klassischen Kategorisierung der Echtzeitanforderungen in zwei Bereiche:

Hard Real-Time bezeichnet zeitliche Anforderungen, deren Verletzung einem System oder deren Nutzer beträchtlichen Schaden zufügen kann. Hierzu zählen Dinge wie Material-, Umwelt- oder Personenschäden ebenso wie das zur Last fallen oder eine steigende Unerträglichkeit eines Produktes, die letztlich die Marktposition schwächen kann.

Soft Real-Time bezeichnet zeitliche Anforderungen, deren Verletzung erträgliche oder kaum bemerkbare Schäden hervorrufen. Hierzu gehören üblicherweise Dinge wie eingeschränkter Bedienkomfort in kurzen Phasen oder zeitweiliger Ausfall nicht sicherheitsrelevanter Funktionalitäten.

Anforderung 4. ist eine typische Anforderung aus dem Bereich der Soft Real-Time-Kategorie, da eine Verletzung der vorgesehenen Zeitschranke vom Bediener des Fahrzeugs üblicherweise gar nicht wahrgenommen wird und weitere Folgefehler nicht vorhanden sind. Anforderung 1. hingegen ist aufgrund der Schwere der Schäden bei einer Verletzung der Zeitbedingen der Kategorie der Hard Real-Time-Systeme zuzuordnen.
In den Anforderungen 1. und 4. lassen sich zudem 3 grundlegende Eigenschaften von Echtzeitsystemen erkennen: Deadlines, Aktivierungsereignisse und Perioden:

Aktivierungsereignis ist eine Änderung des beobachtbaren Zustandsraumes, von dem ausgehend eine bestimmte, dem Zustandswechsel zugeordnete, Reaktion des eingebetteten Systemes erwartet wird. Der Zeitpunkt des Auftretens eines solchen Aktivierungsereignisses wird **Aktivierungszeitpunkt** genannt.

Deadline bezeichnet eine Zeitspanne, die eine Reaktion auf ein Aktivierungsereignis maximal dauern darf (Hard Real-Time Systeme) oder sollte (Soft Real-Time Systeme).

Periode bezeichnet einen konstanten und regelmäßigen Abstand zwischen wiederkehrenden Aktivierungszeitpunkten eines Aktivierungsereignisses.

Aktivierungszeitpunkte entsprechen Beobachtungszeitpunkten, an denen eine Änderung der physikalischen Umgebung wahrgenommen wird, auf die gemäß der Anforderungsdefinition eine Reaktion erwartet wird. Dieser Zeitpunkt muss nicht dem wirklichen Zeitpunkt einer Veränderung der Umgebungsparameter entsprechen, sondern wird oftmals durch Vorverarbeitungsintervalle verzögert, wie beispielsweise Sensorabfragen durch Polling-Methoden oder Filterung von Schwellwerten aus einem eher kontinuierlichen Datenstrom. In der oben angegebenen Anforderungsdefinition ist

beispielsweise das Erkennen eines Crashes in Anforderung 1. ein solches Aktivierungsereignis[1].
Deadlines bezeichnen maximale Reaktionszeiten, die ein eingebettetes Echtzeitsystem einhalten muss und sind die gebräuchlichste Form, Echtzeitverhalten explizit zu spezifizieren (siehe Anforderung 1. und 4. im obigen Beispiel).
Perioden von Aktivierungsereignissen können unterschiedliche Quellen haben. Zum einen werden sie durch Anforderungen festgelegt, beispielsweise die Aktualisierungsperiode der Geschwindigkeitsanzeige in Anforderung 4. des obigen Beispiels. Zum anderen können sie in weiteren Verfeinerungen des Designs entstehen und vor allem durch sogenannte Timer[2] realisiert werden. Beispielsweise muss zur Beobachtbarkeit physikalischer Parameter der Umgebung ein dafür zuständiger Sensor in einer bestimmten Frequenz abgefragt werden, um verlässliche Daten produzieren zu können.

Implizite Zeitanforderungen ergeben sich aufgrund des hohen Abstraktionsgrades der Anforderungsdefinition. So ist beispielsweise in Anforderung 2. keine zeitbezogene Bedingung formuliert, allerdings wird implizit gefordert, die Umdrehungszahl des Motors zu ermitteln. Werden im Zuge eines Entwurfsprozesses diese Anforderungen weiter verfeinert, so ergeben sich daraus auch durchaus zeitliche Anforderungen, da die geforderte maximale Varianz von 10% einen gewissen Grad an Genauigkeit der Messung erfordert. In diesem konkreten Szenario beispielsweise soll die Drehzahl des Motors ermittelt werden. Dies wird üblichweise durch regelmäßige Einkerbungen in der Nockenwelle realisiert. Das Detektieren einer solchen Einkerbung durch eine Lichtschranke erzeugt ein Aktivierungsereignis, das z.B. einen Zähler ansteuert und nachgeschaltet eine Berechnung der Umdrehungszahl initiiert. Es können folglich aus einer relativ abstrakten Anforderungsdefinition durchaus im Laufe weiterer Verfeinerungen auch weitere Aktivierungsereignisse erzeugt werden.
Dieses Beispiel verdeutlicht zudem eine weitere fundamentale Eigenschaft von Echtzeitsystemen, die sogenannten Aktivierungsschemata: Zwar sind die Einkerbungen an der Nockenwelle regelmäßig, aber die Dauer zwischen zwei Aktivierungszeitpunkten des Ereignisses "Nockenwelle-Einkerbung" ist nicht konstant, sondern hängt direkt von der Umdrehungsgeschwindigkeit ab. Aktivierungsschemata eines eingebetteten Echtzeitsystems sind folgendermaßen definiert:

Periodische Systeme bezeichnen Systeme, die von Aktivierungsereignissen angesteuert werden, die eine eindeutige und konstante Periode besitzen.

Sporadische Systeme bezeichnen Systeme, die von Aktivierungsereignissen angesteuert werden, deren Auftrittdistanz nicht über eine konstante Periode verfügen, die aber üblicherweise einen minimalen Abstand zwischen zwei Aktivie-

[1] Man beachte, dass in der Anforderungsdefinition 1. nur der Zeitpunkt der *Detektion* des Craches betrachtet wird. Die Anforderung der erlaubten Zeitspanne vom physikalischen Crash bis zum Zeitpunkt seiner Beobachtung für das elektronische Steuerungssystem muss hingegen zusätzlich spezifiziert werden.

[2] Timer im Kontext dieser Arbeit bezeichnen Uhren, die zu festgelegten Zeitpunkten ein Aktivierungsereignis hervorrufen und sich dann auf 0 zurücksetzen, um wieder von vorne zählen zu können. Sie sind üblicherweise Teil der Hardware eines eingebetteten Systems.

rungszeitpunkten aufweisen. Dieser minimale Abstand wird **minimale Zwischenankunftzeit** genannt.

Burst Systeme bezeichnen Systeme, die innerhalb eines festgelegten Zeitintervalls durch eine maximale Anzahl an Aktivierungsereignissen angesteuert werden. Die Größe dieses Zeitintervalls wird äußere Periode genannt und kann sich sowohl sporadisch als auch periodisch verhalten.

Das dargestellte Verfahren zur Ermittlung der Umdrehungszahl eines Motors ist folglich ein sporadisches System, dessen minimale Zwischenankunftszeit durch die physikalischen Randbedingungen definiert wird: Ein Motor hat aufgrund seiner materialwissenschaftlichen und physikalischen Parameter eine maximale Umdrehungszahl. Umdrehungsgeschwindigkeiten oberhalb dieser Grenze führen automatisch zur Zerstörung des Motors oder sind physikalisch nicht zu erreichen. Demzufolge determiniert die maximal erreichbare Umdrehungsgeschwindigkeit die maximale Frequenz (und damit die minimale Periode) des Aktiverungsereignisses "Nockenwelle-Einkerbung".
Burst Systeme hingegen weisen innerhalb eines Zeitintervalls (der äußeren Periode) keine Beziehung der in diesem Zeitfenster auftretenden Ereignisse auf. Ein Beispiel für derartige Systeme ist die Orts- und Gegenstandserkennung im Sichtbereich eines Fahrzeugs durch Ultraschall: Zu festen Zeitpunkten werden Schallsignale ausgesendet. Ein Sensor empfängt über einen fest definierten Zeitraum (entspricht hier der äußeren Periode) die durch die Umgebung verursachten Reflektionen. Die zeitlichen Abstände der einzelnen Reflektionen an Gegenständen hängen von deren Entfernung ab und weisen somit so gut wie keine Beziehung zueinander auf. Aufgrund von Abschwächungen des Schallpegels sind in einer fest definierten Umgebung nur Gegenstände bis zu einer gewissen Entfernung zu beobachten und damit kann die Anzahl der Reflektionen innerhalb der äußeren Periode beschränkt werden.
Eine weitere Quelle von Echtzeiteigenschaften lassen sich in Anforderung 3. ausmachen: Das Halten einer bestimmten Geschwindigkeit entspricht i.a. einem Regelkreis, der durch einen dem Problem angemessenen Regelungsalgorithmus auf seinem Soll-Wert gehalten wird. Entsprechend der notwendigen Genauigkeit ergibt eine Analyse des Regelverhaltens im Zusammenspiel mit einem mathematischen Modell der Umgebung eine minimale Frequenz, in der der Regelkreis die Stellgrößen von Aktuatoren abhängig von der Beobachtung der Umgebung verändert. Folglich wird hier durch eine Analyse des Verhaltens ein periodisches Teilsystem spezifiziert, obwohl dies so explizit nicht in den Anforderungen auftaucht.
Diese dargestellten Ableitungen von Zeitanforderungen aus einem Anforderungskatalog an das Gesamtsystem sind natürlich nur ein exemplarischer Ausschnitt von in einem Entwurfsprozess auftretenden neuen Zeitbedingungen. Gleichwohl sind an ihnen eine Reihe fundamentaler Echtzeiteigenschaften abzulesen, wie oben bereits erläutert. Eine weitere Eigenschaft, die nicht unmittelbar eine Echtzeiteigenschaft ist, wird an Anforderung 3. zusätzlich ersichtlich: Das Vorhandensein verschiedener sogenannter **Modi**. Ein Modus ist ein Zustand des Systems, in dem bestimmte funktionale Anteile zur Geltung kommen müssen, andere hingegen womöglich nicht. Am Beispiel des Tempomaten ist dies eindeutig ersichtlich, da nur im Modus "Tempomat-Betrieb" die Regelung der Geschwindigkeit auf einen konstanten Wert durchgeführt

werden muss, nicht jedoch in anderen Modi. In den folgenden Kapiteln, wenn wir über Nebenläufige Systeme, Echtzeitanalysen und Analysen von Kommunikationslatenzen argumentieren, wird deutlich werden, dass sowohl die unterschiedlichen Modi als auch die Wechsel zwischen Modi einen nicht unerheblichen Einfluss auf die Echtzeiteigenschaften des Gesamtsystems haben. Allerdings steht die Analyse von Modi und deren Umschaltverhalten nicht im Fokus dieser Arbeit und es sei stattdessen auf die Arbeiten in [149, 147, 185, 173] verwiesen.

Unterzieht man die Anforderungsdefinitionen 1. bis 4. auf Seite 15 einer genaueren Betrachtung bzgl. ihres funktionalen Anteiles, so wird sehr schnell deutlich, dass zumindest Teile der Anforderungen funktional in keinem Zusammenhang zueinander stehen. Der Versuch, alle Anforderungen innerhalb eines monolithischen Programmes zu implementieren, scheitert an den Nicht-funktionalen Anteilen, der Wart- und Erweiterbarkeit (vgl. hierzu [115]), womöglich bestehende Einschränkungen der räumlichen Geometrie eines Systems und nicht zuletzt an den Auflagen von adequaten Entwurfsprozessen. Es besteht daher die Notwendigkeit, die verschiedenen Funktionen nebenläufig oder parallel zu implementieren, wobei die Parallelität hier nicht einer echten Parallelität entsprechen muss. Dazu werden typische Anteile des funktionalen Verhaltens in kleinere, logisch zusammenhängende, Fragmente aufgeteilt, für die es letztlich eine ausführbare Implementierung gibt. Diese Fragmente werden **Tasks** genannt. Die Generierung dieser Tasks aus einer Modellierung des Systems ist eine defizile Arbeit, die zwar durch automatische Methodiken unterstützt werden kann (Beispiele hierzu sind im Bereich des HW/SW-Codesigns zu finden), die jedoch oft ein detailliertes Expertenwissen voraussetzt. Insgesamt muss das Bemühen hierbei sein, eine optimale Abbildung auf Subfunktionen zu erzeugen, die den verschiedenen Dimensionen des Entwurfsraumes, also Anforderungen an die räumlichen Geometrie, Anforderungen an die Leistungsfähigkeit von Steuergeräten, Anforderungen an das zeitliche Verhalten, Anforderungen an das Kommunikationsverhalten, gemeinsames Nutzen von gleichen Funktionalitäten, und dergleichen, genügt. Im Kontext dieser Arbeit wird davon ausgegangen, dass ein solches Verfahren existiert und aus einer Modellierung des Systems (z.B. im Sinne eines modellbasierten Entwicklungsprozesses) ein entsprechendes Tasksystem erstellt wird. Es soll dabei aber nicht explizit ausgeschlossen werden, dass die in den folgenden Kapiteln dargestellte Synthese des Echtzeitsystems ein Teil der Iterationen eines solchen übergeordneten Verfahrens darstellt. In den späteren Kapiteln, insbesondere bei der Integration der Verfahren dieser Arbeit in einen Entwurfsprozess, werden wir dieses Thema nochmals kurz ansprechen, es allerdings nicht in den Mittelpunkt der hier entworfenen Methodiken stellen, sondern nur motivieren, auf welche Art und Weise eine Integration in die Durchmusterung des Entwurfsraumes auf dieser Ebene möglich wäre.
Parallelität von Tasks, wie sie oben motiviert wurde, bedeutet nicht zwingenderweise, dass die Tasks wirklich zeitlich parallel ausgeführt werden müssen. Dies kann z.B. nicht der Fall sein, wenn zwei funktional unabhängige Tasks auf dem gleichen Steuergerät ausgeführt werden, dies jedoch über nur einen Prozessor verfügt, sie sich also die Resource Prozessor teilen müssen. Da der allgemeine Sprachgebrauch von Parallelität aber diese Art der Parallelität nicht umfasst, soll stattdessen der

Terminus **Nebenläufigkeit** hierfür verwendet werden und ausdrücken, dass Tasks nebenläufig sind, wenn sie funktional unabhängig von einander laufen können und nicht in einer sequentialisierten Reihenfolge ausgeführt werden müssen. Dies umfasst echte Parallelität ebenso wie das Teilen einer gemeinsamen Prozessorresource. Letzteres bedingt, das die Ausführungszeitpunkte der Tasks auf der gemeinsamen Laufzeitplattform so geplant werden, dass alle Anforderungen, insbesondere die zeitlichen, eingehalten werden können. Einen solchen Ablaufplan nennt man **Schedule**. Es gibt verschiedene Verfahren, Schedules zu erzeugen, wobei jeder Ansatz grundsätzlich unterschiedliche Strategien verwendet. Diese Verfahren werden im folgenden **Schedulingverfahren** benannt und sollen in Kapitel 2.2 eingehender betrachtet werden. Ist ein Schedule gefunden, so muss überprüft werden, ob unter diesem Schedule alle Tasks ihre Teilanforderungen einhalten können und somit die Korrektheit des Gesamtsystems garantiert werden kann. Schedules, die dies sicherstellen, werden **feasible** genannt und sind das Ziel jeder Systemsynthese und -optimierung innerhalb dieser Arbeit.

2.2 Schedulingverfahren

Die Aufgabe des Schedulingverfahrens ist es, einen Schedule für einen Satz an Tasks zu finden, die gemeinsam aber nebenläufig auf einem Prozessor abzuarbeiten sind. Dabei kann zunächst zwischen dynamischen und statischen Schedulingverfahren unterschieden werden.

Statisches Scheduling bedeutet, dass es einen festen Zeitplan gibt, in dem für jede Task eindeutig determiniert ist, wann sie zur Ausführung gelangt. Dieser Zeitplan wird zur Systemerstellungszeit einmalig bestimmt und ist anschließend während der Laufzeit unabänderlich. Tasks in statischen Schedules werden auch **Zeit-basiert (time-triggered)** genannt[1].

Dynamisches Scheduling bedeutet, dass zur Laufzeit abhängig von den Aktivierungsereignissen entschieden wird, welche Task zu welchem Zeitpunkt aktiviert wird. Tasks in dynamischen Schedules werden auch **Ereignis-basiert (event-triggered)** genannt.

Statisches Scheduling alleine ist nicht gleichzusetzen mit Tasks einer konstanten Periode. Letztere können sowohl in statischen als auch in dynamischen Schedulingverfahren eingesetzt werden. Vielmehr sind in statischen Schedulingverfahren nicht nur die Perioden von Tasks konstant, sondern auch die Abstände der Taskaktivierung unterschiedlicher Tasks weisen eine konstante Zeitdifferenz zueinander aus. Letzteres kann durch die physikalische Umgebung bedingt sein, kann aber auch künstlich durch den statischen Schedule eingefügt worden sein. Letztlich abstrahiert statisches Scheduling von Aktivierungsereignissen und reduziert diese auf Timer-getriebene Aktivierungsereignisse für alle Tasks unter diesem Schedulingverfahren.

[1] Im folgenden wird der Begriff zeit-basierte Tasks auch innerhalb von dynamischen Schedules verwendet und meint dann Tasks, die eine konstante Periode haben und deren Aktivierunsgereignis von einem Timer erzeugt wird.

Beide Schedulingverfahren haben Vor- und Nachteile. Während statisches Scheduling hochgradig vorhersagbar ist, weil die Zeitpunkte der Taskaktivierung stets eindeutig und fest bestimmbar sind, ist dies bei den dynamischen Verfahren nicht so eindeutig. Andererseits sind dynamische Systeme weitaus flexibler im Umgang mit sporadischen Systemen und können direkt auf ein Aktivierungsereignis reagieren, wohingegen bei statischen Systemen die Aktivierung eines Tasks aufgrund eines womöglich gerade verstrichenen Aktivierungszeitpunktes dieser Task lange verzögert werden kann. Die Flexibilität wird jedoch bezahlt durch ein notwendigerweise vorhandenes umfangreicheres Echtzeit-Betriebssystem, welches die Aktivierung (**dispatchen**) und Terminierung von Tasks regelt. Echtzeitbetriebssysteme (RTOS) für statische Schedules können deutlich einfacher gestaltet werden, da sie im Grunde nur abgelaufene Timer analysieren müssen und entsprechend der aktuellen Systemzeit einen Task aktivieren müssen. Typische Vertreter von Echtzeitbetriebssystemen, die statische Schedulingverfahren verwenden, wie etwa TTP/C[189] oder OSEKtime[134] sind im industriellen Einsatz erprobt oder aber stehen vor ihrer Einführung. Wir werden später in Teil II die Einflüsse eines RTOS auf das zeitliche Verhalten von Tasks genauer quantifizieren.
Eine weitere wesentliche Eigenschaft eines Schedulingverfahrens wird durch die sogenannte **Preemptivität** gekennzeichnet: Soll eine Task durch das entsprechende Schedulingverfahren aktiviert werden, läuft aber gerade eine andere Task, so kann man entweder die Aktivierung der neuen Tasks nach hinten schieben (sie erfährt dann eine sogenannte **Blockierung**) oder aber man erlaubt eine Unterbrechung und supendiert die gerade laufende Task, um sie später, sobald der Prozessor wieder frei ist, zu reaktivieren. Problematisch bei preemptivem Scheduling sind Tasks, die aufgrund von Zugriffen auf kritische Bereiche (z.B. über Semaphoren gekapselte Codesegmente) für bestimmte Zeitspannen nicht unterbrechbar sind. In einem solchen Fall wird trotz der Preemptivität eine Blockierung erzeugt, die den Dispatchzeitpunkt einer zu aktivierenden Task zeitlich verzögert. Da derartige Effekte dem Paradigma der Vorhersagbarkeit in statischen Schedulingverfahren widersprechen, verbieten viele statischen Echtzeitbetriebssysteme, die Verwendung von preemptivem Scheduling. Stattdessen muss der konstante Schedule so gewählt werden, dass zu jedem Aktivierungszeitpunkt einer Task alle anderen Tasks terminiert sind. In dynamischen Schedulingverfahren wird hingegen üblicherweise auf preemptives Scheduling vermehrt Wert gelegt, um zeitkritische Tasks nicht unnötigerweise an der Aktivierung zu hindern.

Dynamische Schedulingverfahren haben im Gegensatz zu den statischen Verfahren zunächst keine Einschränkung, wann welche Task zur Ausführung gelangt. Gleichwohl sollen natürlich die in den Anforderungsdefinitionen geforderten zeitlichen Bedingungen eingehalten werden. Die Unterbrechung einer Task, die innerhalb einer $5ms$ Zeitspanne die Zündkaspseln eines Airbags ansteuern muss, durch eine Task, die $50ms$ verbraucht um nicht sicherheitskritische Berechnungen durchzuführen, ist in diesem Zusammenhang sinnlos. Vielmehr sollte die Preemption von Tasks durch andere Tasks anhand deren Wichtigkeit zueinander eingeschränkt werden, also eine Priorisierung der Tasks bedingen. Entsprechend dem jeweiligen Anwendungsszenario einer Echtzeitapplikation gibt es daher eine ganze Reihe von sehr unterschiedlichen

Methoden, Taskaktivierungen und Prioritäten der Tasks zueinander zu definieren. Generell wird hier zwischen sogenanntem **Offline**- und **Online**-Scheduling unterschieden. Die Namensgebung entstammt dem Zeitpunkt, wann Prioritäten für die Tasks erzeugt werden. In Offline-Verfahren werden die Prioritäten bei der Systemerstellung generiert und sind während der Laufzeit als konstant zu betrachten. Online-Verfahren hingegen berechnen Prioritäten während der Laufzeit zum Aktivierungszeitpunkt einer Task. Bekannte Vertreter von Online-Verfahren z.B. sind Proportional-Share-Scheduling[174] und Earliest-Deadline-First[104]. Ersteres wird nur sehr selten eingesetzt und dient vornehmlich einer möglichst gleichmäßigen Aufteilung von Zeitintervallen auf alle aktiven Tasks unter Berücksichtigung der jeweils durch eine Task erzeugte Last. Aufgrund der industriellen Bedeutungslosigkeit dieses Scheduling-Verfahrens sei hier für weitergehende Betrachtungen auf [174] und [115] verwiesen.
Earliest-Deadline-First (EDF) hingegen bestimmt die Priorität einer Task, indem es für alle aktiven Tasks die noch bis zur jeweiligen Deadline verfügbare Zeit vergleicht und diejenige Task am höchsten priorisiert, deren Deadline als nächstes abzulaufen droht. Diese Art der Prioritätsbestimmung erlaubt eine optimale Ausnutzung der Prozessorresourcen, wie schon in [104] gezeigt werden konnte. Auf der anderen Seite ist aber ein erhöhter Aufwand innerhalb der Implementierung des Schedulers eines RTOS notwendig, um diese Operationen durchführen zu können. Auch hier sei auf Teil II dieser Arbeit verwiesen, in dem quantifiziert wird, inwieweit höhere Kosten für das Scheduling sich auf die Echtzeitfähigkeit von Tasksystemen auswirkt. Ein weiterer wesentlicher Nachteil des EDF Verfahrens ist darin zu sehen, dass sehr häufig Tasks suspendiert und andere aktiviert werden, obwohl dies zur Einhaltung aller Deadlines nicht notwendig wäre[170]. Diese häufige Wechseltätigkeit bei der Ausführung von Tasks verschlechtert die Performanz aufgrund der Kosten, die ein Suspendieren und Dispatchen einer Task verursacht. Beide Nachteile zusammen liefern den Grund, warum auch EDF-basierte Verfahren kaum Verbreitung in industriellen Anwendungen finden und wir werden die Online-Schedulingverfahren im Kontext dieser Arbeit nicht weiter verwenden und uns stattdessen auf Offline-Verfahren konzentrieren.

Offline-Verfahren legen, wie oben beschrieben, bei Systemerstellung bereits die Prioritäten der Tasks fest. Bekanntestes Verfahren aus der Reihe der Offline-Verfahren sind Ratenmonotone[104] und Deadline-Monotone Prioritätenzuweisung[100]. Ersteres räumt den Tasks mit der kleinsten Periode die höchste Priorität ein und stuft alle anderen Tasks entsprechend der Längen ihrer Perioden ein. Vorbedingung ist hierbei, dass die Deadlines aller Tasks gleich ihrer Periode sind. Ist dieses nicht der Fall, so kommt Deadline-Monotone Prioritätenvergabe zum Einsatz. Hierbei werden die Tasks nicht hinsichtlich der Periode sondern bezüglich der angegebenen Deadline geordnet. Es kann gezeigt werden, dass ratenmonotones Scheduling unter den für diese Prioritätenvergabe notwendigen Bedingungen optimal ist[104]. Für deadlinemonotones Scheduling kann unter der Voraussetzung, dass die Deadlines grundsätzlich kleiner oder gleich der Periode der Tasks sind, ebenfalls Optimalität nachgewiesen werden[12]. Liegt die Deadline hingegen jenseits der Periode, ist dieses Verfahren nicht mehr optimal[97]. Trotzdem ist keine bessere Heuristik bekannt,

die Prioritäten solcher Tasksysteme in einem Offline-Verfahren zu bestimmen. Da zudem Tasksysteme mit Deadlines jenseits der Periode eher selten in eingebetteten Systemen anzutreffen sind, hat sich dieses Verfahren durchgesetzt und genießt eine weit verbreitete Popularität im industriellen Einsatz für dynamische Schedulingverfahren. Die Vergabe von Prioritäten nach dem ratenmonotonen Schema ist zudem nur ein Spezialfall der deadlinemonotonen Vergabe, daher wird im Kontext dieser Arbeit grundsätzlich von einer Prioritätsvergabe nach letzterem ausgegangen.

Zusammenfassend wird ersichtlich, dass die wesentliche Mehrheit aller eingebetteter Echtzeitsysteme entweder mittels statischen Schedulingverfahren oder aber unter Verwendung von dynamischen Verfahren mit der deadlinemonotonen Prioritätenvergabe implementiert werden. Zwar besteht ein Trend, dynamische Verfahren durch statische zu ersetzen, jedoch scheint der Hauptgrund für diesen Trend das mangelnde Verständnis der Vorhersagbarkeit von dynamischen Systemen zu sein. Dies trifft insbesondere auf die Überragung der dynamischen und statischen Konzepte auf Kommunikationsmedien in verteilten Systemen zu (Aussagen in [121] und [17] stützen diese Einschätzung explizit). Die hier vorliegende Arbeit soll einen Beitrag dazu leisten, die Vorhersagbarkeit und Handhabbarkeit von dynamischen Verfahren in Analyse und Synthese zu verbessern. Letztlich haben beide Verfahren ihre jeweiligen Stärken und Schwächen, so dass im Systementwurf eine solche Kombination aus statischen und dynamischen Schedulingverfahren gewählt werden kann, in der beide ihre jeweiligen Stärken nutzen und die entstehende Implementierung möglichst optimal ist. Wir werden uns daher in dieser Arbeit auf die Betrachtung von dynamischen Verfahren beschränken, jedoch am Beispiel der Kommunikation in verteilten eingebetteten Echtzeitsystemen deutlich machen, wie eine Kombination der beiden Verfahren gelingen kann. Alle übrigen hier im folgenden vorgestellten Methodiken lassen sich mehr oder weniger einfach auf statische Verfahren übertragen. Dies gilt insbesondere auch für die quantitativen Analysen von Echtzeitsystemen, wie sie im folgenden Abschnitt dargestellt werden, auch wenn dies nicht explizit demonstriert wird.

2.3 Schedulinganalysen

Unter Schedulinganalysen versteht man im allgemeinen Sprachgebrauch sowohl eine qualitative als auch eine quantitative Analyse des Echtzeitverhaltens von Tasks eines eingebetteten Echtzeitsystems. Qualitative Analysen überprüfen, ob ein gegebenes System feasible ist, während quantitative Analysen die schlimmstenfalls zu erwartenden Berechnungszeiten von Tasksystemen mit ihren Deadlines vergleicht. In diesem Sinne sind quantitative Analysen zugleich auch qualitative. Letztere beruhen oftmals auf Abstraktionen und sind üblicherweise nur notwendige oder hinreichende Bedingungen, aber zumeist nicht beides gleichzeitig. Im folgenden sollen die wesentlichen Meilensteine in der Entwicklung von Schedulinganalysen dargestellt werden. Gleichzeitig legen diese Arbeiten den Grundstein zu dem im Rahmen dieser Arbeit verwendeten Formalismus. Aufgrund der Vielzahl an Publikationen der letzten 25 Jahre im Bereich der Schedulinganalysen kann dies hier nur einen groben Überblick geben. Für eine genauere, chronologisch geordnete Darstellung sei an dieser Stelle

auf [162] verwiesen; für eine mehr anwendungsorientierte Sicht sei hier [90] empfohlen.

Die sicherlich meist zitierte Arbeit stammt aus den frühen siebziger Jahren von Liu und Layland[104], in der erstmals das Prinzip des ratenmonotonen, preemptiven Scheduling unter Verwendung von festen Prioritäten vorgestellt wurde. Gleichzeitig geben die Autoren eine Analysemethodik an, mit der sich in gewissem Rahmen die Feasibility eines Tasksystems fesstellen lässt. Eine der wesentlichen Aussage dieses Artikels ist, dass für ein System von unabhängigen Tasks mit Deadlines gleich der Periode, die auf einem Prozessor angesiedelt sind und mit Fixed Priority Scheduling dispatched werden, die Feasibility gegeben ist, wenn dieses System die Bedingung

$$\sum_{i=1}^{n} \frac{c_i}{t_i} \leq n \left(2^{\frac{1}{n}} - 1\right) \tag{2.1}$$

erfüllt und die Prioritäten der Tasks gemäß ihrer Auftrittsrate (also dem Kehrwert der Periode) festgelegt sind, wobei c_i die obere Grenze der Laufzeit der Task τ_i, t_i die Periode von τ_i, und n die Anzahl der Tasks ist. Führt man eine Grenzwertbetrachtung dieser sogenannten Utilisationbound durch, so nähert sich der Faktor $n\left(2^{1/n} - 1\right)$ dem Wert des natürlichen Logarithmus von 2 an; mit $ln(2) \approx 0.6931$ ergibt sich die berühmte 69%-Grenze für ratenmonotones Scheduling. Unglücklicherweise ist Gleichung 2.1 nur eine hinreichende Bedingung, d.h. solange ein Tasksystem nach den Vorgaben von [104] eine Auslastung unterhalb dieser Grenze erzeugt, ist das System feasible. Wird hingegen eine Auslastung oberhalb dieses Wertes erreicht, so macht die Analyse nach 2.1 keine Aussage über die Feasibility des Systems. Erst die notwendige Bedingung

$$\sum_{i=1}^{n} \frac{c_i}{t_i} \leq 1.0$$

stellt eine obere Schranke für lösbare Tasksysteme zur Verfügung; Tasksysteme mit Werten zwischen diesen beiden Schranken können feasible sein, müssen es aber nicht. So konnte in [98] gezeigt werden, dass die reale beobachtete mittlere Utilisationbound für zufällig generierte große Tasksysteme etwa bei einer Auslastung von 88% liegt. Liu und Laylands fundamentale Aussage in [104], die maßgeblich die Arbeiten anderer Wissenschaftler im Bereich der Real-Time-Systeme beeinflusst hat, ist das Theorem der sogenannten Kritischen Instanz. Dieses Theorem besagt, dass die Antwortzeit von Tasks in Tasksystemen gemäß den Vorschriften aus [104] dann maximal ist, wenn alle Tasks zum gleichen Zeitpunkt aktiviert werden sollen. Eine Folge aus dieser Beobachtung ist übrigens die hinreichende Bedingung aus Gleichung 2.1.

Die Schwächen der in Gleichung 2.1 skizzierten Schedulinganalyse machen einen Einsatz in realistischen Systemen aber aufgrund der strengen Vorgaben zunichte. Neben der nur hinreichenden Natur der Bedingung sind insbesondere die Einschränkungen auf Deadlines gleich der Periode und die Vorgabe der Prioritätenvergabe zu streng für Echtzeitsysteme im industriellen Einsatz. Ab Mitte der achtziger Jahre bemühten sich daher zahlreiche Arbeitsgruppen darum, exaktere und in den Voraussetzungen allgemeinere Schedulinganalysen zu entwickeln. Basierend auf den Arbeiten in [98] wurden unabhängig in [83] und [9] Algorithmenklassen entwickelt,

die die Antwortzeit von Tasks unter Berücksichtigung von Preemptionen durch höher priorisierte Tasks quantifizieren konnten und dabei auf dem Theorem der kritischen Instanz basieren. Demnach kann die Antwortzeit einer Task durch das Lösen der folgenden rekursiven Gleichung berechnet werden:

$$R_i = c_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{t_j} \right\rceil c_j, \tag{2.2}$$

wobei $hp(\tau_i)$ die Menge derjenigen Tasks ist, deren Priorität über der Priorität der betrachteten Task τ_i liegt. Ein Vergleich der Antwortzeiten mit der jeweiligen Deadline einer Task liefert somit neben der generellen Aussage zur Feasibility des Systems auch noch Hinweise, wieviel Spielraum einer Task noch bleibt, beispielsweise zur Vergrößerung der reinen Laufzeit. Die Priorisierung der Tasks bei diesem Verfahren ist zunächst beliebig, allerdings erweisen sich bestimmte Strategien als vorteilhaft. So führten Leung und Whitehead in [100] die sogenannte Deadline-Monotone Prioritätenvergabe ein, die sich analog dem ratenmonotonen Verfahren von Liu und Layland[104] auf den Kehrwert der Deadline einer Task bezieht. In [12] konnte gezeigt werden, dass diese Strategie für Deadlines kleiner oder gleich der Periode optimal ist.

Die Theorie in Gleichung 2.2 ist in der Folge in vielfältigster Weise erweitert worden. So konnte beispielsweise Tindell in [184] eine erweiterte Antwortzeitanalyse herleiten, die auch für Tasksysteme gilt, in denen die Deadline einer Task ihre Periode überschreiten kann. Weitere, im Kontext dieser Arbeit interessante, Erweiterungen betreffen die Integration von Jitter (also einer gewissen Variabilität von Aufrufen der Tasks bei gegebener strikter Periode) durch Tindell in [181], sowie die Einbeziehung von Blockierungen durch die gleichzeitige Benutzung kritischer Bereiche[82]. Dabei wurde zur Vermeidung von Deadlocks (eine niedrig priorisierter Tasks blockiert einen kritischen Bereicht, den eine höher priorisierter Task benötigt) das sogenannte Priority Inheritance Protocol in [163] eingeführt, was allerdings nicht optimal ist und zur sogenannten Prioritäteninversion führt. Eine Erweiterung dieses Protokolls ist z.B. in [146] eingeführt worden und erzeugt nur steigende Prioritäten für die Zeit der Benutzung eines kritischen Bereiches, während in der übrigen Zeit die Task ihre eigentliche Priorität erhält. Dieses sogenannte Priority Ceiling Protocol führt dazu, dass eine Task nur garantiert einmal blockiert werden kann (vgl. [115]).
Insgesamt führen diese hier genannten Erweiterungen der Antwortzeitanalyse zu der für diese Arbeit fundamentalen Schedulinganalyse für Tasksysteme, in denen die Deadlines kleiner oder gleich der Perioden der Tasks sind:

$$R_i = c_i + B_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i + J_j}{t_j} \right\rceil c_j, \tag{2.3}$$

wobei B_i die maximale Blockierung einer Task τ_i und J_j der Jitter einer Task τ_j ist.

Gänzlich andere Wege gehen Arbeiten, die auf den sogenannten Event Streams[65] basieren. Event Streams stellen quasi Traces der Aufrufstruktur von Tasks dar und sind somit in der Lage, nahezu beliebige Aufrufe von Tasks zu behandeln, insbesondere auch nichtperiodische. [126] beispielsweise basiert auf Operatoren der Algebra

und führt eine qualitative Schedulinganalyse durch, in der die Aufrufstrukturen von Tasks in Event Streams Anforderungs- und Resource-Kurven betrachtet werden. Solange die Anforderungskurve niemals die Resource-Kurve überschreitet, gilt ein System als feasible. Mithilfe dieser Methodik sind zwar beliebige Aufrufstrukturen behandelbar, allerdings leitet sich aus der Analyse keine Antwortzeit von Tasks ab und entsprechen damit der Klasse der Schedulinganalysen, der auch die Arbeiten in [104] zugeordnet werden. Sind diese komplexen Aufrufstrukturen einfacher klassifizierbar, z.B. in aperiodische Tasks mit garantierten minimalen Zwischenankunftzeiten, so sind sie mit der klassischen Schedulinganalyse gemäß Gleichung 2.3 behandelbar. Eine weitere Klasse stellen die Burst-Aktivierungen dar (vgl. Kapitel 2.1). Auch für diese Art der unregelmäßigen Taskaktivierung existieren Mechanismen und quantitative Schedulinganalysen. Die fundamentale Idee hierbei ist es, diesen Tasks Ausführungszeitfenster innerhalb des normalen Schedules zur Verfügung zu stellen [169, 177, 16, 99] und sie somit der Schedulinganalyse gemäß Gleichung 2.3 zugänglich zu machen. Im Kontext dieser Arbeit wird jedoch auf eine detailliertere Betrachtung derartiger Systeme verzichtet (siehe auch Kapitel 4.1).

Die bisher vorgestellten Ansätze basieren sämtlichst auf der Annahme, dass Tasks unabhängig voneinander sind. Lediglich die Einführung von Blockierungen durch die parallele Nutzung von kritischen Resourcen gibt einen ersten Einblick in die möglichen Abhängigkeitsszenarien, die auftreten, sobald Tasks kausal voneinander abhängen. Betrachtet man einzelnen Ketten von Tasks, in denen Nachfolgertasks erst dann gestartet werden können, wenn die Vorgängertasks ihre Ausführung beendet haben, so können zwar weiterhin die bisherigen Analysen eingesetzt werden, sie stellen aber eine Überapproximation des zeitlichen Verhaltens dar. Dies liegt an der Behandlung von Preemptionen durch höher priorisierte Tasks: Ist eine höhere priorisierte Task mit der betrachteten Task in einer Taskkette, so kann sie unter Umständen überhaupt keine Interferenz mit dieser erzeugen. Dies führte zu der Einführung von Offsets für die Startzeitpunkte von Tasks, also einer Abänderung der klassischen kritischen Instanz um jene Zeitanteile, die die Abhängigkeit der potentiell unterbrechenden Task von der betrachteten Task ausmacht[8, 66]. Dazu sind allerdings die Offsets zu bestimmen, und Tindell hat in [181] gezeigt, dass dies bei der Verwendung mehrerer paralleler Taskketten im allgemeinen Fall nicht mehr berechenbar ist. Schwieriger wird dieses Unterfangen, wenn nicht nur von Taskketten ausgegangen wird, sondern von Taskbäumen mit beliebigen Verzweigungen [70], insbesondere auch Zyklen. In Kapitel 4.1 sowie detaillierter in Kapitel 4.5 werden wir auf dieses Problem zurückkommen.

Damit sind die grundlegenden Mechanismen der Analyse von Echtzeitsystemen, wie sie in dieser Arbeit verwendet werden, beschrieben worden und wir können im folgenden diese Analysen verwenden und verbessern. Bevor dies geschehen soll, muss aber zunächst noch erklärt werden, wie ein ganz entscheidender Parameter in den Analysen konstruiert werden kann: Die Laufzeit von Tasks.

2.4 Laufzeitanalysen

Ein wesentlicher Faktor bei der Analyse der Echtzeiteigenschaften von Tasksystemen, wie sie im vorhergehenden Abschnitt beschrieben sind, ist die Information über die Laufzeit c_i einer Task. Die dort verwendeten Scheduling-Analysen berechnen eine obere Grenze der Antwortzeit einer Task und benötigen somit die obere Grenze der möglichen Laufzeiten aller Tasks, die sogenannte WCET (Worst Case Execution Time). Das Gegenstück dazu, die BCET (Best Case Execution Time) wird hingegen verwendet, wenn der Jitter in verteilten Systemen betrachtet werden soll (vgl. Kapitel 4.4). Die Anforderungen an beide Zeiten ist, dass sie sicher sein sollen. Sicher bedeutet im Sinne der Scheduling-Analyse respektive der Jitter-Berechnung, dass es keine mögliche Laufzeit der Task gibt, die außerhalb des angegebenen Zeitintervalls liegt. Eine Verletzung dieser Anforderung würde eine Invalidierung der Berechnungen hervorrufen, was wiederum dazu führen kann, dass Tasks ihre Deadline nicht einhalten können. Auf der anderen Seite sollte die WCET/BCET möglichst akkurat sein, d.h. die Über- bzw. Unterapproximation sollte einen möglichst geringen Abstand zur realen WCET/BCET aufweisen. Da Laufzeiten von Tasks über die Ausführungen von Programmen auf konkreten Architekturen argumentieren, sind sie aufgrund der Unterschiedlichkeit der üblicherweise in eingebetteten Systemen eingesetzten Prozessortypen hochgradig architekturabhängig. Dies ist insbesondere in heterogenen verteilten Systemen zu beachten.
Es existieren mannigfaltige Ansätze zur Analyse der WCET/BCET. Nach [46] lassen sich alle Ansätze in 4 wesentliche Analyseschichten aufspalten:

- *High Level Analysen* untersuchen auf Hochsprachenebene, welche Pfade im Programm einer Task möglich sind und welche nicht.

- *Globale Low Level Analysen* bestimmen den Effekt globaler, prozessorabhängiger Faktoren, wie etwa Caches. Hierbei können die Informationen der High Level Analysen genutzt werden.

- *Lokale Low Level Analysen* betrachten prozessorabhängige Effekte, die auf wenige Instruktionen beschränkt werden können, beispielsweise Sätze von Assemblerinstruktionen in einer Pipeline. Es können hierbei die Ergebnisse der beiden übergeordneten Analysen verwendet werden.

- *Berechnung des WC* berechnet auf Basis der drei übergeordneten Analyse-Ergebnisse den schlimmsten anzunehmenden Pfad im Programm einer Task und liefert die WCET.

Nach einem vollkommen analogen Schema kann auch die BCET einer Task berechnet werden.

High Level Analysen lassen sich in zwei Gruppen klassifizieren: Auf der einen Seite Analysen mit manuellen Zusatz-Informationen (exklusive Pfade, Anzahl von Schleifendurchläufen, etc.), wie sie beispielsweise in [89] durch Programmannotationen oder in [138] mittels einer zusätzlichen speziellen Beschreibungssprache durchgeführt werden. Auf der anderen Seite existieren auch Ansätze, diese Informationen

automatisch oder semi-automatisch bereit zu stellen. Beispiele für derartige Ansätze sind in [5] und [28] unter Benutzung von symbolischer Ausführung oder [47] und [53] mit abstrakter Interpretation zu finden.

Globale Low Level Analysen dienen der Vorhersage des Zeitverhaltens aufgrund von Effekten, die sich über den ganzen Programmablauf erstrecken. Dies sind im wesentlichen Cache-Zugriffe, Branch Prediction und ähnliche, prozessorspezifische Effekte, so dass auf den jeweiligen Prozessor zugeschnittene Verfahren genutzt werden müssen. Üblicherweise werden bei den globalen Low Level Analysen klassifizierende Abstraktionen des Zeitverhaltens generiert, die anschließend in der lokalen Low Level Analyse berücksichtigt werden. Im Bereich der Cache-Zugriffs-Analysen existieren eine ganze Reihe von Arbeiten. In [68] werden mittels statischer Simulation Kategorien für den Zugriff auf den Instruction Cache generiert (*always miss, always hit, first miss*). [197] erweitert diese Betrachtung um Data Caches. Einen ähnlichen Weg, aber unter Benutzung von Abstrakter Interpretation gehen die Autoren in [54] und erweitern die Analyse in [53] um die Betrachtung von Unified Caches. Im Bereich der Instruction Cache Analyse haben wir in [117] gezeigt, dass die Verwendung von Techniken der formalen Verifikation genauere Vorhersagen des zeitlichen Verhaltens generieren kann. Im Bereich der Analyse von Branch Prediction ist beispielsweise die Arbeit in [34] zu nennen, in denen der Effekt von Branch Target Buffern in Intel Pentium Prozessoren analysiert wird. Da diese formalen Methodiken grundsätzlich auf eine sehr dedizierte Kenntnis der Architektur der zu analysierenden Prozessoren zurückgreifen müssen, diese aber im allgemeinen nicht oder nur unter großen Schwierigkeiten verfügbar ist, verwenden einige Ansätze messende Methoden. So werden in [140] Läufe des instrumentierten Programmes der Tasks gemessen und daraus das Cacheverhalten abgeleitet. [201] verwenden eine ähnliche Technik, allerdings unter Benutzung eines Simulators anstelle physikalischer Ausführung. Beide Verfahren haben allerdings den Nachteil, dass sie keine Garantie für Sicherheit der gefundenen WCET abgeben können.

Lokale Low Level Analysen beschäftigen sich mit der Vorhersage des Zeitverhaltens von Teilen des Programmes im Ablauf einer Pipeline. Aufgrund der vollkommen unterschiedlichen Architekturen der in eingebetteten Systemen verwendeten Prozessoren sind dies jeweils proprietäre Analyseverfahren. Beispiele für derartige Analysen sind in [103] für einen MIPS R3000 Prozessor, in [68] für eine MicroSPARC, in [35] für einen Pentium, in [53] für einen Coldfire oder in [140] für einen Athlon Prozessor zu finden.

Die letztlich notwendige Berechnungsphase hat die Aufgabe, alle Pfade des Programmes unter Berücksichtigung der durch die oberen drei Phasen erzeugten Informationen zu explorieren und den längsten Pfad zu liefern. Hier gibt es eine Reihe von Techniken, von denen sich die sogenannte Implicit Path Enumeration Technik (IPET)[101] der größten Beliebtheit erfreut. Die grundlegende Idee hierbei ist, den Programmfluss, ausgedrückt durch einen Flussgraphen mit sogenannten Basic Blocks (eine Sammlung zusammenhängender Instruktionen mit nur einem Sprungausgang und nur einem initialen Sprungeingang), mit den gewonnenen Zeiten der oberen Analysen zu annotieren und mittels einer Zählvariablen für die Anzahl der

Durchläufe durch jeden Block zu versehen. Optimale Optimierungsverfahren, wie beispielsweise Integer Linear Programming können dann genutzt werden, den längsten Pfad in einem Programm zu finden. Dieser Ansatz wird in einer ganzen Reihe der oben angegebenen Arbeiten verwendet.

Mithilfe der hier vorgestellten Techniken können nun auch in heterogenen Systemen die für die Schedulinganalysen notwendigen Parameter bestimmt werden. Allerdings sei an dieser Stelle noch darauf hingewiesen, dass die Bestimmung der WCET alleine mitunter nicht ausreichend ist, da sie i.a. eine isolierte Analyse ist, also den Einfluss von Unterbrechungen auf die Laufzeit einer Task (beispielsweise durch Invalidierung von Cacheinhalten der unterbrochenen Task) vernachlässigt. Wir werden dieses Thema in Teil II dieser Arbeit noch intensiver diskutieren und verweisen daher für eine ausführlichere Darstellung auf Kapitel 8.4.2.

Teil I

Automatische Platzierung im Entwicklungsprozess eingebetteter Systeme

Kapitel 3

Verteilte eingebettete Systeme

3.1 Komplexität und Vernetzung

Komplexe eingebettete Systeme, wie sie in Kapitel 1.2 beschrieben wurden, umfassen heutzutage einige hundert Funktionen oder Features, die in bis zu 80 und mehr Steuergeräten implementiert und vernetzt sind. Wie bereits erörtert, nimmt diese Zahl mit einem konstanten Wachstum jedes Jahr zu und erhöht die Komplexität der Systemintegration mit jedem neuen Steuergerät beträchtlich. Die Schwierigkeiten innerhalb der Systemintegration lokalisieren sich vornehmlich bei der zunehmenden Vernetzung der Funktionen: Im Gegensatz zum Funktionsumfang nimmt die Anzahl der auf die Umgebung einwirkenden Aktuatoren in kaum nennenswerter Steigerungsrate zu, da sie im wesentlichen durch die Antriebseinheiten (Powertrain), Lenkung sowie einige zusätzlicher Komfortmechaniken definiert sind. Zwar existiert im Bereich Komfort ebenfalls der Bedarf nach zusätzlicher Aktuatorik, beispielsweise elektronische Fensterheber oder sich dem Fahrverhalten anpassende Sitzflächen, jedoch ist diese Zahl im Vergleich zu der Zahl neuer Funktionen verschwindend gering. Dies führt dazu, dass eine wachsende Anzahl an Regelungen mit eine nahezu konstant bleiben Anzahl an Aktuatorik interagieren muss. Zur Vermeidung von unkontrollierten Interferenzen der einzelnen Funktionen müssen diese sich folglich auf eine festgelegt Art und Weise synchronisieren. Zudem verlangen neue Funktionen oftmals Eingriffe in Systeme, die nicht unmittelbar mit der neuen Funktion verbunden sind. Ein beliebtes Beispiel für ein derartiges Verhalten ist das Verhalten eines Automobils im Falle eines Zusammenstoßes: Wird ein Crash von einem entsprechend dafür vorgesehenen Sensor detektiert, so sollen zunächst die passenden Airbags ausgelöst werden. Gleichzeitig greift die Crash-Ermittlung aber auch in die Steuerung der Zentralverriegelung ein und öffnet die Schließsysteme aller Türen, damit nach dem Unfall Rettungspersonal an die Insassen herankommt oder diese ggf. das Fahrzeug verlassen können. Hier wird eine Vernetzung von vormals unabhängigen Steuergeräten verlangt, die fordert, dass eine physikalische Verbindung zwischen den einzelnen Steuergeräten, die die jeweilige Funktion implementieren, existiert. Betrachtet man die Anzahl der Funktionen, die Anzahl der Abhängigkeiten und die Anzahl der Steuergeräte, so wird die steigende Komplexität der Integration des Systems deutlich: Ein Mercedes S-Klasse Fahrzeug von 1998 verfügte über eine Anzahl von ca. 80 Steuergeräten, die in einer hierarchisch organisierten Topologie

durch 5 verschiedene Bussysteme verbunden waren. Bild 3.1 gibt einen Eindruck der Vernetzung eines typischen elektronischen Systems im Automobil der gehobenen Preisklasse.

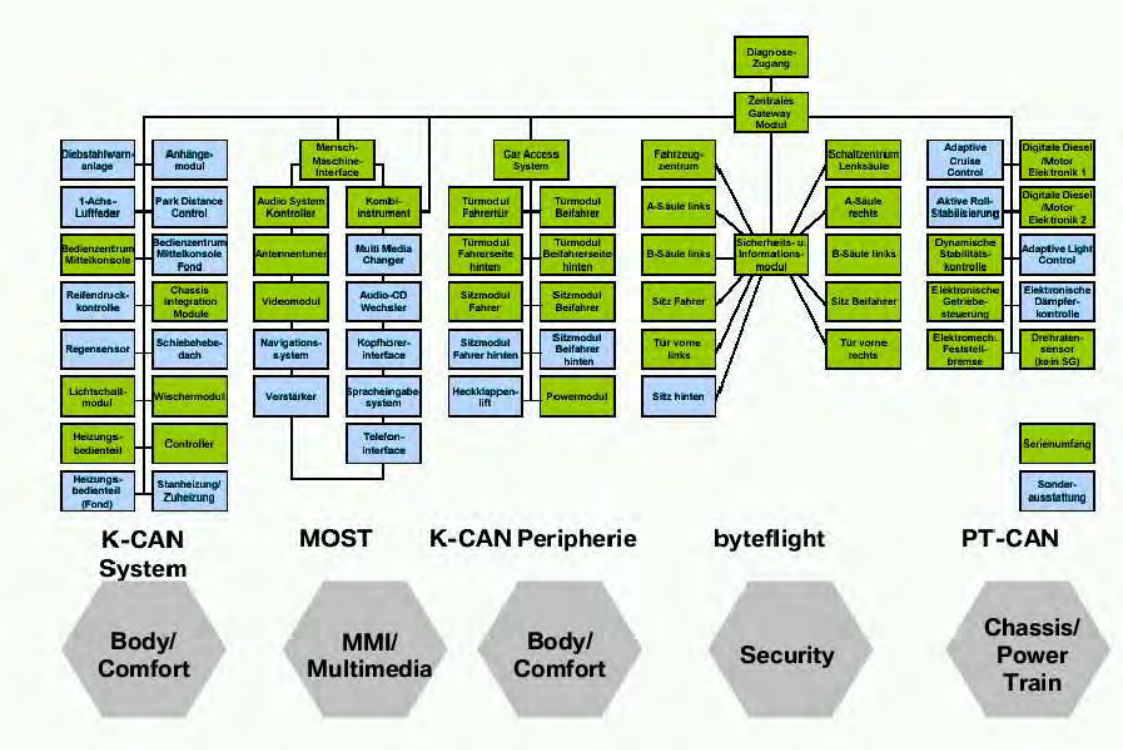


Abbildung 3.1: Hierarchische verteilte Steuergerätearchitektur am Beispiel eines BMW der 7er-Serie. [Quelle: BMW AG, München]

Alle in Abbildung 3.1 dargestellten Steuergeräte müssen prinzipiell eine oder auch mehrere Informationen mit anderen Steuergeräten über die verschiedenen Bussysteme, auch über mehrere Hierchiestufen der Topologie hinweg, austauschen. Dies impliziert ein komplexes Kommunikationsverhalten auf eben jenen Bussystemen, welches während der Integration, aber auch schon während der Spezifikation (siehe Kapitel 1.3) entworfen und überprüft werden muss. Die Auswirkungen dieser zunehmenden Vernetzung einer steigenden Anzahl an Steuergeräten auf die Komplexität des Entwurfes wird in Abbildung 3.2 deutlich (die Abbildung wurde einer Präsentation der BMW Car IT GmbH entnommen und stellt eine Einschätzung der Auswirkung durch zusätzliche Vernetzung im Automobil dar).

An den verschiedenen Kurven in Abbildung 3.2 lässt sich neben der nahezu quadratischen Steigerung der Anzahl der Nachrichten auch ablesen, welchen organisatorischen Mehraufwand (hier gezählt in Abstimmungsgesprächen mit den Zulieferern) eine steigende Vernetzung im Entwicklungsprozess des Gesamtsystems bewirkt.

Die Entwicklung von Funktionen in der Anwendungsdomäne Automobilindustrie (aber ebenso auch in anderen Anwendungsdomänen) wird typischerweise auf Zulieferunternehmen verteilt, vgl. Kapitel 1.3. Dies hat zur Folge, dass nahezu für jede zu

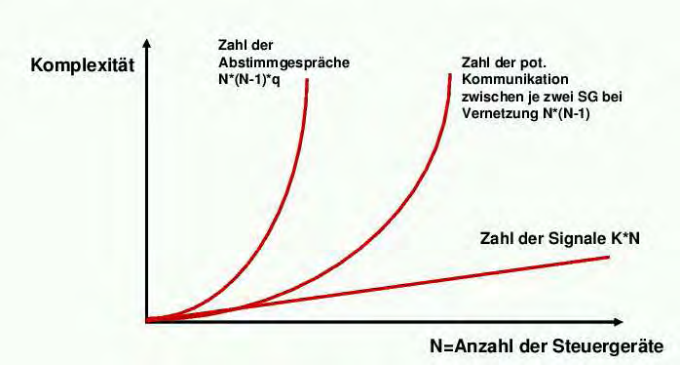


Abbildung 3.2: Steigende Komplexität des Kommunikationsverhaltens bei einer wachsenden Anzahl an Steuergeräten. [Quelle: BMW Car IT GmbH, München]

implementierende Funktion ein eigenes Steuergerät mit einer entsprechenden Anbindung an die Infrastruktur des Fahrzeugs konzipiert wird. Der starke Kostendruck sowie die Diversifizität der Zulieferunternehmen mit ihren jeweiligen über Jahre gewachsenen technischen Ausprägungen führt zu einer sehr großen Heterogenität der Steuergeräte innerhalb eines vernetzten eingebetteten Systems. Dieser Prozess wird zudem noch durch das Angebot eines Fahrzeugtyps mit einer außerordentlich hohen Zahl an verschiedenen Varianten des Leistungsumfangs beschleunigt (vgl. Abbildung 3.1, in der die heller gefüllten ECUs Zusatzausstattungen darstellen, die je nach Wunsch des Kunden integriert werden oder nicht). Insgesamt haben wir es also mit einer komplexen, heterogenen und hierarchisch organisierten Steuergerätearchitektur zu tun, deren Heterogenität sich nicht nur in der Auswahl der jeweiligen Prozessoren für ein Steuergerät niederschlägt, sondern die ebenfalls im Bereich der die Steuergeräte verbindenden Kommunikationsmedien aufzufinden ist. Hier werden unterschiedliche Typen und Paradigmen der Nachrichtenübertragung aus Gründen der Anforderungsdefinition gewählt, beispielsweise weil im Infotainment-Bereich eine sehr hohe Datenübertragungsrate notwendig ist, wohingegen in den X-by-Wire Domänen die Übertragungssicherheit und Echtzeitfähigkeit eine ganz wesentliche Rolle spielt.
Eine weitere sehr wesentliche Auswirkung der Zuordnung von Funktionen zu Steuergeräten ist die damit einhergehende inhärente Kopplung der Steigerungsraten von Software-Funktionen und der Anzahl der (zu vernetzenden) Steuergeräte. Geht man, wie in Kapitel 1.2 bereits erörtert, von jährlichen Steigerungsraten von bis zu 25% im Bereich der Software-Funktionen aus, so wird sehr schnell eine Anzahl an Steuergeräten erreicht, deren Komplexität im Entwurf kaum noch beherrschbar ist; weder seitens des Entwicklungsprozesses noch seitens der Wirtschaftlichkeit der Serienproduktion. Abbildung 3.3 gibt hier einen Überblick über die Entwicklung der letzten 30 Jahre Fahrzeugbau.

Als Konsequenz aus diesen Tatsachen kann nur eine generelle Umstellung der Entwicklung derartig komplexer Systeme erfolgen: Funktionen können nicht mehr grundsätzlich jeweils einem neu zu implementierenden Steuergerät entsprechen, sondern sie müssen in einer offeneren Systemarchitektur mit höheren Abstraktionsniveaus

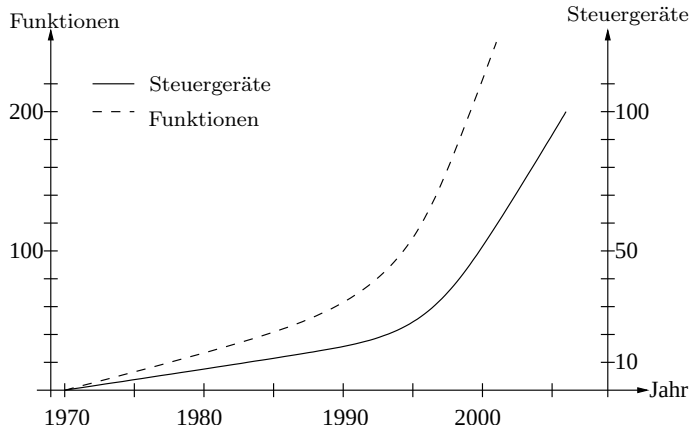


Abbildung 3.3: Anzahl der Funktionen und der integrierten ECUs in einem Automobil über die letzten 35 Jahre betrachtet.

geschaffen werden und insbesondere die Eigenschaft der Verschiebbarkeit aufweisen. Dieser Erkenntnis folgend, hat die Automobilindustrie ein Konsortium — das bereits erwähnte AUTOSAR, an dem nahezu alle namenhaften Automobilhersteller und große Teile der Zuliefererindustrie partizipieren — geschaffen, in der eben jene Prozesse eingeleitet werden sollen. Langfristig ist dort das Ziel verankert, die Anzahl der Steuergeräte in einem Fahrzeug auf etwa 20 zu fixieren, die dann jeweils mit einer im Vergleich zur heute eingesetzten Technologie deutlich leistungsstärkeren Hardware ausgestattet sind, mehrere Funktionen gleichzeitig abarbeiten können und zudem eine signifikant geringere Komplexität der Vernetzung untereinander bedürfen.
Offenere Architekturen von Softwarekomponenten bedeuten aber zugleich auch eine wesentlich höhere Flexibiliät in der Exploration des Entwurfsraumes und bieten Platz für Optimierungsverfahren, die die jeweils am besten passende Konfiguration aus Architektur-Topologie und der auf ihr angesiedelten Teile der Softwarefunktionen auswählen können. Ein solches Optimierungsverfahren werden wir im Rahmen dieser Arbeit vorstellen, wollen aber zunächst den Einfluss der Vernetzung und Verteilung von Funktionen auf die Echtzeit-Eigenschaften dieser Systeme betrachten.

3.2 Einfluss der Verteilung auf Echtzeit

Zunächst einmal bedeutet der Begriff “Verteilung”, dass es etwas zu verteilen gibt, d.h. die verschiedenen Funktionen eines technischen Systems tatsächlich in verteilbare Fragmente zergliedert werden können, die dann notwendigerweise miteinander kommunizieren müssen. Bereits in Kapitel 2 wurde dies unter dem Begriff der Nebenläufigkeit eingeführt. Zusätzlich haben wir in Kapitel 3.1 gesehen, dass es durchaus auch Funktionen gibt, die mit anderen Funktionen kommunizieren müssen (beispielsweise Crash-Verarbeitung mit Zentralverriegelung). Weitere derartige Konfigurationen entstehen, wenn Funktionen in kleinere Einheiten dekomponiert werden und damit verteilbare Fragmente mit der Notwendigkeit des Informationsaustausches entstehen. Solche Ketten von interagierenden Funktionen werden Funktionsnetzwerke genannt. Sie bilden üblicherweise azyklische Bäume, können allerdings auch zyklisch aufgebaut sein, wenn es sich um ein Regelverhalten mit Feedback-Loop han-

delt.

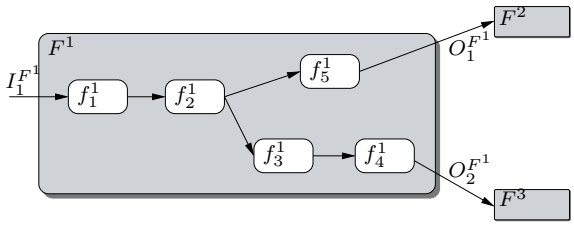


Abbildung 3.4: Funktion F_1, die mit anderen Funktionen kommuniziert und intern in einem Dekompositionsschritt in Fragmente f_1^1 bis f_5^1 komponiert ist.

Abbildung 3.4 zeigt beispielhaft ein Szenario eines Funktionsnetzwerkes: Funktion F^1 kommuniziert mit den Funktionen F^2 und F^3 und ist gleichzeitig intern in ein weiteres Funktionsnetzwerk f_1^1 bis f_5^1 dekomponiert. Gemäß den Ausführungen in Kapitel 2 können nun Echtzeiteigenschaften für Funktionen oder Funktionsnetzwerke definiert werden, die dann offensichtlich über die sequentielle Ausführung der miteinander kommunizierenden Funktionen eines Funktionsnetzes aufgespannt werden. Üblicherweise wird für jeden möglichen und aus Sicht der Anforderungsspezifikation sinnvollen Pfad innerhalb eines Funktionsnetzwerkes eine solche Zeitanforderung (also eine Deadline) gelegt. Deadlines über Pfaden in Funktionsnetzwerken werden "End-to-End-Deadlines" genannt und beschreiben ein maximal zu konsumierendes Zeitintervall von der Detektierung des Aktivierungsereignisses der ersten Funktion eines Pfades bis zur Terminierung (plus ggf. Ansteuerung einer Aktuatorik) der letzten Funktion des Pfades. Damit genügen die in Kapitel 2 vorgestellten Analysemethoden alleine nicht mehr den Anforderungen zur garantierten Sicherstellung des Echtzeitverhaltens, da sich im Falle von End-to-End-Deadlines die gesamte Ausführungszeit eines Pfades im Funktionsnetzwerk aus 1. den Summen der Antwortzeiten der einzelnen Funktionen des Pfades, 2. zuzüglich der Latenzzeiten für die Kommunikationen zwischen den Funktionen und 3. gegebenenfalls zuzüglich Synchronisationskosten zwischen Empfang einer Nachricht und setzen des Aktivierungsereignisses der Empfängerfunktion zusammensetzt. Dementsprechend sind in einem verteilten System jeweils die einzelnen Funktionen in der klassischen Art und Weise zu analysieren und zusätzlich müssen die Kommunikationen betrachtet werden, die über komplexe hierarchische Bussysteme, wie etwa das in Abbildung 3.1 dargestellte, übermittelt werden. Typischerweise wird innerhalb der Integrationsphase des Systems Nachrichtenübermittlung durch simulative oder messende Analysemethoden betrachtet. Ein Beispiel einer solchen Methode ist die weit verbreitete sogenannte Restbussimulation[84], bei der ein physikalisches Bussystem und eine Teilmenge der physikalischen Steuergeräte mit einem Teil simulierter Steuergeräte verschaltet wird und das zeitliche Verhalten der Nachrichten auf dem Bussystem beobachtet wird. Wie eingangs schon erwähnt haben derartige Methoden aber den gravierenden Nachteil, dass sie auf der einen Seite kein vollständiges Abbild aller möglichen Szenarien liefern können und auf der anderen Seite nicht geeignet sind, in der Exploration des Entwurfsraumes intensiv eingesetzt zu werden. Insbesondere wenn wir uns mit der Verteilung von Funktionsnetzwerken auf verteilte Architekturen befassen, müssen in-

nerhalb eines Optimierungsverfahrens aber sehr viele alternative Verteilungsinstanzen betrachtet werden, so dass ein Einsatz von simulations- und messungsbasierter Verfahren nur sehr eingeschränkt benutzt werden kann. Stattdessen muss hier Ziel der Bemühungen sein, eine formale Analyse auch für die Kommunikationsübermittlung und das Zusammenspiel der verteilten Funktionen zu erhalten.
Neben der notwendigen Analyse des Echtzeitverhaltens dieser Kommunikationen existieren aber auch weitergehende Einflüsse der Verteilung auf die Analyse von durch Kommunikationen angesteuerte Funktionen: Selbst wenn die erste Funktion eines Pfades über ein streng periodisches Aktivierungsereignis verfügt, sorgen die durch verschiedene aktuelle Szenarien hervorgerufenen Schwankungen in der Antwortzeit der Funktion und auch der Kommunikationen dafür, dass nachfolgende Funktionen bezüglich ihres Aktivierungsereignisses (in diesem Fall also das Ankommen der zugehörigen Kommunikation) ebenfalls diesen Schwankungen unterworfen sind. Je länger ein Pfad eines Funktionsnetzwerkes, desto höher werden diese Schwankungen für die letzten Funktionen des Pfades. Schwankungen dieser Art werden Jitter genannt und können in die Echtzeitanalyse integriert werden (vgl. Gleichung 2.3 in Kapitel 2.3 auf Seite 25), können aber im Zusammenhang mit verteilten Systemen ein sehr komplexes Einschwingverhalten der zeitlichen Analysen verursachen. Wir werden dieses Thema in Kapitel 4.4.1 eingehender diskutieren und das Phänomen durch Überapproximation eingrenzen.
Werden letztlich Funktionsnetzwerke auf Tasknetzwerke projeziert, wie es in Kapitel 2.1 angedeutet und hier als gegeben vorrausgesetzt wird, kann eine Verteilung der Ausführungseinheiten unter Einbeziehung der oben skizzierten Analyse auf eine verteilte Architektur erfolgen.

3.3 Platzierungsproblem

Die Berechnung einer Verteilung von Funktions-, oder genauer gesagt Tasknetzwerken auf eine verteilte Architektur erfordert zunächst das Finden einer bestimmten Zuordnung von Tasks zu Steuergeräten, die keine der angegebenen Randbedingungen (beispielsweise Echtzeit) verletzt. Hinzu kommt in verteilten Systemen auch noch das Finden einer geeigneten Zuweisung von Nachrichten zu Kommunikationsmedien, wobei auch hierbei wiederum entsprechende Randbedingungen einzuhalten sind. Damit ergibt sich ein klassisches kombinatorisches Zuordnungsproblem, das mit Hilfe von verschiedenartigen Optimierungsverfahren gelöst werden kann. Die Randbedingungen, die bei der Zuordnungsfindung beachtet werden müssen, decken die Bereiche ab, die in Kapitel 1.3 bereits erörtert wurden, wobei im Fokus dieser Arbeit die Echtzeitfähigkeit stehen soll. Trotzdem werden im weiteren Verlauf auch weitere, für die gewählte Anwendungsdomäne Automobilindustrie typische, Randbedingungen in das Verfahren integriert und zeigen beispielhaft, wie dies auch für andere, nicht betrachtete, Randbedingungen geschehen kann. Im Einzelnen werden wir neben der Echtzeitfähigkeit noch als Vertreter für beschränkte Resourcen den Speicherverbrauch betrachten, sowie als Vertreter für Ausfallsicherheit und Verfügbarkeit Redundanzsysteme aufnehmen. Die in zahlreichen Domänen (beispielsweise beim Entwurf von Mobiltelefonen, PDAs, etc.) überaus wichtige Frage nach der Lei-

stungsaufnahme spielt in der Automobilelektronik nahezu keine Rolle, so dass im weiteren für dieses Themenfeld keine Integration in ein Optimierungsverfahren erfolgen muss. Gleichwohl wird in Kapitel 6.4 gezeigt, wie eine Integration auch von Leistungsaufnahme in einer mehrfachparametrisierten Optimierung möglich ist.
Die Frage nach der Güte einer gefundenen Platzierung entscheidet das jeweilige Optimierungskriterium. Gütefunktionen können mannigfaltiger Natur sein und hängen ganz erheblich vom Kontext der Entwicklung ab. Da die in dieser Arbeit vorgestellten Verfahren sich vornehmlich der Echtzeitfähigkeit widmen, werden die Optimierungsverfahren jeweils bezüglich dieser Sicht gewählt, also beispielsweise die Minimierung der Zykluszeiten von Buszyklen oder der Auslastung von Steuergeräten[1].

Eine weitere wichtige Betrachtungsebene ergibt sich aus der Analyse der Komplexitäten des Integrationsvorgehens im Entwurfsprozess der anvisierten Anwendungsdomäne. Wir haben bereits in den vorangegangenen Kapiteln erörtert, dass der Entwurf auf Basis der einzelnen Funktionen sehr zuliefererorientiert ist und die Integration der verschiedenen Teilsysteme nur dann komplexitätsreduzierend wirken kann, wenn eine einfache Komposition der Komponenten möglich ist. Der Begriff "einfach" ist hier so zu verstehen, dass die Komposition von Komponenten nur von diesen direkt betroffene Teilsysteme verändert, sich jedoch nicht transitiv in das gesamte System ausbreitet. Ist dies gegeben, so kann auch eine inkrementelle Konstruktion des Gesamtsystems erfolgen, die dem Umstand unterschiedlicher Auslieferungszeiten aber auch einer virtueller Integration Rechnung trägt und somit die oftmals eingesetzte Technik der Restbussimulation durch formale Komposition (zumindest für den Bereich der Echtzeitanalysen) ersetzen kann. Für die Echtzeitanalysen selbst bedeutet dies beispielsweise das zurverfügungstellen von lokalen Analysen auf den von neuen Funktionsanteilen betroffenen Teilen der Architektur. Dieses Vorgehen würde im Prinzip dem Begriff des "Plug and Play" entsprechen und erlaubt eine deutlich vereinfachte Integration. Vorbedingung hierfür sind dann aber klar definierte Schnittstellen der einzelnen Komponenten bzgl. der Echtzeitparameter, sowie ein abstrahierter Entwurfsstil der Komponenten, wie er unter anderem auch im AUTOSAR-Ansatz festgeschrieben ist.

Generell sind also im Folgenden drei Fragen zu erörtern:

1. Wie sehen die Echtzeitanalyse-Techniken für verteilte eingebettete Echtzeitsysteme aus, die bei einer Platzierungsberechnung eingehalten werden müssen?

2. Wie sieht ein Optimierungsverfahren für eine Platzierungsberechnung auf eine verteilte Architektur aus, die alle angegebenen Randbedingungen berücksichtigt und gleichzeitig gute (im Sinne des jeweiligen Optimierungskriteriums) Verteilungen liefert?

3. Wie kann ein solches Verfahren unterstützend in den Integrationsprozess innerhalb des Vorgehensmodelles (hier im wesentlichen das V-Modell und unter Berücksichtigung der domänenspezifischen Gegebenheiten) eingreifen?

[1] Dies ist vollkommen analog zur Betrachtung von Resourcenverbrauch allgemein, und auch andere Kriterien können prinzipiell in die vorgestellten Verfahren integriert werden.

Diese drei Fragen werden in den nun folgenden drei Kapiteln beantwortet, indem zunächst die mathematische Modellierung des Platzierungsproblems inklusive seiner Randbedingungen angegeben wird. Anschließend werden wir die Modellierung nutzen um ein speziell gewähltes Optimierungsverfahren auf diese Problemklasse anzuwenden, und abschließend wird die unterstützende Integration des Verfahrens in den Entwurfsprozess gezeigt. Um den Rahmen dieser Arbeit nicht zu sprengen, werden in der jeweiligen Beantwortung obiger Fragen einige Einschränkungen gemacht. So wird die Klasse der zu analysierenden Echtzeitsysteme auf offline generiertem prioritätenbasierten preemptiven Scheduling (also ereignisbasierten Tasksystemen) basieren und die Nachrichtenübermittlung wird ebenfalls auf die ereignisbasierte Versendung beschränkt. Gleichwohl werden zeitbasierte Kommunikationsmedien für die Übermittlung derartiger Nachrichten verwendet. Andere Arten des Scheduling und der Kommunikationsübermittlung können auf analoge Weise behandelt werden. Zum Zweiten wird bei der Unterstützung des Integrationsprozesses auf eine detaillierte Betrachtung der Komposition von Komponenten inklusive einer Kompositionstheorie für Echtzeitsysteme verzichtet. Eine derartige Theorie kann zweierlei bewirken: Entweder sie führt einen Kompatibilitätstest der Schnittstellen miteinander verbundener Komponenten durch oder aber sie induziert eine Synthese von Echtzeitverhalten in einer Schnittstelle einer Komponente, die mit anderen Komponenten komponiert wird[1]. Die Erstellung von solchen Kompositionsregeln ist ein komplexes eigenes Teilgebiet und soll hier nicht eingehender betrachtet werden, sondern wir verweisen den interessierten Leser hier auf entsprechende aktuelle Arbeiten im Umfeld der Rich Component Modelle[192] sowie anderer[151]. Stattdessen wird im Kontext dieser Arbeit eine simplifizierende Theorie benutzt (vgl. Kapitel 4.1), und die Konzentration liegt auf der Gewinnung der Echtzeitschnittstellen außerhalb der kompositionsinduzierten Aktivierungsschemata.

Mit diesem Vorbemerkungen und den bisher beschriebenen Randbedingungen können wir nun an die Beantwortung der 3 obigen Fragen gehen und zunächst die formale Natur der Problemstellung analysieren, sowie die Echtzeitanalysen für die hier zu betrachtenen Systeme, insbesondere bezüglich der Kommunikationsabwicklung, definieren.

[1] Als Beispiel soll hier das Aktivierungsschema betrachtet werden: Streng periodisch aktivierte Komponenten bewirken auch eine streng periodische Aktivierung von mit ihnen verbundener Nachfolgekomponenten, fügen allerdings zusätzlich einen Jitter hinzu. Kompabilitätstests müssten prüfen, ob die angeschlossene Schnittstelle als mindestens “sporadisch mit Jitter” definiert wurde, bzw. ein syntheseorientierter Ansatz würden eben jenes Aktivierungsschema erzeugen.

Kapitel 4

Formalisierung des Verteilungsproblems

4.1 Begriffsbildung

Steuergerätenetzwerke, wie sie im letzten Kapitel vorgestellt wurden, können in unterschiedlichsten Ausprägungen und Topologien vorkommen. Wie schon ausgeführt, ist in den hier betrachteten Anwendungsdomänen ein wesentliches Merkmal die Heterogenität der Steuergeräte und Verbindungsnetzwerke. Verfahren, die eine Platzierung von Software-Funktionen auf ein solches Steuergerätenetzwerk berechnen sollen, müssen diesem Umstand Rechnung tragen. Als Basis für alle im Rahmen dieser Arbeit beschriebenen Techniken dient die folgende Definition einer vernetzten, heterogenen Architektur:

Definition 4.1.1 (Architektur). *Eine Architektur $\mathcal{A}$ ist gegeben durch ein Tupel $\mathcal{A} = \big(P, \mu^{\mathcal{A}}, K, \kappa\big)$, mit*

- *$P = (\mathcal{P}, OS)$: Menge der Prozessorknoten. Jeder Prozessorknoten ist mit einer Menge von spezifischen Attributen ausgestattet, die das Ausführungsmodell von Programmen auf diesem Knoten charakterisieren, mit*
  - *$\mathcal{P}$ Ausführungsmodell des Prozessors auf dem Knoten. Dieses wird verwendet, um die durch die Prozessorarchitektur bedingten Einflüsse auf die Laufzeit ermitteln zu können (vergleiche die Techniken aus Kapitel 2.4, in denen basierend auf Programm und $\mathcal{P}$ die Laufzeit ermittelt wird). Wir werden dies im folgenden vorraussetzen und nicht weiter betrachten.*
  - *$OS \in \{FPS, TT, \ldots\} \times \mathbb{N} \times \mathbb{N}$ charakterisiert das verwendete Echtzeitbetriebssystem und damit die verwendete Scheduling-Art (z.B. Time-triggered oder Event-driven, preemptiv oder blockierend, etc., die jeweils durch die Wahl einer RTOS Implementierung pro ECU festgelegt sind). Im folgenden wird von einem fixed-priority preemptiven Scheduling (FPS) ausgegangen. Die weiteren Parameter dienen der Quantifizierung von Laufzeitverlusten durch Abarbeiten von Interrupts und OS-Routinen und werden in Teil 2 der Arbeit eingehender diskutiert.*
- *$\mu^{\mathcal{A}} : P \to \mathbb{N}$: Größe des Speichers auf den jeweiligen Knoten.*

- $K = \{(k_i, id_i) | k_i \in 2^P \wedge id_i \in \mathbb{N}\}$*: Menge der Bussysteme. Jedes Bussystem* $k = (\{p_1, \ldots, p_n\}, id) \in K$ *verbindet die angegebene Menge von Prozessorknoten* $p_i \in k$ *mit allen anderen Knoten* $p_j \in k$*.*

- $\kappa : K \rightarrow (\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N})$*: die spezifischen Leistungsdaten der Bussysteme mit* $\kappa(k) = (h_k, \eta_k, \upsilon_k, \beta_k)$*, wobei*
  - h_k *die Anzahl der Headerbits eines Datenpaketes (Frame),*
  - η_k *die Anzahl der Nutzdatenbytes eines Datenpaketes,*
  - υ_k *die Übertragungsrate des Mediums in Bit/Zeiteinheit*
  - *und schließlich* β_k *die Bit-Stuffing-Distanz angibt. Bit-Stuffing ist ein der Synchronisation dienendes Verfahren, dass in Kapitel 4.3 eingehend erläutert wird.*

- *zusätzlich wird eine Abbildung* $type : K \rightarrow \{CAN, Flexray, TTP, Tokenring, \ldots\}$ *eingeführt, die die Art des verwendeten Busprotokolls und damit gleichzeitig auch die Scheduling-Policy auf diesem Bussystem festlegt.*

Diese allgemein gehaltene Definition der Architektur führt zu beliebigen, heterogenen Prozessornetzwerken, in denen zum einen die Prozessorknoten selbst differieren und zum anderen unterschiedlichste Bussysteme für die Verbindung der Prozessorknoten untereinander eingesetzt werden können. Abbildung 4.1 zeigt ein Beispiel für eine derartige Architektur.

Die beliebige Vernetzung der Prozessorknoten führt auch dazu, dass manche Kno-

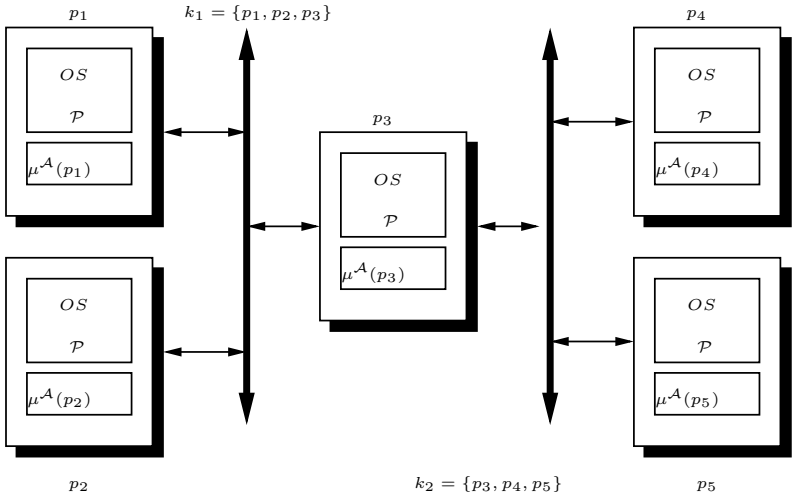


Abbildung 4.1: Beispiel einer heterogenen verteilten Steuergerätearchitektur.

ten von anderen nur über sogenannte Gateway-Knoten, die mehrere Bussysteme verbinden, zu erreichen sind. In Abbildung 4.1 beispielsweise ist $p_3 \in k_1 \cap k_2$ ein solcher Gateway-Knoten.

Die unterschiedlichen Arten der Implementierung der jeweiligen Prozessorknoten selbst (z.B. dedizierte MicroController) werden im Attribute $\mathcal{P}$ gekapselt und innerhalb einer Analyse der Ausführungszeiten berücksichtigt (vgl. Kapitel 2.4). Der

Einfluss der verschiedenen Echtzeit-Betriebssysteme beschränkt sich im wesentlichen auf die Echtzeitanalyse von Tasks auf diesen Knoten (siehe hierzu Abschnitt 4.2). Weitergehende Architekturkonzepte, wie beispielsweise die Kombination von Prozessoren und DSPs innerhalb eines Steuergerätes, sind abbildbar auf eine lokale vernetzte Struktur gemäß obiger Definition 4.1.1, so dass auch komplexere Architekturen mit diesem Formalismus der Platzierungsberechnung zugänglich gemacht werden. Prinzipiell sind mithilfe der Definition auch beliebige Zyklen und multiple Wege durch das Netzwerk möglich. Wir werden später bei der Modellbildung zeigen, auf welche Art und Weise neben der Platzierung von Tasks auch die Pfadsuche für Nachrichten auf komplexen, hierarchischen und potentiell zyklischen Topologien gelingen kann. Die einzige Einschränkung, die hierfür gemacht wird, ist die Verhinderung einer mehrfachen Benutzung eines Kommunikationsmediums durch eine Nachricht, da dies offensichtlich ein unerwünschtes Verhalten ist.

Tasknetzwerke sind in Definition 4.1.2 thematisiert und jede Task wird dabei über ihre jeweiligen Eingeschaften (Laufzeit, Speicherverbrauch, Platzierungsrestriktionen, etc.) sowie über die von ihr ausgesendeten Nachrichten charakterisiert. Gleichzeitig erfolgt in ihnen die Definition der Taskketten, über die jeweils eine End-to-End-Deadline gefordert ist. Gemäß der Anschauung hat zunächst nur die initiale Task einer Taskkette sowie alle etwaig enthaltenen zeitbasierten Tasks einer Kette eine fest definierte Periode. Im folgenden Formalismus wird trotzdem auch für alle anderen Tasks einer Kette eine Periode (oder minimale Zwischenankunftzeit bei sporadischen Tasks) eingeführt. Da, wie bei der folgenden Definition von Tasknetzwerken zu sehen sein wird, wir hier davon ausgehen, dass Nachrichten einer Task bei jeder Aktivierung der Task versendet werden, wird die Periode der initialen Task durch die gesamte Kette propagiert. Die konkreten Werte der Periode einer Task einer Taskkette mit den beschriebenen Eigenschaften werden demnach von der Periode der Taskkette geerbt und sind — bis auf zeitbasierte Tasks — Teil der Komposition von Tasks und nicht Teil der Taskspezifikation selbst. Insgesamt kann ein Tasknetzwerk dann folgendermaßen definiert werden:

Definition 4.1.2 (Tasknetzwerk). *Ein Tasknetzwerk auf einer Menge von Tasks* $\mathcal{T}$ *ist definiert durch:* $\forall \tau_i \in \mathcal{T} : \tau_i = (t_i, d_i, \mu_i^{\mathcal{T}}, c_i, \gamma_i, \pi_i, \delta_i, tr_i)$ *mit:*

- $t_i \in \mathbb{N}$ *: Periode einer periodischen Task oder minimale Zwischenankunftszeit bei sporadischen Tasks*
- $d_i \subseteq \mathcal{T}^{\leq N} \times \mathbb{N}$*: End-to-End-Deadlines der Taskketten, in denen* τ_i *die erste Task einer Kette ist, d.h. End-to-End-Deadlines werden in diesem Formalismus immer dem ersten Task einer Kette zugeordnet.*
- $\mu_i^{\mathcal{T}} : P \rightarrow \mathbb{N}$ *: Speicherverbrauch dieser Task*
- $c_i : P \rightarrow \mathbb{N}$ *: WCET der Task auf den jeweiligen Prozessoren einer Architektur*
- $B_i : P \rightarrow \mathbb{N}$ *: Maximale Blockierungszeit die durch die Benutzung einer kritischen Resource oder nicht-preemptive Anteile erzeugt wird.*

- $\gamma_i \subseteq \mathcal{T} \times \mathbb{N}$ *: $g_j = (\tau_j, s) \in \gamma_i$ ist eine Kommunikation von Task τ_i zu Task τ_j mit einem Datenpaket der Größe s Bytes. Das Senden einer Nachricht g_j ist nicht blockierend, die Nachricht wird bei jeder Invokation der Task an deren Ende versendet.*

- $\pi_i \subseteq P$ *: Menge der für diese Task erlaubten Prozessorknoten; es sind nur Platzierungen auf einen Knoten dieser Menge für τ_i erlaubt.*

- $\delta_i \subseteq \mathcal{T}$ *: Redundanzenkollisionsvermeidung: Tasks aus der Menge δ_i dürfen nicht zusammen mit τ_i auf einem Knoten platziert werden.*

- $tr_i \in \{TRIGGER, TIME\}$ *: Triggerung dieser Task. TRIGGER gibt an, dass dieser Task durch eingehende Nachrichten aktiviert wird, TIME gibt an, dass die Aktivierung zeitbasiert erfolgt, unabhängig von womöglich anliegenden Nachrichten.*

Die Definition eines Tasknetzwerks basiert bzgl. der Deadlines auf End-to-End-Deadlines in der Eigenschaft d_i. Dabei ist d_i eine Menge von Tupeln über einem Wort der maximalen Länge $|\mathcal{T}|$ über $\mathcal{T}$ und der eigentlichen Deadline. Das Wort gibt den Pfad der Taskkette über den Tasks an, also beispielsweise $d_1 \ni (\text{“}\tau_1\tau_2\tau_5\tau_7\text{”}, 25)$. Task-lokale Deadlines sind einfach auszudrücken, indem Taskketten der Länge 1 definiert werden. Im Rahmen dieser Arbeit wird davon ausgegangen, dass Deadlines für Taskketten der Länge 1 grundsätzlich die Verschickung von Nachrichten dieser Task beinhalten. Zusammen mit der Menge der Kommunikationen γ_i wird der Daten- und Kontrollflussgraph des Tasknetzwerkes repräsentiert[1]. Zyklen im Kontrollflussgraphen eines Tasknetzwerkes sind nur dann erlaubt, wenn mindest eine zeitbasierte Task in dem Zyklus liegt[2]. Formal muss also sichergestellt sein, dass die folgende Bedingung für eine Taskkette $\tau_1 \rightarrow \ldots \tau_i \rightarrow \tau_j \rightarrow \ldots \rightarrow \tau_{j+l} \rightarrow \tau_i \rightarrow \ldots \rightarrow \tau_n$ erfüllt ist:

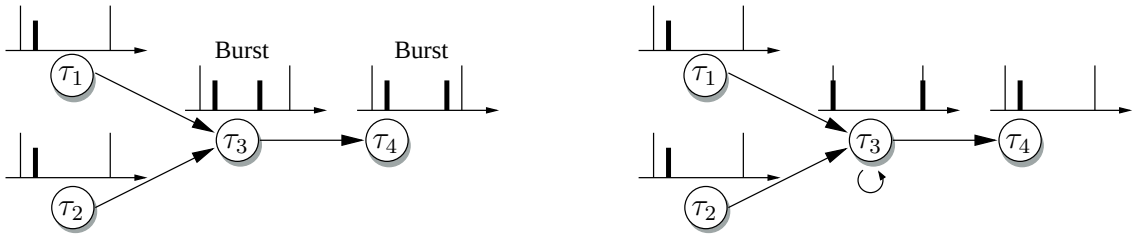
$$\exists r \in \{i, j+l\} : tr_r = TIME \tag{4.1}$$

Dieser Mechanismus verhindert, dass Zyklen mehrfach durchlaufen werden. Diese Art der Terminierung von Zyklen ist angelehnt an die aus der Hardware-Synthese bekannte Stabilisierung an flankengesteuerten Registern. Allerdings ist die Bedingung 4.1 nicht streng genug, generell das zumindest zweifache Ausführen von Teilmengen eines Task-Zyklus zu verhindern.
Eine weitere Einschränkung ist im Kontext dieser Arbeit für Tasks notwendig, die von mehreren Vorgängertasks kausal abhängig sind. Abbildung 4.2(a) macht das Problem beispielhaft deutlich: Selbst wenn alle die betrachtete Task τ_3 startenden Tasks streng periodisch ausgeführt werden, lässt dies nur in gewissem Rahmen Rückschlüsse auf das Verhalten der kausal abhängigen Task τ_3 zu. Aufgrund der nicht verfügbaren Relation der Perioden von τ_1 und τ_2 zueinander, muss die angesteuerte Task ein Burst-Verhalten[90] aufzeigen.
Burst-Verhalten bedeutet, dass innerhalb eines festdefinierten Zeitintervalls maximal eine durch eine obere Grenze n beschränkte Menge an Aufrufen der Tasks erfolgen.

[1] Der Kontrollflussgraph bezieht sich auf kausale Abhängigkeiten von Tasks über Nachrichten, d.h. die Empfänger-Task muss die Eigenschaft TRIGGER haben.

[2] Andernfalls sind die im folgenden vorgestellten Analysemethoden nicht terminierend.

Burst-Tasks können auf einfache Art und Weise durch Schedulinganalysen erfasst werden, wenn davon ausgegangen wird, dass sie innerhalb von sogenannten sporadischen Servern[169] behandelt werden. Sporadische Server stellen eine eigenständige Task dar, deren Aufgabe es ist, die mehrfachen Ausführungen der Burst-Task innerhalb der Periode des sporadischen Servers aufzunehmen. Die für diese Operation notwendige Berechnungszeit stellt der Server zur Verfügung. Im Rahmen dieser Arbeit werden sporadische Server jedoch nicht weiter betrachtet[1].



(a) Triggerung durch mehrere Tasks erzeugt Burst-Verhalten

(b) Zeit-basierte Task verhindert Burst-Verhalten

Abbildung 4.2: Auflösung des Burst-Verhaltens durch die Umstellung der Aktivierung einer Zieltask auf Zeit-basiertes Verhalten ($tr_3 = TIME$).

Ohne die Verwendung sporadischer Server sind die angewendeten Zeitanalysen nicht korrekt, da sie von einer Periodizität der Tasks ausgehen. Das Burst-Verhalten kann jedoch aufgelöst werden, wenn die kausale Abhängigkeit der Zieltasks von allen Vorgängertasks abgeschwächt wird und stattdessen die Zieltask mit einem Zeit-basierten Aktivierungsmechanismus ausgestattet wird (vgl. Abbildung 4.2(b)). Mithilfe der dann vorliegenden strikten Periodizität kann auch für alle nachfolgenden Tasks wiederum das durch den Zieltask induzierte Burst-Verhalten in ein periodisches oder sporadisches Verhalten transformiert werden. Diese Einschränkung bzgl. der Eigenschaften von Tasksystemen für die hier betrachteten Analysetechniken lässt sich in der folgenden Wohlgeformtheitseigenschaft für Tasksysteme formalisieren:

$$\forall \tau_k \in \mathcal{T}, \exists g_k^i \in \gamma_i, \exists g_k^j \in \gamma_j : g_k^i = (\tau_k, .) \wedge g_k^j = (\tau_k, .) \rightarrow tr_k = TIME \qquad (4.2)$$

Diese zeitbasierte Verwendung einer Task an der Vereinigungsstelle zweier oder mehrerer Taskketten wirkt nun faktisch wie ein pollender Task für den Nachrichtenempfang (vgl. [90]). Dementsprechend kann die zeitbasierte Task auch mit einer höheren Frequenz ausgestattet werden, als dies für die Taskkette insgesamt definiert wurde und induziert damit eine höhere Frequenz auch für alle nachfolgenden Tasks. Dies widerspricht allerdings der Anschauung, dass Tasks einer Taskkette zunächsteinmal selbst keine Periode haben, sondern diese für die Taskkette insgesamt festgelegt ist. Um diesem Dilemma zu entgehen, wird im folgenden (wie in Kapitel 3.3 bereits

[1] Ihre Integration in den hier dargestellten Formalismus stellt kein großes Problem dar, aus Gründen der Übersichtlichkeit wird aber darauf verzichtet. Zugleich stellen Standard-Echtzeitbetriebssysteme diesen Mechanismus meist nicht zur Verfügung, so dass sporadische Server in der hier betrachteten Anwendungsdomäne kaum Verwendung finden.

angedeutet) eine Simplifizierung eingeführt, die die ausgehenden Nachrichten dieser Schnittpunkttasks nicht mit der Frequenz der pollenden Task ausstattet, sondern immer mit der Frequenz der zugehörigen Taskkette. In diesem Sinne vollführt die pollende Schnittstellentask im Prinzip ein Aufsammeln der Nachrichten in Form eines "Leaky Bucket"-Modells[22] (bekannt beispielsweise aus dem ATM-Protokoll). Dieses Verfahren ist an dieser Stelle unschädlich, da wir davon ausgehen, dass die End-to-End-Deadline einer Taskkette grundsätzlich kleiner oder gleich der Periode der Taskkette ist. Gilt diese Vorraussetzung nicht, so erfüllt diese Simplifizierung nicht mehr das intendierte Verhalten, und es müssen andere Techniken gewählt werden, auf die wir hier allerdings aus Gründen des Umfangs verzichten.
Dieser Ansatz ist ein Spezialfall des in [151] vorgeschlagenen Vorgehens, in dem an derartigen Stellen eine Synchronisation der beteiligten Tasks durch die Verwendung von Puffern für eingehende Nachrichten eingeführt wird. [151] aber auch die Arbeiten im Umfeld der Rich Component Models zu diesem Thema[192] kennen hier noch weitere Kompositionsmöglichkeiten, die eine näher am intendierten Verhalten liegende Analyse unterstützen.
Gleichung 4.2 erlaubt damit wieder die Verwendung von Analysetechniken ohne die Benutzung von sporadischen Servern. Zugleich stellt sie eine Verschärfung der Gleichung 4.1 dar: Für Zyklen, die innerhalb einer Taskkette liegen, wird durch Gleichung 4.2 schon gefordert, dass die Starttask eines Zyklus bereits eine Zeit-basierte Aktivierung aufweisen muss. Weitere Zeit-basierte Aktivierungen von Tasks innerhalb des Zyklus sind nicht mehr notwendig. Für Tasknetzwerke die an sich einen kompletten Zyklus darstellen, ist hingegen die Erfüllung von Gleichung 4.1 nachwievor zu gewährleisten.

Deutlich wird mit obiger Definition 4.1.2 eines Tasksystems, dass ein Tasknetzwerk auch in gewissen Teilen über die Architektur charakterisiert wird. Dies gilt für die Bereiche der erlaubten Knoten in π_i und die unterschiedlichen WCETs, die direkt vom verwendeten System auf den verschiedenen Prozessorknoten abhängig sind. Damit kann nun das Platzierungsproblem angegangen werden, indem die Zuordnung der Tasks zu Prozessorknoten stattfindet und die Nachrichten auf den Verbindungsnetzwerken gerouted werden.

Definition 4.1.3 (Platzierung). *Eine Lösung des Platzierungsproblems ist dann gegeben, wenn die Abbildungen* $\Pi : \mathcal{T} \to P$, $\Gamma : \mathcal{T} \times (\mathcal{T} \times \mathbb{N}) \to \mathrm{K}^{\leq \mathrm{N}}$, *sowie* $\phi : \mathcal{T} \times \mathcal{T} \to \{0, 1\}$ *existieren, die Tasks Prozessoren und Nachrichten Sequenzen von Bussystemen zuordnen, sowie Prioritäten von Tasks festlegen. Diese Abbildungen exitieren genau dann, wenn die folgenden Bedingungen erfüllt sind:*

- *Auf jedem Prozessorknoten muss ausreichend Speicher vorhanden sein, also*

$$\forall p \in P : \mu^{\mathcal{A}}(p) - \sum_{\tau_i : \Pi(\tau_i) = p} \mu_i^{\mathcal{T}}(p) \geq 0 \tag{1}$$

- $\forall \tau_i \in \mathcal{T} : \Pi(\tau_i) = p$ *muss gelten:*
  - p *muss für* τ_i *erlaubt sein:*

$$p \in \pi_i \tag{2}$$

– *Auf p dürfen keine Redundanzkollisionen auftreten:*

$$\left\{\tau_j \mid \Pi(\tau_j) = p\right\} \cap \delta_i = \emptyset \tag{3}$$

– *Für alle Kommunikationen von Task τ_i existiert ein gültiger Pfad durch das Architekturnetzwerk zur Zieltask τ_j, d.h. $\forall g_j = (\tau_j, s_j) \in \gamma_i$ gilt:*

$$\Gamma(\tau_i, g_j) = \begin{cases} \text{“}\varepsilon\text{”} & \text{wenn } \Pi(\tau_i) = \Pi(\tau_j) \\ \text{“}k_1 \dots k_n\text{”}, \text{ mit } \forall l \in \{1, \dots, n-1\} : & \\ k_l \cap k_{l+1} \neq \emptyset \wedge \Pi(\tau_i) \in k_1 \wedge \Pi(\tau_j) \in k_l & \text{sonst} \end{cases} \tag{4}$$

Die Abbildung $\Gamma(\tau_i, g_j)$ stellt hierbei das Routing auf der Architektur dar.

– *Alle End-to-End-Deadlines, beginnend mit Task τ_i müssen eingehalten werden:*

$$\forall d_i^k \in d_i \text{ mit } d_i^k = \left(\text{“}\tau_i \dots \tau_n\text{”}, \lambda\right) : \psi_i^k \leq \lambda \tag{5}$$

ψ_i^k subsumiert die Antwortzeiten der Tasks einer Taskkette bzw. die Antwortzeiten der Kommunikationen. Beides wird weiter unten in Abschnitt 4.2 definiert und ist unter anderem von der Prioritätenrelation ϕ abhängig.

– *Es existierte eine Prioritätenrelation gemäß Definition 4.1.4:*

$$\forall \tau_j \in \{\tau | \tau \in \mathcal{T} \wedge \Pi(\tau) = \Pi(\tau_i)\} : \phi(\tau_i, \tau_j) \in \{0, 1\} \tag{6}$$

Das Platzieren der Kommunikation in der Bedingung (4) ist über die Abbildung $\Gamma(\tau_i, g_j)$ und $g_j = (\tau_j, s_j) \in \gamma_i$ definiert, die einen gültigen Kommunikationspfad auf der Topologie liefert oder für den Fall, dass beide Sender- und Empfängertask auf den gleichen Knoten platziert sind, ein leeres Wort zurückgibt. Wendet man die Vereinfachung an, dass nur jeweils ein Pfad zwischen zwei Knoten existiert und keine Zyklen im Architekturgraphen vorhanden sind, ist die Konstruktion des Routings eine relativ einfache Aufgabe, die keiner weiteren Optimierung unterworfen werden muss. Andernfalls obliegt es dem Optimierungsverfahren, die jeweils günstigsten Wege durch die Topologie des Systems zu finden. Auf welche Weise die Suche nach kostengünstigen Wegen innerhalb eines Optimierungsalgorithmus vollzogen werden kann, ist in Kapitel 5.3.5 dargestellt.
Eine weitere Eigenschaft von Tasks in statischen preemptiven Systemen ist die Priorität, anhand derer ein Scheduler auswählt, welche Task aus einer Menge an bereiten Tasks aktiviert wird[1]. Dies ist eine Knoten-lokale Eigenschaft, die allerdings eindeutig sein muss. Es gibt daher eine partielle Abbildung, die die Prioritätsrelation der Tasks auf einem Knoten festlegt.

[1] In dynamischen preemptiven Systemen wird die Priorität während der Laufzeit bestimmt, so dass in diesem Fall die folgende Definition nicht notwendig ist. Dynamische Prioritätenschemata sollen allerdings im Rahmen dieser Arbeit nicht betrachtet werden.

Definition 4.1.4 (Prioritätsbildung). *Formal ist die Prioritätsbeziehung eine partielle Abbildung* $\phi : \mathcal{T} \times \mathcal{T} \rightarrow \{0, 1\}$ *mit den folgenden Eigenschaften:*

- *Eindeutigkeit der Priorität:* $\phi(\tau_i, \tau_j) = 0 \leftrightarrow \phi(\tau_j, \tau_i) = 1$ *wenn* $\Pi(\tau_i) = \Pi(\tau_j)$
- *Transitivität der Prioritätsrelation:* $\phi(\tau_i, \tau_j) = 1 \wedge \phi(\tau_k, \tau_i) = 1 \rightarrow \phi(\tau_k, \tau_j) = 1$ *wenn* $\Pi(\tau_i) = \Pi(\tau_j)$

Es wird die Schreibweise $\phi_i < \phi_j$ *verwendet, wenn gilt:* $\phi(\tau_i, \tau_j) = 0$.

Die Transitivitätseigenschaft der Abbildung ϕ trägt der Implementierung eines Schedulers Rechnung. Dieser kann nur anhand der Priorität bestimmen, welche Task suspendiert und welche aktiviert wird. Dazu müssen die Werte der Prioritäten auf natürliche Zahlen abgebildet werden und erhalten dadurch eine festdefinierte Ordnung, auf deren Basis der Scheduler seine Entscheidung fällt. Würde für ϕ keine Transitivitätseigenschaft gefordert, so könnte die Priorität nicht in die Menge der natürlichen Zahlen abgebildet werden und es gäbe keine Implementierung eines Schedulers, der dies durch ϕ bestimmte Priortätensysteme verarbeiten könnte. Ähnliches gilt für die Eindeutigkeit, da im Falle uneindeutiger Prioritätenbeziehungen nichtdeterministisch Tasks suspendiert und aktiviert werden müssten.

4.2 Formalisierung des Scheduling-Problems

Scheduling-Analysen berechnen das zeitliche Verhalten einer Menge von Tasks, die einem Prozessorknoten zugeordnet sind. Dabei analysieren sie gemäß der Definition 4.1.3(5) denjenigen Anteil an den gegebenen End-to-End-Deadlines, der alleine von den Tasks aufgrund ihrer Laufzeit unter Berücksichtigung von Preemptionseinflüssen verbraucht wird. Kommunikationslatenzen werden hingegen getrennt im nachfolgenden Abschnitt untersucht.
Basis der Formalisierung des Schedulingproblems sollen im folgenden die in Kapitel 2.3 vorgestellten Analysemethoden sein. Diese sind jedoch im Hinblick auf Möglichkeiten zur freien Platzierung von Tasks auf eine verteilte Architektur zu erweitern. Gleichzeitig werden diejenigen Aspekte nicht weiter betrachtet, die keine qualitativen Einschränkungen der Analysemethodiken darstellen, jedoch die Komplexität der Darstellung deutlich erhöhen würden. Dies gilt insbesondere für die sogenannten "every"-Attribute von Nachrichten und Tasks (wie sie z.B. in [181] eingeführt wurden), sowie Deadlines, die die Periode der zugehörigen Tasks überschreiten[1].
Einzelne Tasks werden mittels der Antwortzeitberechnung aus Kapitel 2.3 analysiert und liefern einen Fixpunkt oder aber die Überschreitung der zur Task gehörenden Deadline. Abhängigkeiten durch unterbrechende, höher priorisierte Tasks bestehen lediglich zu Tasks, die auf dem selben Knoten platziert sind. Dabei ist zu beachten, dass in einer heterogenen, verteilten Architektur die Worst Case Execution Times einer Task τ_i von der Platzierung $\Pi(\tau_i)$ abhängig ist. Dies ist in der Grundstruktur des Tasksystems durch das Vorhalten verschiedener, knoten-abhängiger Werte für

[1] Das Einarbeiten in die in dieser Arbeit vorgestellten Methoden ist an den meisten Stellen einfach möglich.

c_i subsumiert, so dass sich letztlich die folgende Form der Antwortzeitberechnung ergibt:

$$r_i^n = c_i(\Pi(\tau_i)) + B_i + \Omega_i(\Pi(\tau_i)) + \sum_{\tau_j:\phi_j>\phi_i \wedge \Pi(\tau_j)=\Pi(\tau_i)} \left\lceil \frac{r_i^n + J_j - O_j^i}{t_j} \right\rceil \cdot c_j(\Pi(\tau_j)) \quad (4.3)$$

Offsets O_j^i sind hier zwingend nur dann in der angegebenen Art und Weise zu betrachten, wenn es sich um ein rein Zeit-basiertes Tasksystem handelt. Nur in diesem Fall gibt es eine eindeutige Beziehung zwischen den Startzeitpunkten der verschiedenen Tasks. Für den allgemeinen Fall, also Offsets in Ereignis-basierten Tasksystemen oder aber Offsets in Zeit-basierten ohne Kenntniss der Beziehung der Startzeitpunkte, hat schon Tindell in [181] gezeigt, dass diese einfache Form der Integration nicht ausreichend ist. Vielmehr ergibt sich die Notwendigkeit, alle möglichen Verschiebungen der Tasks zueinander betrachten zu müssen. Im Ergebnis ist dies nicht mehr berechenbar.
Da Offsets in Ereignis-basierten Systemen kaum sinnvolle Anwendungsszenarien darstellen, wird im Rahmen dieser Arbeit nur im Kontext Zeit-basierter Kommunikationsysteme eine Integration berücksichtigt. Der Offset O_j^i einer unterbrechenden Nachricht ist dabei relativ zum Startzeitpunkt von τ_i zu betrachten. Dieser kann einfach aus absoluten Offsets O_j und O_i errechnet werden, wenn davon ausgegangen wird, dass die Offsets bzgl. eines dem Kommunikationssystem gemeinsamen Startzeitpunktes vergeben werden[1]:

$$O_j^i = \begin{cases} O_j - O_i & \text{wenn } O_j \geq O_i \\ t_j + O_j - O_i & \text{sonst} \end{cases}$$

Offset-basierende Schedulinganalysen werden weiter unten in Kapitel 25 verwendet, wenn es um die Analyse der Antwortzeit auf gemischt Zeit- und Ereignis-getriebenen Kommunikationsmedien geht. Für die Antwortzeit von Tasks wird dies im folgenden hingegen keine weitere Verwendung finden[2].

Einflüsse eines Echtzeit-Betriebssystems auf das zeitliche Verhalten einer Task τ_i auf einem Knoten sind durch den Summand $\Omega_i(\Pi(\tau_i))$ gegeben und leitet sich aus dem verwendeten Betriebssystem ab (Ω hängt ab von $os = (FPS, d_{isr}, d_{sched}) \in OS$ mit $\Pi(\tau_i) = ((P, OS), \mu^{\mathcal{A}}, K, \kappa)$). Dieser soll hier zunächst uninterpretiert bleiben; eine detaillierte Analyse von durch das Betriebssystem induziertem Overhead erfolgt in Teil II dieser Arbeit.
Generell ist Gleichung 4.3 für alle Arten von statischem Scheduling gültig, wobei die verschiedenen Parameter genutzt werden können, eine jeweilige Anpassung an das verwendete Schedulingverfahren zu erreichen. Dynamische Mechanismen der Taskaktivierung, wie beispielsweise EDF[104] oder Proportional Share Scheduling [174], können mithilfe dieser Gleichungen nicht betrachtet werden, sind aber — wie

[1] Dies kann bei Zeit-basierten Systemen immer einfach geschehen, da die Startzeitpunkte der Nachrichtenslots in der Zeit eindeutig definiert sind.

[2] Zeit-basierte Tasksysteme liegen nicht im Fokus dieser Arbeit, können aber auf ähnliche Weise integriert werden, wie dies im folgenden anhand der Ereignis-basierten Tasks demonstriert wird.

eingangs bereits dargelegt — eher selten in eingebetteten Echtzeitsystemen zu finden. Für eine Anpassung an statische Verfahren sei nachfolgend eine Auflistung mit den Anpassungsmethoden angegeben:

Event-driven, preemptiv verwendet Gleichung 4.3. Blockierungszeiten nur über die Nutzung von kritischen Bereichen, beispielsweise gekapselt durch Semaphoren. Die Abhängigkeit von Nachrichten und sporadischer Ereignisse erzeugt unterschiedliche Jitter der Tasks.

Event-driven, nicht preemptiv verwendet den Blockierungsfaktor B_i in Gleichung 4.3, um das nicht-unterbrechbare Verhalten von Tasks mit niedrigerer Priorität zu berechnen.

Time-triggered, preemptiv kann ebenfalls mithilfe der Gleichung 4.3 berechnet werden, wobei für alle Tasks $J_i = 0$ gilt und die Offsets genutzt werden, den zeitlichen Versatz der Tasks zueinander zu quantifizieren.

Time-triggered, nicht preemptiv verwendet zusätzlich den Blockierungsfaktor B_i, um die Nichtunterbrechbarkeit von Tasks auszudrücken.

Taskketten mit End-to-End-Deadlines werden, wie in [181] dargestellt, additiv durch Verwendung der Antwortzeiten der einzelnen Tasks einer Kette und der Nachrichten zwischen diesen behandelt. Die Konsequenz ist eine stark vernetzte Analyse aufgrund wechselseitiger Beziehungen zwischen Antwortzeit und Jitter. Dies ist eingehender in Abschnitt 4.4.1 beschrieben. Problematisch ist auch die Analyse bei der Kopplung durch asynchrone Nachrichten, da dies in Kapitel 2.3 nicht berücksichtigt ist und die angegebenen Gleichungen diesen Sachverhalt nicht ausreichend abbilden können.
Um Tasks in einer Komponenten-orientierten Art und Weise analysieren und platzieren zu können, sind sogenannte "lokale" Analysen notwendig, die allerdings nur dann effektiv einsetzbar sind, wenn die allgemeine Form der End-to-End-Deadline zugunsten Task spezifischer Deadlines aufgegeben wird. Da die End-to-End-Deadlines aus dem Entwurfsprozess heraus aufgebaut wurden und i.a. Requirements an das System ausdrücken, sind sie unverzichtbar. Stattdessen sollen lokale, Task spezifische Deadlines aus ihnen abgeleitet werden. Die Vorteile dieser Herangehensweise aus Sicht des Entwicklungsprozesses komplexer Systeme wird in Kapitel 6.2 detailliert diskutiert; hier soll zunächst die Information genügen, dass diese lokalen Deadlines existieren.
Für jede Task $\tau_i \in \mathcal{T}$ und jede Kommunikation $g_j \in \gamma_i$ wird folglich eine lokale Deadline synthetisiert.

$$\Delta_\tau : \mathcal{T} \to \mathbb{N}$$
$$\Delta_\gamma : (\mathcal{T} \times \mathbb{N}) \to \mathbb{N}$$

Die Deadlinesynthese Δ unterliegt Randbedingungen, die dafür sorgen, dass die definierten End-to-End-Deadlines des Tasknetzwerkes eingehalten werden. Aufgrund der

speziellen Behandlung von Taskketten der Länge 1 bzgl. der End-to-End-Deadlines erfolgt hier eine differenzierte Betrachtung der Randbedingung:

$$\forall d_i = (\text{“}\tau_i\text{”}, \lambda) : \Delta_\tau(\tau_i) + \max_{g_j \in \gamma_i}\{\Delta_\gamma(g_j)\} \leq \lambda$$

$$\forall d_i = (\text{“}\tau_1 \ldots \tau_n\text{“}, \lambda) : \sum_{i=1}^{n} \Delta_\tau(\tau_i) + \sum_{i=1}^{n-1} \Delta_\gamma(g_j) + \sigma \leq \lambda, \text{ mit } g_j = (\tau_{i+1}, s) \in \gamma_i$$

Bei Taskketten der Länge 1 wird davon ausgegangen, dass alle ausgehenden Nachrichten parallel versendet werden können. Dies hat zur Folge, dass von den durch Δ_γ erzeugten Deadlines für die Nachrichten die längste Deadline auf Einhaltung der End-to-End-Deadline λ überprüft werden muss.
Im allgemeinen Fall von Taskketten ist zusätzlich ein Synchronisationsfaktor σ zu berücksichtigen, der die Kopplung von Nachrichten zu zeitgesteuerten Tasks innerhalb einer Taskkette abbildet. Dies ist in der Beschreibung der Analyseverfahren in Kapitel 2.3 für Tasksysteme nicht enthalten, wohl aber in der Antwortzeitberechnung von Nachrichten (vgl. Gleichung 4.12). Demzufolge ist der Faktor σ hier nur für die Schedulinganalysen von Tasksystemen explizit zu betrachten. Die folgenden Analyseverfahren von Nachrichten und ihrer Latenzzeit subsumieren diesen Faktor. Der Grund für diese unterschiedliche Behandlung eines sehr ähnlichen Effektes liegt in der Eigenart der Kommunikationsmethoden, bei denen der Effekt des Wartens auf die nächste Invokation (also hier die nächste Möglichkeit zum Versenden der Nachricht) eine inhärente Systemeigenschaft bestimmter Bussysteme darstellt (wir werden dies in Kapitel 4.3 sehen). Diese Wartezeiten gehören damit zur Antwortzeit einer Nachricht und entsprechen in etwa den Preemptionen in der Taskanalyse. Dies gilt für Tasksysteme nicht gleichermaßen, da der Zeitfaktor σ durch die Kopplung von zwei Tasks über eine Nachricht erzeugt wird und nicht eine lokale Eigenschaft einer Task auf einem Prozessorknoten ist.
Der Zeitpunkt des Erreichens einer asynchronen Nachricht einer Task τ_i an einer Nachfolgetask τ_j liegt im schlimmsten Fall (von dem hier auszugehen ist) eine unendlich kleine Zeitspanne nach dem (zeitgesteuerten) Aktivieren der Zieltask τ_j. Die Reaktion auf diese Nachricht kann demnach erst in der nächstfolgenden Invokation erfolgen, die t_j Zeiteinheiten später liegt, also entsprechend der Periode. Es wird dabei davon ausgegangen, dass zeitgesteuerte Tasks streng periodisch sind. Für den Synchronisationsfaktor σ ergibt sich damit für eine Taskkette $d = (\text{“}\tau_1\tau_2 \ldots \tau_n\text{“}, \lambda)$:

$$\sigma = \sum_{i \in \{2,\ldots,n\} : tr_i = TIME \wedge tr_{i-1} = TRIGGER} t_i$$

Mithilfe dieser Bedingungen für die Deadlinesynthese lässt sich nun auch die Einhaltung der End-to-End-Deadline einer Platzierung aus Definition 4.1.3(5) präzisieren: Durch den Einzelnachweis der Einhaltung aller lokalen Deadlines der Tasks und Kommunikationen einer Taskkette kann die gesamte Antwortzeit der betrachteten Taskkette ψ_i^k im Prinzip als Summe der lokalen Deadlines plus Synchronisationsoverhead σ betrachtet werden, also vollkommen analog der Randbedingung für Deadlinesynthesen. Damit ergibt sich also für Taskketten der Länge 1 mit $d_i^k = (\text{“}\tau_i\text{”}, \lambda)$:

$$\psi_i^k = \Delta_\tau(\tau_i) + \max_{g_j \in \gamma_i}\{\Delta_\gamma(g_j)\},$$

sowie für allgemeine Taskketten $d_i^k = (\text{“}\tau_1\tau_2 \ldots \tau_n\text{“}, \lambda)$:

$$\psi_i^k = \sum_{i=1}^{n} \Delta_\tau(\tau_i) + \sum_{i=1}^{n-1} \Delta_\gamma(g_j) + \sigma, \text{ mit } g_j = (\tau_{i+1}, s) \in \gamma_i$$

Die Bedingung $\psi_i^k \leq \lambda$ aus Defintion 4.1.3(5) wird damit allerdings zu einer Vorschrift über die Wahl der lokalen Deadlines $\Delta_\tau(\tau_i)$ und $\Delta_\gamma(g_j)$ abgeschwächt und stellt somit allein kein Kriterium zum Nachweis der Deadline-Einhaltung dar. Vielmehr muss die entsprechende Bedingung innerhalb der Definition der Platzierungsabbildung 4.1.3(5) erweitert werden:

$$\psi_i^k \leq \lambda \wedge \forall \tau_j \in \{\tau_1, \ldots, \tau_n\}, \forall g_{j+1} \in \gamma_j : r_j^{\text{fix}} \leq \Delta_\tau(\tau_j) \wedge r_{g_{j+1}}^{\text{fix}} \leq \Delta_\gamma(g_{j+1})$$

Der zusätzliche Anteil fordert nun neben der korrekten Wahl der lokalen Deadlines auch die Einhaltung dieser durch die Ausführungen der entsprechenden Tasks und Nachrichten einer Taskkette. r^{fix} für Tasks bezeichnet dabei den Fixpunkt der Antwortzeitberechnung gemäß Gleichung 4.3, r^{fix} für Nachrichten ist analog zu betrachten und wird in dem folgenden Abschnitt 4.3 detailliert betrachtet.

Insgesamt lässt sich mit Hilfe dieser Art der Formalisierung erreichen, dass Tasks und Kommunikationen abhängig von ihrer Platzierung in einem verteilten System lokal betrachtet werden können und nicht, wie beispielsweise in [181] die gesamte Taskkette. Gleichwohl schränkt eine Festlegung auf lokale Deadlines natürlich den Verteilungsspielraum etwas ein, da potentiell Lösungen als invalide erkannt werden, die unter der allgemeineren Form der Analyse und Verteilung valide sind. Dem entgegenzuwirken ist Aufgabe der Deadlinesynthese Δ, die entsprechend dem Anwendungsszenario Deadlines einer hohen Güte konstruieren muss. Verfahren, die dies sicherstellen, werden in Kapitel 6.2 eingehend erläutert und bewertet. Problematisch bleiben Kommunikationen, die aufgrund der Platzierung der Kommunikationspartner auf dem selben Knoten in sich zusammenfallen. Um den Anteil der Kommunikationsdeadline an der gesamten End-to-End-Deadline nicht zu verlieren, wird der Betrag der Kommunikationsdeadline der Deadline des Sendertasks zugeordnet, so dass für jede lokale Deadline Δ_τ^* eines Task τ_i innerhalb einer Taskkette $d = (\text{“}\tau_1\tau_2 \ldots \tau_n\text{“}, \lambda)$ gilt:

$$\Delta_\tau^*(\tau_i) = \begin{cases} \Delta_\tau(\tau_i) + \min\limits_{\forall g_j \in \gamma_i} \{\Delta_\gamma(g_j)\} & \textit{wenn } \forall g_j \in \gamma_i : \Pi(\tau_i) = \Pi(\tau_j) \\ \Delta_\tau(\tau_i) & \textit{sonst} \end{cases} \tag{4.4}$$

Die Addition des Minimums aller Kommunikationsdeadlines verschenkt zwar zeitliche Anteile an der End-to-End-Deadline, ist aber notwendig für den Fall, dass τ_i Teil mehrerer Taskketten ist: Die Verwendung des Maximums (oder anderer Varianten) würde ansonsten aufgrund der Konstruktion der lokalen Deadlines die Invariante (5) aus Definition 4.1.3 verletzen.

Damit können Schedulinganalysen von Taskketten effizient lokal pro Task durchgeführt werden und gleichzeitig kann eine hohe Freiheit bei der Platzierungsentscheidung gewahrt werden. Ebenso sind die Zeitanforderungen an die Kommunikationen lokal zu bewerten, wie im folgenden Kapitel ausgeführt.

4.3 Formalisierung von Kommunikation

Mit der Tendenz, eingebettete Echtzeitsysteme zunehmend verteilt zu realisieren kommt der Übermittlung von Nachrichten über ein Steuergerätenetzwerk mit heterogener Topologie zunehmend mehr Bedeutung zu. Die Einhaltung von Deadlines hängt nicht nur von der Antwortzeit von Tasks einer Taskkette ab, sondern wird in gleicher Weise von der Latenzzeit der Nachrichten bestimmt. Simulationstechniken, wie beispielsweise die Restbussimulation (siehe Kapitel 3.3) tragen diesem Umstand nicht ausreichend Rechnung, da sie aufgrund ihrer Unvollständigkeit und Testmusterabhängigkeit nur jeweils einen kleinen Ausschnitt der Wirklichkeit zeigen können. Vielmehr ist an dieser Stelle wie auch bei Tasks eine formale Berechnungsgrundlage notwendig. Erschwert wird dieses Vorhaben durch die ausgeprägte Heterogenität der Netzwerke, die sich in einer Kopplung unterschiedlichster Kommunikationsmedien niederschlägt und weit über die Heterogenität der Schedulingverfahren auf den Prozessorknoten hinausgeht; typische Steuergerätenetzwerke beispielsweise in der Automobildomäne verfügen über bis zu 5 Typen von Bussystemen, die sich nicht nur hinsichtlich ihrer Übertragungsgeschwindigkeit unterscheiden, sondern zudem noch vollkommen unterschiedliche Zugriffsstrategien verwenden. Trotzdem gelingt eine Übertragung der Schedulingtheorie, wie wir sie im letzten Abschnitt dargestellt haben, auf die Nachrichtenübermittlung. Grundlegende Arbeiten auf diesem Gebiet sind von der Universität York vorangetrieben worden (vgl. [181, 182]) und zeigen, dass sowohl Zeit-basierte Systeme, wie beispielsweise TTP[189] als auch Ereignis-basierte Systeme, wie der CAN-Bus[152], auf eine ähnliche Weise formal vollständig behandelt werden können. Im folgenden werden wir diese Arbeiten aufgreifen und sie entsprechend den Anforderungen in verteilten Eingebetteten Systemen adaptieren. Dabei wird abweichend von der einfachen Struktur, wie sie TTP oder CAN bieten, auch auf komplexere Bussysteme eingegangen, wie beispielsweise gemischte Ereignis- und Zeit-basierte Systeme, wie FlexRay[55] oder TTCAN[61]. Hierfür sind die bisher verfügbaren Analysen nicht ausreichend. Zwar existieren Arbeiten zu beispielsweise FlexRay (siehe [143, 145]), jedoch beruhen diese auf vereinfachenden Annahmen, die davon ausgehen, dass Kommunikationsschemata für den FlexRay vor ihrem Einsatz in eine einfache, mit bisherigen Analysemethoden abdeckbare, Form transformiert werden. Dies zerstört allerdings für den industrielle Einsatz entscheidende Vorteile[121] dieses Bussystems. Daher werden im folgenden Analyseverfahren angegeben, die eine akkuratere Echtzeitanalyse von Nachrichten auf diesen komplexen Systemen erlauben.
Wie bei der Analyse von Tasksystemen wird im folgenden die Echtzeitanalyse von Kommunikationen für Ereignis-basierte Nachrichtenübertragung durchgeführt. Dies bedeutet jedoch nicht, dass keine Zeit-basierten Kommunikationsmedien, wie TTP oder FlexRay betrachtet werden. Vielmehr wird mithilfe der im folgenden angegebenen Analysemethoden das komplexe Zusammenspiel zwischen Ereignis-basierten Nachrichten und Zeit-basierten Übertragungsprotokollen abgedeckt und ermöglicht es somit, Platzierungen in bisher nicht dagewesenen Freiheitsgraden des Entwurfes zu konzipieren und analysieren.

Eine Implementierung einer Funktion als verteiltes System von Tasks, die sich mit-

hilfe von Nachrichten gegenseitig aktivieren, macht zusätzlich den Einfluss der Architektur auf die Schedulability des Systems notwendig: Nachrichten müssen an einem Kommunikationsmedium abgesetzt werden und konkurrieren hierbei mit anderen Nachrichten auf dem Prozessorknoten. Es geht folglich nicht nur um die Latenzzeit einer Nachricht auf einem physikalischen Medium, sondern es ist vielmehr auch die Zeit zu beachten, die vor dem eigentlichen Senden der Nachricht verbraucht wird. Verkompliziert wird diese Analyse durch hierarchische Topologien, in denen Nachrichten nicht mehr nur ein Kommunikationsmedium benutzen, sondern womöglich eine ganze Reihe von diesen. Der Zugang zu jedem Medium erfolgt hierbei notwendigerweise über Prozessorknoten, die als Gateway zwischen die Bussysteme geschaltet werden. Konflikte mit anderen Nachrichten treten als zwangsläufige Konsequenz folglich nicht nur am Sende-Knoten sondern auch an allen Zwischenstationen auf und müssen in der Analyse berücksichtigt werden. All dies hängt ganz wesentlich von der Implementierung der Zugriffsmethode an das jeweilige Bussystem ab. Eine typische Architektur dieser Anbindung ist in Abbildung 4.3 dargestellt.

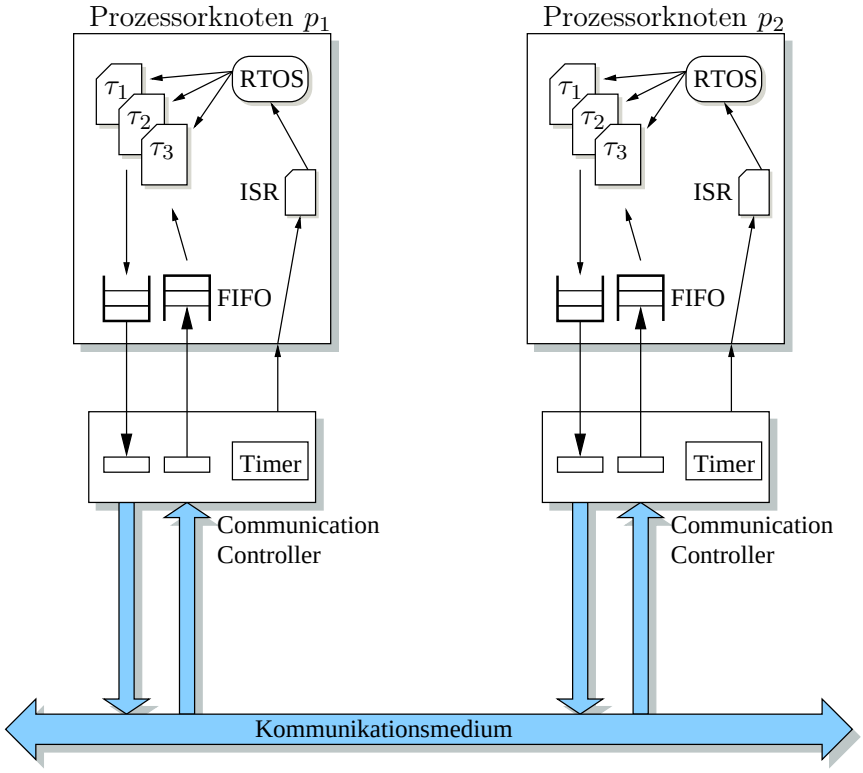


Abbildung 4.3: Typische verteilte Architektur mit nach Prioritäten der Nachrichten sortierten FIFOs in den Prozessorknoten. Die Communication Controller wickeln das Protokoll ab und signalisieren dem Prozessor ankommende Nachrichten.

Echtzeitfähigkeit wird analog zum Taskscheduling durch eine Priorisierung der Nachrichten erreicht. Um zu verhindern, dass nieder priorisierte Nachrichten höher priorisierte ausbremsen, sind die Eingangs- und Ausgangs-Warteschlangen an den Buscontrollern nicht einfache FIFOs, sondern prioritätensortierte Puffer. Tasks, die eine Nachricht versenden müssen, rufen eine Methode des RTOS auf, welche dafür sorgt, dass die entsprechende Nachricht an einer gemäß ihrer Priorität richtigen Stelle im

Zugangs-Puffer eingeordnet wird. Dadurch werden andere Nachrichten potentiell blockiert; ein Effekt, der den Preemptionen in Tasksystemen gleicht. Das Versenden der Nachrichten erfolgt dann autonom über den Buscontroller ohne weiteren Eingriff des Betriebssystems oder der Task. Eingehende Nachrichten werden ebenfalls in ein Eingangs-Puffer einsortiert. Bei Vollständigkeit der Nachricht erfolgt ein Aufruf einer speziellen Interrupt-Service-Routine, die die Nachricht abgreift, den Zieltask auf "bereit" setzt und den Scheduler des Systems aufruft. Timer können zusätzlich notwendig sein, wenn es sich um Zeit-basierte Bussysteme, wie etwa TTP handelt.

Im folgenden wird vorausgesetzt, dass die Zeit für das Einstellen und Auslesen der Puffer-Einträge der Laufzeit der jeweiligen Task zugeschlagen wird, die diese Operation durchführt. Das Füllen und Leeren der Puffer in Richtung des Bus Controllers kann allerdings nicht von den Tasks selbst übernommen werden, da die Controller möglicherweise zum Zeitpunkt des Einstellens nicht bereit sind, eine Nachricht zu übernehmen. Zudem muss bei einem nach Prioritäten sortierten Puffer nicht unbedingt die gerade eingestellte Nachricht auf dem Bus gesendet werden, sondern es können noch höher priorisierte in den Puffer warten. Daher wird für diese Aktion auf jedem Prozessorknoten, der eine sendende Verbindung zu einem Kommunikationsmedium hat, eine synthetische Betriebssystem-Task τ_{OUT}^{OS} installiert. Für ankommende Nachrichten gilt gleiches, d.h. es wird ebenfalls ein Task τ_{IN}^{OS} hinzugefügt. Das Hinzufügen derartiger Systemtasks aufgrund von Nachrichtenverschickung ist erstmals in [171] beschreiben worden. Beide Arten von synthetischen Tasks müssen in der Scheduling-Analyse des Tasksystems beachtet werden. Aufruffrequenzen dieser Tasks ergeben sich aus der Spezifikation des jeweiligen Bussystems. Ausschlaggebend ist die größte Frequenz, mit der Nachrichten eintreffen oder abgesetzt werden können. In den meisten Kommunikationsmedien wird dies die Versendezeit des kürzesten Paketes sein. Aufgrund ihrer Zugehörigkeit zum RTOS Kernel haben sie eine höhere Priorität als Applikationstasks. Optimiert werden kann, wenn ein Medium nur einen lesenden *oder* einen schreibenden Zugriff innerhalb eines Zyklus erlaubt, da dann bekannt ist, dass nur maximal einer der beiden Tasks aktiv sein kann. Geht man davon aus, dass die Ausführungszeiten von τ_{OUT}^{OS} und τ_{IN}^{OS} gleich sind, so kann die Analyse mit nur einem der beiden synthetischen Kommunikations-Tasks rechnen.
τ_{IN}^{OS} darf jedoch nicht mit der Interrupt Service Routine verwechselt werden, die aufgrund einer Nachricht einen neuen Scheduleraufruf initiiert. Da τ_{IN}^{OS} nur zum Entgegennehmen eines Paketes vom Bus Controller zuständig ist, eine Task-initiierende Nachricht aber möglicherweise aus mehreren Paketen zusammengesetzt ist, ist der potentielle Dispatch-Vorgang durch die ISR getrennt vorzunehmen. Dies ist in der Antwortzeitberechnung aus Gleichung 4.3 durch Vorhalten des $\Omega_i(\Pi(\tau_i))$ schon berücksichtigt und wird in Teil II dieser Arbeit betrachtet.

Sind Prozessorknoten mit mehreren Kommunikationsmedien verbunden (wirken also gleichzeitig als Gateways), so muss für jedes angeschlossene Medium ein spezifisches Paar an synthetischen Tasks τ_{OUT}^{OS} und τ_{IN}^{OS} in das Tasksystem dieses Knotens — quasi als Grundlast — injeziert werden. Dies ermöglicht auch die korrekte zeitliche Vorhersage von Nachrichtenverschickung über hierarchische Bussysteme hinweg. Ab-

bildung 4.4 macht dies an einem Beispiel deutlich

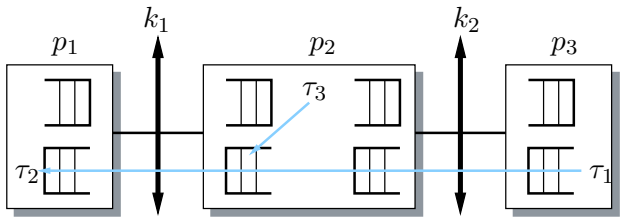


Abbildung 4.4: Nachrichten in einer hierarchischen Topologie müssen Gateway-Knoten nutzen und unterliegen dort wieder potentiell Preemptionen durch höher priorisierte Nachrichten.

Das Weiterleiten von Nachrichten in der Funktion als Gateway hat zudem noch weitergehende Effekte auf die Antwortzeitberechnung von Nachrichten, die von diesem Knoten ausgehen. So können nicht nur Nachrichten von Tasks auf dem betrachteten Knoten den Versand einer anderen Nachricht durch eine höhere Priorisierung verzögern, sondern dies gilt auch für alle durchgeleiteten Pakete. Letztere müssen, ebenso wie lokal entstandene Nachrichten, in die Ausgangs-Puffer am Zugang zum gewünschten Bussystem eingetragen werden. Antwortzeit-Berechnungen, die Preemptionen durch Nachrichten berücksichtigen, müssen diesem Umstand Rechnung tragen.

Anspruch dieses Kapitels ist eine möglichst realitäts-nahe Modellierung für typische Vertreter der Ereignis-basierten und der Zeit-basierten Bussysteme zu finden. Nur wenige notwendige Überapproximationen des zeitlichen Verhaltens sollen dafür Sorge tragen, dass diese Techniken auch in realistischen Systemen eingesetzt werden können. Dazu gehört beispielsweise die Betrachtung von Nachrichtenübertragungen, deren Nutzdaten in Pakete verpackt sind ebenso, wie die Betrachtung des Verhalten von Paketrahmen auf den unterschiedlichen Bussystemen.
Nachrichtenpakete, sogenannte Frames, und deren Grundlagen bzgl. der Echtzeiteigenschaften, sowie prinzipielle Entwurfsmethodiken, wie beispielsweise unterschiedliche Schichtenmodelle der definierten Busprotokolle und deren Auswirkungen auf das Zeitverhalten werden zunächst detailliert am CAN-Bus als Vertreter der Ereignis-basierten Kommunikationsmedien behandelt. Da sich insbesondere die Verfahren der Framebildung und -verschickung sowie die der Einflüsse höherer Schichten der Kommunikationsabwicklung trotz der unterschiedlichen Arbitrierungsansätzen sehr ähneln, wird anschließend bei der Betrachtung des Zeit-basierten Ansatzes lediglich auf die Ausführungen des CAN-Bus verwiesen.

4.3.1 Ereignis-basierte Bus-Systeme — CAN Bus

Ereignis-basierte Bus-Systeme bieten einen Zugang für Nachrichten, der an den *Zeitpunkt des Auftretens* der Nachricht gebunden ist und nicht einem festen vordefinierten Zeitpunkt zugeordnet wird. Analog zu Ereignis-basierten Tasksystemen erfolgt der Zugang zum Bus dabei nach einer Priorisierung der einzelnen Nachrich-

ten. Demzufolge ist für jede Nachricht, die das Ereignis-basierte Kommunikationsmedium nutzt, eine globale und eindeutige Priorität zu bestimmen. Ein typischer und sehr weit verbreiteter[1] Standard aus dieser Klasse von Bus-Systemen ist der CAN-Bus[152].
Das Kürzel *CAN* steht für *C*ontroller *A*rea *N*etwork und definiert ein serielles Übertragungsprotokoll, dass von der Bosch AG 1991 als ein effizientes Protokoll für verteilte Echtzeitanwendungen entwickelt wurde und besonders in der Automobil-Industrie eine weitreichende Anwendung erlebt. Die CAN-Spezifikation in der Version 2.0 erlaubt Übertragungsraten bis zu 1 MBit/s, jedoch sind auch leistungsschwächere Varianten im Einsatz. Neben der Übertragungsrate sind für die im Kontext dieser Arbeit zu betrachtenden Echtzeitanwendungen auch noch die Übertragungsart und deren Overhead zu quantifizieren, so dass im folgenden die CAN-Spezifikation unter genau diesem Gesichtspunkt überblicksartig beschrieben ist. Für genauere Studien des CAN-Busses sei hier auf die Spezifikation [152] verwiesen.

Die Architektur des CAN-Busses gliedert sich in 4 Schichten, die jeweils unterschiedliche Aufgaben auf abstrakten Schnittstellen durchführen können.

- Die **Anwendungsschicht** stellt die Verbindung zu den Nachrichten-absetzenden Tasks eines Echtzeitsystems her
- Die **Objektschicht** führt Nachrichten- und Statusbehandlungen aus und bietet zugleich Nachrichtenfilter an, die die für den angeschlossenen Prozessor bestimmten Nachrichten herausfiltert und andere verwirft.
- Die **Transportschicht** ist für die Nachrichtenübermittlung verantwortlich. Hier werden die Basisoperationen auf den Paketen der Übermittlung vorgenommen. Im Einzelnen sind dies:
  - Fehlereingrenzung
  - Fehlerdetektion und -Signalisierung
  - Nachrichtenvalidierung
  - Nachrichtenbestätigung
  - Arbitrierung
  - Nachrichtenframes
  - Transferrate und -Timing
- Die **Physikalische Schicht** ist verantwortlich für Signallevel und Repräsentation von Bits und das Übertragungsmedium selbst

Aus Sicht der Echtzeitanalyse sind insbesondere die unteren 3 Schichten von Interesse, wobei die Objektschicht als klassische Aufgabe des Betriebssystems an der Schnittstelle zur Anwendung gesehen werden kann. So erfolgt hier im wesentlichen die Filterung der Nachrichten, die notwendig ist, da das CAN-Bus Protokoll ein Mithören aller nicht sendender Komponenten am Bus verlangt und die Objektschicht

[1] Allein bis 1997 sind laut [152] mehr als 50 Millionen CAN-Knoten installiert worden.

entscheiden muss, welche Nachricht auf einem Knoten verworfen wird und welche beispielsweise das Dispatchen eines Tasks erforderlich macht. Beispielhaft sei hier auf das OSEK-Betriebsystem [133] verwiesen, dass die Objektschicht dem Interaction Layer in der OSEK-COM Spezifikation [135] zuordnet. Innerhalb des Interaction Layer sind beispielsweise rund ein Dutzend Funktionen zum Filtern von Nachrichten vorgesehen, um Nachrichten zu verwerfen, die nicht aufgrund ihres Identifikators (vergleiche Abschnitt über den Frameaufbau weiter unten) bereits in der Transportschicht (Datalink Layer im OSEK-Terminus) aussortiert wurden. Aus Sicht der Echtzeitanalyse sind dies Operationen, die in den Nachrichtenprozessen, wie sie im vorherigen Abschnitt eingeführt wurden, angesiedelt sind. Die dafür notwendigen Laufzeiten werden durch Einbeziehen dieser Prozesse in die Scheduling-Analyse berücksichtigt.
Interessanter aus der Perspektive der Echtzeitanalysen ist die Transportschicht, da dort unter anderem das Aufteilen der Nachrichten-Daten in spezifische CAN-Frames vorgenommen wird. Dies hat nicht nur Einfluss auf die Laufzeit der Funktionen der Transportschicht, die beispielsweise wiederum in den Nachrichtenprozessen berücksichtigt werden, sondern beeinflusst zusätzlich die Echtzeitanalyse von partiell preemptiver Nachrichtenübermittlung. Die möglichen Vorgehensweisen und Konsequenzen der unterschiedlichen Nachrichtenpartitionierung wird weiter unten detailliert beschrieben und bewertet.
Innerhalb der Transportschicht werden auch entsprechend der CAN-Spezifikation die notwendigen Protokoll-spezifischen Daten, wie beispielsweise Header, CRC-Werte o.ä., aufgebaut und an die physikalische Schicht weitergeleitet. Auch diese Daten beeinflussen maßgeblich die Echtzeiteigenschaften des CAN-Busses, da sie zusammen mit der physikalischen Schicht die eigentliche physikalische Übertragungslatenz einer Nachricht definieren.
Ebenfalls in der Transportschicht ist die Arbitrierung des Busses angesiedelt und damit die zentrale Zugangskontrolle, die die für eine Echtzeitanalyse von partiell preemptiver Nachrichtenübermittlung notwendigen Grundlagen liefert.

Da die Objektschicht nur mittelbaren Einfluss auf die Echtzeitanalyse hat, wird auf eine detailliertere Beschreibung an dieser Stelle verzichtet. Stattdessen werden Transportschicht und physikalische Schicht hier genauer betrachtet.

CAN-physikalische Schicht

Die physikalische Schicht definiert den Zugriff auf den Bus so, dass mehrere Prozessorknoten gleichzeitig auf den Bus schreiben können, ohne dass es zu technischen Defekten (elektrischer Kurzschluss beispielsweise) kommt.
Die Signallevel des Busses sind in zwei unterschiedliche Werte aufgeteilt: dominante und rezessive Bits. Das dominante Bit dominiert auf der Leitung des Busses und überschreibt den Wert des rezessiven Bits. Analog dem elektrischen Aufbau von wired-AND-Schaltungen wird also bei gleichzeitigem Schreiben von dominanten und rezessiven Bits stets das dominante Bit auf dem Bus zu lesen sein. Hierfür wird das rezessive Bit mit logisch 1 (High-Pegel) und das dominante Bit mit logisch 0 (Low-Pegel) ausgelegt.

Alle angeschlossenen Teilnehmer lesen parallel zu ihrem Schreibzyklus den Bus ständig und können durch einen Vergleich der geschriebenen und gelesenen Werte feststellen, ob ein ggf. geschriebenes rezessives Bit überschrieben wurde oder erhalten blieb. Dieser Mechanismus wird später zur Arbitrierung und Quittierung von Nachrichten in der Transportschicht benutzt.

Eine weitere aus Sicht der Echtzeitanalyse wesentliche Eigenschaft ist die Verwendung des sogenannten Bit-Stuffing: Um eine gewisse Synchronität der angeschlossenen Knoten erhalten zu können, wird bei langen Phasen ohne Pegelwechsel auf der Leitung ein künstliches Bit eingefügt, dass einen Pegelwechsel darstellt. In der CAN-Bus Spezifikation wird definiert, dass nach jeweils 5 gleichen Bits ein Stuffing-Bit eingeführt wird. Dies ist bei der Berechnung der Latenzzeiten von Nachrichten zu beachten. Allerdings gilt dieses Bit-Stuffing nicht für alle Teile der Übertragung, insbesondere ist davon die Bus-Idle-Phase (siehe unten) nicht betroffen.
Die physikalische Schicht stellt damit die grundsätzlichen Übertragungseigenschaften zur Verfügung, die eine konkurrierende Arbitrierungsphase verschiedener Nachrichten ermöglicht. Die Art der Arbitrierung und der Aufbau der für diese Arbeit wesentlichen CAN-Pakete ist Aufgabe der Transportschicht.

CAN-Transportschicht

Nachrichten haben beim CAN-Bus global eindeutige Prioritäten, die durch einen Identifikator der Nachricht festgelegt sind. Je niedriger der Identifikator, desto höher ist die Priorität der Nachricht. Zur Erstellzeit des Systems ist sicherzustellen, dass diese Prioritäten eindeutig sind (vgl. auch die Definition 4.1.4 der Prioritätsabbildung für Tasks).
Zu Beginn jeder Übertragungsphase findet zunächst eine Arbitrierung statt, in der entschieden wird, welcher der Sende-bereiten angeschlossenen Knoten seine Nachricht auf dem Bus übertragen darf. Hierzu legen alle Knoten gleichzeitig Bit für Bit ihren Identifikator auf den Bus. Durch den oben beschriebenen Mechanismus kann nun jeder Knoten den geschrieben Wert mit dem aktuellen Buszustand vergleichen. Sollte ein rezessiver Wert eines Teilnehmers durch einen dominaten überschrieben worden sein, so nimmt dieser Teilnehmer nicht mehr weiter an der Arbitrierung teil. Abbildung 4.5 macht dies beispielhaft deutlich[1].

Nach Beendigung der Arbitrierungsphase kann die Übertragungsphase beginnen. Hierfür sind in der CAN-Spezifikation verschiedene Frame-Formate angegeben, von denen die für die Echtzeitanalyse wichtigen im folgenden betrachtet werden sollen. Bemerkenswert ist noch, dass die Arbitrierung über die Identifikatoren selbst schon Teil der Frame-Versendung auf dem Bus ist, so dass alle lesenden Knoten über diese eindeutige ID auch die spezifische Kennung der dann folgenden Nachricht vorliegen haben und entscheiden können, ob sie an die Objektschicht weitergeleitet werden soll.

[1] Man beachte, dass dominante Bits durch eine logische 0 übertragen werden, so dass im angegebenen Beispiel tatsächlich die Nachricht mit der kleinsten ID die höchste Priorität und damit den Zuschlag erhält.

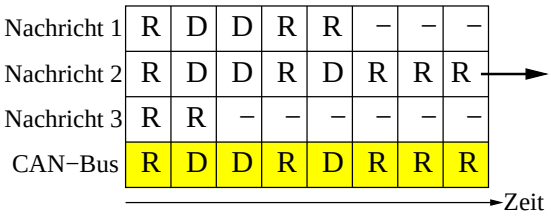


Abbildung 4.5: Arbitrierung des CAN-Busses auf der Basis von rezessiven und dominaten Pegeln und eines eindeutigen Identifikators. Nachricht 1 und 3 nehmen ihre Anfrage jeweils vom Bus, sobald der geschriebene Wert nicht mehr dem Buszustand entspricht.

Nachrichten werden, wie schon erwähnt, in sogenannte Frames verpackt, die als atomare Einheit über das Bussystem verschickt werden. Weitere Arbitrierungsphasen schließen sich erst an, sobald der sendende Knoten eine End-of-Frame-Kennung (*EOF*) (dargestellt durch 7 aufeinanderfolgende rezessive Bits) versendet oder der Bus sich in der Bus-Idle-Phase (oder Interframe spacing) befindet (beliebig viele aufeinanderfolgende rezessive Bits nach einer EOF). Der erste Knoten, der eine Arbitrierung vornehmen möchte, sendet das sogenannte Start-of-Frame-Bit auf den Bus, dargestellt durch ein dominantes Bit. Alle Knoten, die ebenfalls eine Arbitrierung durchführen wollen, müssen sich an diesem Bit synchronisieren oder aber auf eine erneute EOF-Kennung warten.

Die CAN-Spezifikation kennt 6 verschiedene Arten von Frames:

- **Data Frame** dient zum Übertragen von Daten
- **Extended Data Frame** dient ebenfalls der Übertragung von Daten, verfügt aber über einen erweiterten Frame-Header, der einen zusätzlichen Identifikator transportieren kann. Diese zusätzliche ID kann nicht für die Arbitrierung benutzt werden, hilft aber Nachrichten der gleichen Kennung voneinander zu unterscheiden.
- **Remote Frame** ähnelt einem Data Frame, enthält jedoch keine Nutzdaten, sondern fordert eine Übertragung von einem anderen Knoten an. Dazu wird ein bestimmtes Bit (das sogenannte Remote Transmit Request *RTR*) rezessiv besetzt, während dieses Bit bei korrespondierenden Data Frames dominant ausgelegt ist. Dieser Mechanismus gewährleistet, dass Remote Frames unterdrückt werden, wenn der dazugehörige passende Data Frame übertragen werden soll. Zu diesem Zweck ist das Versenden des RTR Teil der Arbitrierung, die damit also aus ID- und RTR-Übertragung besteht.
- **Overload Frame** dient dazu, das Senden von Daten oder Remote Frames zu verzögern. Maximal zwei Overload Frames können generiert werden.
- **Error Frame** werden im Fehlerfall gesendet und besteht aus einem Error Flag, das entweder 6 dominante oder 6 rezessive Bits enthält. Diese 6 Bits sind nicht mit Bit-Stuffing ausgestattet und verletzen damit eine inheränte

Protokolleigenschaft des CAN-Bus. Angeschlossene Knoten können diese Verletzung leicht detektieren und bemerken, dass ein Fehler vorliegt.

Von dieser Liste der Frames werden in den folgenden Echtzeitanalysen nur die Data Frames verwendet, da sie die hier betrachteten Echtzeiteigenschaften maßgeblich definieren. Remote Frames werden nicht betrachtet, da diese Art der Nachrichtenübertragung blockierende Kommunikationen vorraussetzen und dies im Rahmen dieser Arbeit nicht behandelt wird. Als Referenz sei hier auf die Arbeiten in [171] verwiesen.
Error Frames und Overload Frames werden bei Fehlerfällen benutzt. Auch dies wird im Rahmen dieser Arbeit nicht weiter betrachtet; Es wird hier von einer fehlerfreien Übertragung ausgegangen. Abschließende Bemerkungen zur Berücksichtigung von fehlerhaften Kommunikationen sind lediglich im Anschluss an den nächsten Abschnitt zu finden.

Folglich verdienen die Data Frames eine genauere Betrachtung. Data Frames können eine begrenzte Anzahl (maximal 8 Byte) Nutzdaten übertragen. Sowohl die Daten als auch die Informationen des Headers werden durch spezielle fehlererkennende Maßnahmen abgesichert (CRC-Sequenzen). Beispielhaft soll in Abbildung 4.6 der Standard-Data-Frame betrachtet werden (Extended Data Frames verfügen lediglich über eine rezessive Identifier Extension *IDE* und anschließend eine zusätzliche 18-bit breite ID mehr):

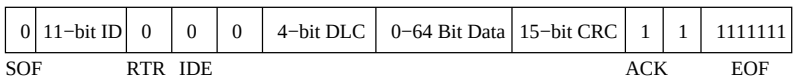


Abbildung 4.6: Aufbau des Data Frame. Identifier und RTR dienen der Arbitrierung, DLC bestimmt die Anzahl der Nutzdaten im Feld data.

Das Data Length Code *DLC* gibt die Anzahl der Nutzdatenbytes an, die von 0 bis 8 variieren kann. Die 15-Bit breite CRC-Sequenz dient der Fehlererkennung. Darauf folgt ein rezessiver Begrenzer und darauf das rezessive ACK Bit. Dieses dient der Quittierung des korrekten Empfangs der Nachricht: Zumindest einer der lesenden Knoten muss an dieser Stelle den korrekten Empfang mit einem dominanten Bit bestätigen.
Damit sind alle für die unter den angegebenen Randbedingungen durchzuführende Echtzeitanalysen von Nachrichten auf dem CAN-Bus gegeben.

Echtzeitanalysen von CAN-Nachrichten

Das zeitliche Verhalten von Nachrichten auf einem Kommunikationssystem hängt zunächst von den spezifischen Daten des Mediums ab. Diese sind in der Formalisierung der Architektur eines verteilten eingebetteten Systems gemäß Definition 4.1.1 in der Abbildung $\kappa(k)$ zu finden. Neben der Übertragungsrate υ_k sind dort weitere Faktoren, wie Größe des Headers und Anzahl der Nutzdatenbytes eines Frames sowie ggf. Bit-Stuffing-Distanz (siehe beispielsweise physikalische Schicht des CAN-Bus) anzugeben.

Auf den CAN-Bus übertragen bedeutet dies für $\kappa(k_{CAN}) = (h_k, \eta_k, \upsilon_k, \beta_k)$:

$$\begin{aligned} h_k &= 43Bits \\ \eta_k &= 8Bytes \\ \upsilon_k &= 1 \cdot 10^6 Bit/s \\ \beta_k &= 5Bit \end{aligned}$$

Die Übertragungsdauer eines Frames auf dem Kommunikationsmedium $k \in K$ lässt sich dann folgendermaßen berechnen:

$$\rho_k = \frac{h_k + 8 \cdot \eta_k + \chi_k}{\upsilon_k} \text{ mit } \kappa(k) = (h_k, \eta_k, \upsilon_k, \beta_k) \tag{4.5}$$

χ_k quantifiziert den maximalen zeitlichen Overhead einer Paketübertragung, der sich aus Synchronisationsphasen und Kodierungsmaßnahmen, wie beispielsweise Bit-Stuffing, zusammensetzt. Für den CAN-Bus gemäß Spezifikation 2.0[152] gibt es den folgenden Overhead:

- Interframe spacing, bestehend aus 3 rezessiven Bits
- Bus Idle kann aus minimal 0 Bits bestehen, so dass direkt nach dem Interframe spacing eine neue Nachricht auf den Bus gelegt werden kann
- Vom SOF-Bit bis zum CRC-Delimiter eines Frame wird Bit-Stuffing verwendet, d.h. im schlimmsten anzunehmenden Fall muss davon ausgegangen werden, dass nach jeweils 5 übertragenen Bits des Frame ein zusätzliches Stuffing-Bit verwendet wird.

Mithilfe dieser Informationen kann der Übertragungsoverhead χ_k für eine Nachricht berechnet werden. Dabei spielt für das Bit-Stuffing die Anzahl der zu übertragenden Nutzdatenbytes und die Größe des Headers eines Frame eine Rolle. Bei der Größe des Headers ist zu beachten, dass laut CAN-Spezifikation das ACK-Bit sowie die 7 EOF-Bits nicht mit Bit-Stuffing ausgestattet sind, ebensowenig wie der Interframe Space. Letztlich ergibt sich für einen Frame, der alle Nutzdatenbytes verwendet, im schlimmsten anzunehmenden Fall folgender Wert für χ_k:

$$\chi_k = 3 + \left\lfloor \frac{h_k - 8 + 8 \cdot \eta_k}{\beta_k} \right\rfloor \tag{4.6}$$

Nachrichten $g_t = (\tau_t, s_t)$ haben eine festgelegte (oder aber im Sinne einer worst-case Analyse überapproximierte) Anzahl an Datenbytes, die kommuniziert werden müssen. Diese Anzahl muss nicht notwendigerweise gleich der Anzahl der Nutzdaten eines CAN-Frames sein. Zwei unterschiedliche Fälle können hierbei auftreten: Die Nachricht ist sehr klein ($s_t \ll \eta_k$) oder die Nachricht ist größer als die in einem Frame zur Verfügung stehenden Nutzdaten ($s_t > \eta_k$). Im ersteren Fall entsteht ein sehr hoher Overhead durch den Frame, wenn jede Nachricht in einem eigenen Frame verschickt wird. Hier bietet es sich an, gleichartige Nachrichten (beispielsweise Nachrichten mit gleicher Periodizität) gemeinsam in einen Frame zu verpacken. Für den CAN-Bus ist dies in [154] exemplarisch durchgeführt worden, während [143] die Frage der Frame-Partitionierung von Nachrichten für gemischt Event-/Zeit-basierte

Systeme durchführt. Es gibt also Arbeiten auf diesem Gebiet, gleichwohl hat eine Zusammenfassung von Nachrichten in einem Frame aber auch nicht unerhebliche Einflüsse auf die Scheduling-Analysen, da zum einen die Frame-Größen auch für Frames einer Nachricht stark differieren können, und dies insbesondere dynamische Änderungen sein können[1]. Zum anderen hat das Sammeln von Nachrichten in bestimmten Frames eine andere Latenzzeit von kausal abhängigen Tasks (und damit von Taskketten insgesamt) zur Folge, weil die aufzusammelnde Nachricht plötzlich von der Verschickzeit anderer Nachrichten abhängt. Damit ist dies nur bei Zeitbasierten Systemen praktikabel und wird im Rahmen dieser Arbeit nicht weiter betrachtet[2]. Aus den obigen Gründen ist diese Art der Zusammenfassung von Nachrichten in realen Entwurfsprozessen nicht praktikabel und wird zumindest beim Entwurf der Elektronikkomponenten in der Automobilindustrie weitgehend vermieden (vergleiche auch [121]).
Für den Fall, dass $s_t > \eta_k$ gilt, ist die Nachricht auf mehrere Fragmente zu verteilen. Auch hier sind unterschiedliche Varianten denkbar, beispielsweise könnte es von Vorteil für die Schedulability des CAN-Bus-Systems sein, wenn lange Fragmente von niedrig priorisierten Tasks in mehrere kürzere Fragmente aufgeteilt werden. Dies würde die Blockierung von höher priorisierten Tasks auf dem Medium abschwächen (zur Blockierung siehe unten), allerdings ist dieser Effekt aufgrund der wenigen Nutzdatenbytes eines CAN-Frames nicht besonders gravierend[3]. Zudem entsteht auch hier der Analyse-verkomplizierende Effekt vieler unterschiedlicher Frame-Längen für Frames einer Nachricht[4] .
Aus diesen Gründen wird hier vereinfachend davon ausgegangen, dass für jede Nachricht mindestes 1 Frame verwendet wird und das die minimale Anzahl an Frames benutzt wird. Die Anzahl der Pakete einer Nachricht auf einem Kommunikationsmedium $k \in K$ berechnet sich demnach durch:

$$c_{g_t}^k = \left\lceil \frac{s_t}{\eta_k} \right\rceil \text{ mit } \kappa(k) = (h_k, \eta_k, \upsilon_k, \beta_k) \wedge g_t = (\tau_t, s_t) \tag{4.7}$$

Zwar ist die Anzahl der pro Nachricht zu verschickenden Frames damit eindeutig bestimmbar, jedoch unterscheidet sich der letzte Frame in seiner Länge potentiell von vorhergehenden, da die Größe der Nachricht (also die Anzahl der Nutzdatenbytes) nicht notwendigerweise ein ganzzahliges Vielfaches des Datenfeldes eines CAN-Frames sein muss. Eine erhöhte Genauigkeit kann hierbei geliefert werden, wenn in den später folgenden Antwortzeitberechnungen für Nachrichten dieser letzte Frame gesondert behandelt wird. Ausschlaggebend für eine Sonderbehandlung ist die Übertragungsdauer eines Frame auf dem Bussystem ρ_k, die gemäß Gleichung 4.5 für Frames bestimmt wurde, die alle Nutzdatenbytes verwendet. Frames mit einer

[1] Beispielsweise könnte eine Nachricht mit einer Periode t_1 zusammen mit einer Nachricht der Periode $t_2 = 2t_1$ versendet werden. Die Framegrößen würden dann alternierend differieren.

[2] Im übrigen ist die Frame-Partitionierung wiederum eine eigenständige Optimierungsaufgabe, die eine entsprechende Modellierung notwendig macht.

[3] Der Anteil der Nutzdaten eines CAN-Frames ist selbst bei Nutzung aller 8 Datenbytes gerade einmal 47%. Eine Aufteilung der Daten in kleinere Pakete würde also einen kaum nennenswerten Effekt erzielen.

[4] Auch dieses Vorgehen verlangt nach einem Optimierungsverfahren, dass wiederum in das Optimierungsverfahren der Platzierung integriert werden müsste.

kürzeren Länge verbrauchen demnach nur einen berechenbaren Anteil der Latenzzeit ρ_k. Dieser Anteil wird im folgenden durch den Faktor $\tilde{c}^k_{g_t}$ für eine Nachricht $g_t = (\tau_t, s_t)$ berechnet.

$$\tilde{c}^k_{g_t} = \frac{h_k + 8 \cdot N^k_{g_t} + 3 + \left\lfloor \frac{h_k - 8 + 8 \cdot N^k_{g_t}}{\beta_k} \right\rfloor}{\upsilon_k} \cdot \frac{1}{\rho_k} \text{ mit } N^k_{g_t} = s_t - \left\lfloor \frac{s_t}{\eta_k} \right\rfloor \cdot \eta_k \quad (4.8)$$

$\tilde{c}^k_{g_t} + c^k_{g_t}$ stellt also aus Sicht der Antwortzeitberechnung eine nicht-ganzzahlige Paketanzahl dar, die mit ρ_k multipliziert die schlimmste anzunehmende Latenzzeit auf einem CAN-Bus mit den in κ angegebenen Leistungsdaten errechnet.

In prioritäten-gesteuerten Versendemechanismen, wie der CAN-Bus es anbietet, kann die Verschickung einer Nachricht durch andere Nachrichten aufgehalten werden. Dies geschieht analog zu Unterbrechungen im Scheduling von Tasks auf einem Prozessorknoten. Zur Berücksichtigung dieses Effektes in der Berechnung der Übertragungsdauer einer Nachricht sind zwei Faktoren ausschlaggebend: Zum einen muss bekannt sein, welche anderen Nachrichten auf dem für die Berechnung der Antwortzeit einer betrachteten Nachricht zu analysierenden Kommunikationsmedium angesiedelt sind. Zum anderen ist die Beziehung der Nachrichten untereinander wichtig, also welche Nachricht kann die betrachtete Nachricht blockieren. Letzteres wird, analog zum Task-Scheduling und konform zum Arbitrierungsverfahren des CAN-Busses, durch eindeutige, globale Prioritäten determiniert.
Zunächst bezeichne $\breve{C}_k$ die Menge der Nachrichten, die durch die Kommunikationsabbildung Γ auf ein Kommunikationsmedium k platziert werden:

$$\breve{C}_k = \left\{ g_i \mid \forall \tau_j \in \mathcal{T},\ \forall g_i \in \gamma_j :\ \Gamma(\tau_j, g_i) = \text{“}\ldots k \ldots\text{“} \right\}$$

Da diese Menge der konkurrierenden Nachrichten global aus dem gesamten Tasksystem konstruiert wird, enthält sie automatisch auch diejenigen Nachrichten, die das betrachtete Medium k nur als Zwischenmedium benutzen und die angeschlossenen Knoten als Gateway-Knoten nutzen.
Bei der Analyse eines Kommunikationsmediums k bzgl. der darauf allokierten Nachrichten kann nun nur anhand einer Prioritätenbeziehung der Nachrichten untereinander der jeweilige Blockierungsfaktor berechnet werden. Prioritäten für Nachrichten werden wiederum analog dem Vorgehen bei Tasks auf einem Prozessorknoten gebildet:

Definition 4.3.1 (Prioritätsbildung für Nachrichten). *Formal ist die Prioritätsbeziehung für Nachrichten g_i und g_j eine partielle Abbildung $\phi : \left(\mathcal{T} \times \mathbb{N}\right) \times \left(\mathcal{T} \times \mathbb{N}\right) \to \{0, 1\}$ mit den folgenden Eigenschaften:*

- *Eindeutigkeit der Priorität: $\phi(g_i, g_j) = 0 \leftrightarrow \phi(g_j, g_i) = 1$*
- *Transitivität der Prioritätsrelation: $\phi(g_i, g_j) = 1 \wedge \phi(g_h, g_i) = 1 \rightarrow \phi(g_h, g_j) = 1$*

Es wird die Schreibweise $\phi_{g_i} < \phi_{g_j}$ verwendet, wenn gilt: $\phi(g_i, g_j) = 0$.

Beachtenswert ist in diesem Zusammenhang, dass Nachrichten nicht pro Kommunikationsmedium mit einer jeweils lokalen Priorität versehen werden, sondern das dies

global geschieht. Dies ist hier eine pragmatische Vorgehensweise, die im wesentlichen von der Synthese von sublokalen Deadlines (vgl. Kapitel 4.4.2) beeinflusst wird.
Mithilfe der nun definierten globalen Priorität von Nachrichten kann die Antwortzeit einer Nachricht g_t mit $\tau_s \xrightarrow{g_t} \tau_t$ in Beziehung zu ihrer Deadline $\Delta_\gamma(g_t)$ gesetzt werden. Dies erfolgt vollkommen analog zur bekannten Scheduling-Analyse.

$$r^n_{g_t} \leq \Delta_\gamma(g_t) \text{ mit}$$

$$r^n_{g_t} = \left(c^k_{g_t} - 1\right) \cdot \rho_k + \tilde{c}^k_{g_t} \cdot \rho_k + B_{g_t} + I^k_{g_t}(r^n_{g_t}) \cdot \rho_k \tag{4.9}$$

Zwecks genauerer Vorhersageergebnisse der Antwortzeitberechung wird der letzte Frame einer Nachricht gemäß Gleichung 4.8 behandelt. Alle übrigen Frames der Nachricht rufen eine worst-case Latenz von ρ_k hervor.
Der Blockierungsfaktor B_{g_t} wird durch einen belegten Bus hervorgerufen, dessen Übertragung abgewickelt werden muss, bevor die Nachricht g_t eine Arbitrierung versuchen kann. Dies liegt daran, dass ein Frame, sobald er mit der Übertragung begonnen hat, nicht unterbrochen werden kann. Trotzdem also die Nachricht g_t eine höhere globale Priorität hat als die gerade übertragende Nachricht, erfährt g_t eine Blockierung. Abbildung 4.7 macht dies an einem Beispiel deutlich.

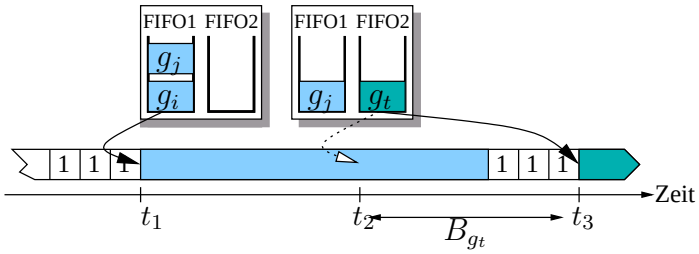


Abbildung 4.7: Blockierung durch bereits begonnende Übertragung mit $\phi_{g_j} < \phi_{g_i} < \phi_{g_t}$: g_t wird zum Zeitpunkt t_2 in die FIFO2 eingetragen, der Frame von g_i wird jedoch gerade übertragen.

Während die Nachricht g_i aus FIFO1 auf Grund der Detektion des Bus Idle Frame direkt zum Zeitpunkt t_1 seine Arbitrierung starten kann und auch den Zuschlag erhält, gilt dies für Nachricht g_t in FIFO2 nicht. Der Zeitpunkt ihres Auftretens t_2 liegt zu einem Zeitpunkt, da g_i bereits den SOF gesetzt hatte und sich in der Übertragungsphase befindet. g_t kann mit der Arbitrierung des Busses erst starten, sobald er sich auf ein Busy Frame synchronisiert hat (t_3). Im schlimmsten Fall liegt t_2 gerade eine Bitübertragung nachdem g_i das SOF-Bit gesendet hat, also gilt für den Blockierungsfaktor (um eine Bitübertragung $\frac{1}{\upsilon_k}$ überapproximiert)

$$B_{g_t} = \rho_k.$$

Neben dieser Art der Blockierung können Nachrichten auch durch andere Nachrichten mit einer höheren Priorität ϕ_{g_i} blockiert werden. Diese Art der Blockierung, in Anlehnung an das Task-Scheduling “Interferenz” genannt, tritt zum einen in den Prioritäten-sortierten FIFOs auf, die am Zugang zu den Bus-Controllern angebracht sind, in dem neuere, aber höher priorisierte Nachrichten ältere nach hinten

drängen. Zum anderen kann während der Arbitrierungsphase Nachrichten mit einer niedrigeren Priorität der Zugang zum Bus verweigert werden (Tindell hat in [181] gezeigt, dass beide Effekte äquivalent zu einer globalen, Prioritäten-sortierten FIFO-Zugangsstrategie sind und somit die Interferenzberechnung identisch der Interferenzberechnung bei der Scheduling-Analyse von Tasks ist).
Der Interferenzfaktor $I_{g_t}^k$ einer Nachricht g_t auf dem Kommunikationsmedium k gibt an, wieviele Frames duch höher priorisierte Nachrichten zum betrachteten Zeitpunkt berücksichtigt werden müssen:

$$I_{g_t}^k(r_{g_t}^n) = \sum_{\forall g_i \in \breve{C}_k : \phi_{g_i} > \phi_{g_t}} \left(\left\lceil \frac{r_{g_t}^n + J_{g_i}}{t_{g_i}} \right\rceil \cdot \left(\left(c_{g_i}^k - 1 \right) + \tilde{c}_{g_i}^k \right) \right) \tag{4.10}$$

Dabei dürfen nur diejenigen Nachrichten betrachtet werden, die auf dem betrachteten Kommunikationsmedium allokiert sind, also entsprechend alle Tasks aus $\breve{C}_k$ mit einer höheren Priorität. Wiederum werden hier die nicht-ganzzahligen Frames $\tilde{c}_{g_i}^k$ für eine exaktere Vorhersage verwendet.

Letztlich kann mittels der oben dargestellten Analysemethode eine exakte Antwortzeitberechnung für Nachrichten auf einem CAN-Bus angegeben werden. Diese Antwortzeit geht allerdings davon aus, dass es zu keinem Übertragungsfehler kommt. Techniken, die Ausfallraten formal zu berücksichtigen, sind nicht bekannt oder unpraktikabel. So wäre es zwar möglich, bei einer gegebenen tolerierbaren Ausfallwahrscheinlichkeit (hier also die Verletzung einer Deadline) die für diese Anzahl wahrscheinliche Mehrfachübertragung im schlimmsten Fall zu quantifizieren, aber der dabei notwendigerweise entstehende Overhead wirkt sich im allgemeinen so drastisch auf die Performanz des Systems aus, dass es für praktische Systeme nicht einsetzbar ist. Es gibt eine Reihe von Arbeiten, die sich diesem Thema unter Berücksichtigung bestimmter Fehlermodelle widmet. Dabei wird die Beobachtung ausgenutzt, dass Fehler in der hier betrachteten Anwendungsdomäne (vorwiegend Automobiltechnik) weitgehend von sogenannten ElektroMagnetischen Interferenzen (EMI) verursacht werden. Übertragungsfehler dieser Art treten zufällig auf und unterliegen einer Poisson Verteilung, mit der es möglich ist, realistischere Antwortzeitberechnungen von Nachrichten durchzuführen (unter der Annahme, dass ab einer gewissen Grenzwahrscheinlichkeit Ausfälle zu tolerieren sind; ein in der Sicherheitsanalyse typisches Verfahren). Beispielhaft seien an dieser Stelle die Arbeiten in [24] und [128] genannt, die diesen Effekt in die bekannte Antwortzeitanalyse integrieren.
Ein pragmatischer Ansatz, wie er beispielsweise in [179] eingesetzt wurde, ist die idealisierte Analyse der Antwortzeiten im Nachhinein um am physikalischen System gemessene durchschnittliche Ausfallraten anzureichern. Dieser Ansatz kann allerdings nur Freiräume für Ausfälle schaffen und ist nicht in der Lage auf formal nachvollziehbaren Gründen Deadlineüberschreitungen zu vermeiden. Auswirkungen auf die Einhaltung von End-to-End-Deadlines in Taskketten sind allerdings noch nicht untersucht worden. So könnte es beispielsweise tolerierbar sein, dass Deadlines von Nachrichten innerhalb einer Taskkette durch fehlerhafte Übertragung verletzt werden, dieser Effekt jedoch durch weitere Nachrichten und Tasks insgesamt aufgefangen wird. Dieser Ansatz könnte sich möglicherweise als ein weiteres Feld der Optimierung in der Platzierung erweisen, wobei aber grundsätzlich der Einfluss einer

lokalen Deadlineüberschreitung auf den Schedule des Nachrichtensystems auf dem betreffenden Medium untersucht werden muss. Insgesamt ist in diesem interessanten Gebiet noch viel Forschungsarbeit zu leisten, die aber im Rahmen dieser Arbeit nicht weiter vertieft werden soll.

4.3.2 Zeit-basierte Bus-Systeme — Time-Triggered Architekturen

Zeit-basierte Bussysteme beruhen auf dem sogenannten TDMA-Prinzip (Time Devision Multiplex Access)[78][1], dass besagt, dass jede Nachricht streng zyklisch zu einem bei Systemerstellung festgelegten Zeitpunkt auf dem Bus gesendet werden darf. Alle Knoten außer dem Sender horchen auf dem Bus und greifen die Nachricht ggf. ab. Abbildung 4.8 macht dies an einem Beispiel deutlich.

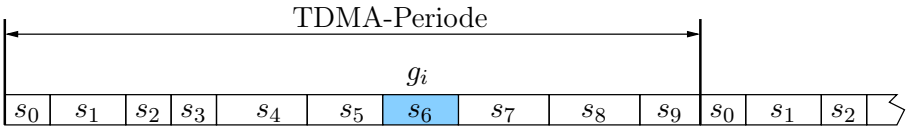


Abbildung 4.8: Prinzip des TDMA-Verfahrens am Beispiel einer Nachricht g_i, die einem Slot s_6 zugordnet ist.

Zeitabschnitte, in denen einer Nachricht Zugriff auf den Bus gewährt wird, werden *Slots* genannt. Die durch die TDMA-Definition des Systems festgelegte Abfolge von Slots wird TDMA-Round genannt und wiederholt sich zyklisch gemäß einer durch die Länge der jeweiligen Slots festgelegten konstanten Periode. Wieviele Slots in einer TDMA-Round existieren dürfen, hängt wesentlich vom verwendeten Protokollstandard ab und differiert zwischen 64 in TTP[189] und 1028 in FlexRay[55]. Diese beiden Protokolle stellen zwei z.B. in der Domäne der Automobilindustrie weit verbreitete Standards dar und sollen exemplarisch die für TDMA-Verfahren notwendigen Analysen illustrieren.
In statischen Systemen, wie sie im Kontext dieser Arbeit betrachtet werden[2], ist jede Nachricht eindeutig einem sendenden Prozessorknoten zugeordnet. Im folgenden werden daher auch die Slots einem Prozessorknoten zugordnet.

Sei S die Menge der eindeutigen Slots, die einem dedizierten Prozessorknoten exklusiven Zugriff auf die Resource "Bus" gewährt. Dann ist eine TDMA-Round definiert durch eine zyklisch wiederkehrende Sequenz von Slots. Für jeden Slot $s_i \in S$ ist eine Abbildung ζ definiert, die die Zuordnung jeweils eines Prozessorknoten zu diesem Slot definiert. Eine weitere Abbildung ψ definiert die Länge des jeweiligen Slot und damit implizit der TDMA-Round insgesamt.

[1] Bekannt geworden ist dieses Verfahren in erster Linie durch den weit verbreiteten Einsatz in der Sprachübermittlung bei Telekommunikationsanlagen.

[2] Gemeint sind hier Systeme, bei denen die Tasks keiner Migration unterliegen, d.h. immer fest genau einer ECU zugeordnet sind.

Definition 4.3.2 (TDMA-Round). *Gegeben sei eine Menge S von Slots. Dann ist eine TDMA-Round $TDMA \in S^{\leq N}$ eine Sequenz von Slots, die zyklisch wiederholt wird. Für jede TDMA-Round sind die folgenden Abbildungen definiert:*

1. $pr : \mathbb{N} \times \mathrm{S}^{\leq \mathrm{N}} \rightarrow \mathrm{S}$ *eine Projektion über den Wörtern der Sprache S liefert, für die $pr(j, TDMA)$ den j-ten Slot aus $TDMA$. $pr(j, TDMA)$ ist definiert durch:*
   $pr(j, TDMA) = s_j$ mit $TDMA = \text{“}s_1 \ldots s_n\text{“} \wedge 1 \leq j \leq n$.
   Schreibweise für $pr(j, TDMA)$ sei $pr_j(TDMA)$.

2. $\psi_{TDMA} : S \rightarrow \mathbb{N}$ *ordnet jedem Slot $s \in \bigcup_{j \in \{1,\ldots,|TDMA|\}} pr_j(TDMA)$ aus der TDMA-Round $TDMA$ eine Länge zu.*

3. $\zeta_{TDMA} : S \rightarrow P$ *ordnet jedem Slot $s \in \bigcup_{j \in \{1,\ldots,|TDMA|\}} pr_j(TDMA)$ aus der TDMA-Round $TDMA$ eindeutig einen Prozessorknoten zu, der im Slot s exklusiven Zugriff auf das Bussystem hat.*

4. $\Lambda : S^{\leq N} \rightarrow \mathbb{N}$ *gibt die Länge der TDMA-Round an und ist definiert durch:*

$$\Lambda(TDMA) = \sum_{j=1}^{|TDMA|} \psi\left(pr_j\left(TDMA\right)\right)$$

Die Anzahl der Frames einer Nachricht kann analog der Betrachtung beim CAN-Bus in Gleichung 4.7 berechnet werden. Aus Gründen der Übersichtlichkeit wird im folgenden auf eine gesonderte Berechnung des letzten Frames einer Nachricht (vgl. Gleichung 4.8) verzichtet[1]. Stattdessen wird davon ausgegangen, dass der letzte Frame die gleiche Länge hat, wie alle anderen Frames. Ausnahme von dieser Regel ist das Verpacken einer Nachricht in einen einzelnen Frame; Hierbei wird die exaktere Länge des Frames benutzt. Dies schlägt sich in der Übertragungszeit ρ nieder, die allerdings bei den beiden hier betrachteten Bussystemen unterschiedlich zu Gleichung 4.5 ist. Der Grund hierfür ist in dem fehlenden Mechanismus des Bit-Stuffung, dass sowohl im TTP als auch im FlexRay nicht verwendet wird, zu suchen. Die Übertragungsdauer einer Nachricht $g_t = (\tau_t, s_t)$ kann somit wie folgt berechnet werden:

$$\rho_k(g_t) = \begin{cases} \dfrac{h_k + 8 \cdot \eta_k + \chi_k}{\upsilon_k} & \text{wenn } s_t > \eta_k \\ \dfrac{h_k + 8 \cdot s_t + \chi_k}{\upsilon_k} & \text{sonst} \end{cases} \qquad (4.11)$$
$$\text{mit } \kappa(k) = (h_k, \eta_k, \upsilon_k, \beta_k)$$

[1] Dies stellt keine große Veränderung gegenüber dem CAN-Bus dar, weil die Anzahl der Nutzdaten-Bytes sowohl bei FlexRay als auch bei TTP deutlich über denen des CAN-Bus liegen (16 bzw. 246 Bytes vs 8 Bytes). Folglich wird es nur sehr wenige Nachrichten geben, die mehrere Frames nutzen müssen. Gleichwohl ist die Anpassung an die im folgenden aufgestellten Analysen jederzeit einfach möglich, da es sich, wie bei der Integration in die CAN-Bus-Analysen, nur um simple Erweiterungen handelt.

Da sowohl in FlexRay als auch in TTP $\beta_k = 0$ gilt, muss die Overhead-Berechnung laut Gleichung 4.6 angepasst werden und umfasst nur noch Idle-Phasen und Synchronisations-Phasen des jeweiligen Kommunikationsmediums.
Auch hier stellt sich wiederum die Frage, ob mehrere Nachrichten kurzer Länge in einem gemeinsamen Frame verpackt werden können. Arbeiten, die diese Art der Optimierung auf Zeit-basierten Bussystemen untersuchen, sind beispielsweise in [92] für TTP und [143] für TTP und FlexRay zu finden, sollen hier jedoch nicht weiter betrachtet werden: Die Einflüsse auf den Entwicklungsprozess sind dramatisch, weil jede noch so kleine Änderung bzgl. der Slotnummer z.B. unkalkulierbare Risiken im Integrationsprozess bedeutet. Diese treten durch weitgehende Abhängigkeiten auf, die auf den ersten Blick nicht zu erkennen sind, aber gravierende Kosten zur Folge haben (vgl. [121]).
Für die minimale Länge ψ_{TDMA} wird gefordert, dass es ein ganzzahliges Vielfaches der maximalen Frame-Größe sein muss. Würden die Längen unterhalb dieses Maßes liegen, könnten bestimmte Frames nicht versendet werden, auch wenn sie laut Priorität versendet werden müssten. Dies hat entweder einen nichtgenutzten Slot zur Folge oder aber bewirkt eine nicht gewollte Prioritäten-Inversion durch das Vorziehen anderer, niedriger priorisierter Nachrichten. Beide Effekte sind zu vermeiden, und dies lässt sich nur durch die Beschränkung der minimalen Slot-Größe erreichen.

Letzteres deutet schon an, worum es in diesem Abschnitt schwerpunktmäßig geht: Die Kopplung von Ereignis-basierten Tasksystemen mit Zeit-basierten Kommunikationsmedien. Bei der Optimierung von Zeit-basierten Tasksystemen mit ebensolchen Kommunikationsmedien sind derartige Einschränkungen nicht notwendig, da durch die entsprechend geschickt zu wählenden Offsets der Taskaktivierung und der Slots auf dem Kommunikationsmedium eine spezielle angepasste Verteilung entsteht. Diese Art des Platzierungsproblems ist beispielsweise in [14, 15] mithilfe von Mixed Integer Linear Programming angegangen worden und soll im Kontext dieser Arbeit nicht weiter betrachtet werden. Interessanter ist hier die Kopplung von Ereignisbasierten Tasksystemen mit Time Triggered Architekturen: Die nicht vorhandenen Relationen der Startereignisse von Tasks zu den festen Offsets und die Priorisierung der Tasks auf einem Knoten verlangt komplexe Scheduling-Analysen, wie sie auch im Kontext des CAN-Busses vorzunehmen waren.

Weitere Einflüsse durch die unterschiedlichen Schichten der Protokolle gleichen sehr stark den im CAN-Bus dargestellten Maßnahmen (z.B. das Problem der Blockierung) und können daher von dort übernommen werden ohne hier nochmals dargestellt werden zu müssen.
Verschiedene Zeit-basierte Bussysteme können sich in der (sehr wesentlichen) Frage unterscheiden, wie oft ein Prozessorknoten während einer TDMA-Round einen Slot auf dem Bussystem besitzt. Beide hier betrachteten Time Triggered Protokolle unterscheiden sich diesbezüglich, so dass hier zunächst der TTP-Ansatz vorgestellt wird. Dieser schreibt vor, dass jeder Knoten nur einen Slot während einer TDMA-Round besitzen darf. Anschließend wird der FlexRay-Bus analysiert, der eben jene Beschränkung nicht hat. Abschließend erfolgt aus Gründen der Vergleichbarkeit der hier vorgestellten Arbeiten eine Betrachtung eines einfachen Tokenring-Protokolls.

TTP — Eine einfache Variante

Das Time Triggered Protocol (TTP)[189] ist ein TDMA-basiertes Busprotokoll, das an der TU Wien entstanden und mittlerweile durch ein Time-Triggered OS und entsprechende Spezifikationen für Bus-Controller u.ä. zu einem verfügbaren Gesamtsystem ausgebaut wurde. Entscheidender Wert wurde hierbei auf Fehlertoleranz und Ausfallsicherheit gelegt.
Es besteht die Möglichkeit, bis zu 64 Slots innerhalb einer TDMA-Round anzuordnen, wobei jeder Slot durch eine sogenannte SRU gekapselt wird, die diesem Slot virtuell mehreren Nachrichten zuteilen kann, wenn diese sich in ihrer Periode so unterscheiden, dass es zu keinen Kollisionen innerhalb der virtuellen Slots kommt.
Ein entscheidender Faktor bezüglich der Echtzeiteigenschaften bei TTP ist die Beschränkung auf nur einen Slot eines Prozessors pro TDMA-Round, d.h. es gilt für eine TDMA-Round $TDMA^{TTP} = \text{“}s_1 \dots s_n\text{“}$:

$$s_i \in \bigcup_{j \in \{1,\dots,|TDMA|\}} pr_j(TDMA) : \zeta(s_i) = p \rightarrow \forall 1 \leq j \leq |TDMA|, j \neq i : \zeta(s_j) \neq p$$

Der Zugang zum Bus ist streng an feste Zeiten gekoppelt, die in einem entsprechenden Bus Controller geprüft werden. Hierzu ist eine globale Zeitsynchronisation auf den am Bus hängenden Controllern notwendig, welche durch ein spezielles Protokoll im Transfer-Layer des TTP durchgeführt wird. Zeit-basierte Tasks sind so einzuplanen, dass sie zu einem bestimmten Zeitpunkt ihre Nachricht an den Bus Controller absetzen und dieser die Nachricht bei Beginn des diesem Prozessorknoten zugeteilten Slots auf dem Bussystem versendet. Die Zuordnung der Nachrichten zu Slots auf der empfangenden wie auf der sendenden Seite wird über sogenannte Message Descriptor Lists (MEDL) verwaltet, die bei der Systemerstellung erzeugt werden müssen.
Ist die Nachrichtenversendung von Ereignis-basierten Tasks vorgesehen, so kann bzgl. des Bus Controllers das gleiche Protokoll verwendet werden. Allerdings muss dann die RTOS-Schicht, die für die Kommunikation zuständig ist, die im vorigen Kapitel erwähnten FIFOs bereitstellen. Diese stellen eine Versendung der Nachrichten gemäß ihrer Priorisierung sicher.

Konzeptionell ist dieses Verfahren dem Ereignis-basierten Versendemechanismus des vorangegangenen Abschnitts sehr ähnlich, nur dass durch die strenge Aufteilung des Zugriffes in Slots zusätzliche Blockierungen entstehen. Diese sind in der Scheduling-Analyse zu berücksichtigen. Tindell hat dies in [181] bereits durchgeführt, so dass an dieser Stelle lediglich das Ergebnis dieser Arbeiten angegeben werden soll. Demnach ist die Antwortzeit einer Nachricht analog dem Vorgehen in Gleichung 4.9 anzugeben, allerdings erweitert um einen Blockierungsfaktor $B_{TTP}(s_j)$ und abhängig von dem Slot, in dem die Nachricht versendet werden muss:

$$r^n_{g_t}(s_g, TDMA) = c^k_{g_t} \cdot \rho_k(g_t) + B_{g_t} + I^k_{g_t}(r^n_{g_t}, s_g) + B_{TTP}(r^n_{g_t}, s_g, TDMA) \quad (4.12)$$

Die Blockierung $B_{TTP}(s_g)$ entspricht der Situation, in der die Nachricht g_t gerade ihren Sendezeitpunkt im eigenen Slot s_g verpasst hat und dies um einen unendlich kleinen Zeitraum. Daher kann die Übertragung von g_t erst wieder starten, wenn

der Slot erneut aktiviert wird. Dies ist entsprechend eine TDMA-Round (ohne den betroffenen Slot) später. Für den Blockierungsfaktor ergibt sich demnach:

$$B_{TTP}(r^n_{g_t}, s_g) = \left\lceil \frac{r^n_{g_t}}{\Lambda(TDMA)} \right\rceil \cdot (\Lambda(TDMA) - \psi(s_g)) \tag{4.13}$$

Der Interferenzfaktor ist analog dem Vorgehen beim CAN-Bus (vgl. Gleichung 4.10) vorzunehmen. Lediglich bei der Größe der Nachrichten ergeben sich wiederum Veränderungen ($\rho_k(g_t)$ bzw. $\rho_k(g_i)$ für unterbrechende Nachrichten).

$$I^k_{g_t}(r^n_{g_t}, s_g) = \sum_{\forall g_i \in \breve{C}^{\zeta(s_g)}_k(p):\phi_{g_i} > \phi_{g_t}} \left\lceil \frac{r^n_{g_t} + J_{g_i}}{t_{g_i}} \right\rceil \cdot c^k_{g_i} \cdot \rho_k(g_i)$$

Unterbrechende Nachrichten sind hier, anders als beim CAN-Bus, nicht mehr global zu betrachten, sondern da die Slots an einen einzigen Prozessorknoten gekoppelt sind, können innerhalb dieses Slots auch nur Nachrichten untereinander interferieren, die auf dem selben Knoten erzeugt wurden. Die Menge der höher priorisierten Nachrichten $\breve{C}^p_k$ für einen Prozessorknoten p ist dann gegeben durch:

$$\begin{aligned} \breve{C}^p_k = \quad & \left\{ g_i \mid \forall \tau_j \in \mathcal{T},\ \forall g_i \in \gamma_j : \Gamma(\tau_j, g_i) = \text{“}\ldots k \ldots\text{“} \wedge\ \Pi(\tau_j) = p \right\} \\ \bigcup\ & \left\{ g_j \mid \forall \tau_h \in \mathcal{T},\ \forall g_j \in \gamma_h : \Gamma(\tau_h, g_j) = \text{“}\ldots k' k \ldots\text{“} \wedge p \in k' \cap k \right\} \end{aligned}$$

Die erste Teilmenge der Konstruktion von $\breve{C}^p_k$ enthält alle Nachrichten, die von Tasks auf dem Knoten p erzeugt werden und k zur Übertragung nutzen. Die zweite Teilmenge enthält alle Nachrichten, die p als Gateway von einem Kommunikationsmedium k' zu k nutzen. Da auch diese Tasks, wie eingangs erläutert, in die Ausgangs-FIFO zu Medium k eingetragen werden müssen, sind sie potentielle Kandidaten für interferierende Blockierung und müssen folglich in $\breve{C}^p_k$ berücksichtigt werden. Es wird hierbei von nur einem Gateway-Knoten pro Kommunikationsmedien-Paar ausgegangen[1].
Damit ist die vollständige Charakterisierung des Echtzeitverhaltens von Nachrichten auf einem auf TTP-basierenden Kommunikationssystems gegeben. Die Scheduling-Analyse muss nun letztlich für jede Nachricht, die über dieses Bussystem gesendet werden muss, die Feasibility nachweisen:

$$r^n_{g_t}(s_g, TDMA) \leq \Delta_\gamma(g_t) \text{ mit } g_t \in \gamma_s, \Pi(\tau_s) = p, \zeta(s_g) = p, g_t \mapsto s_g$$

FlexRay — Komplexe Slotverteilung

Während Time Triggered Architekturen, wie sie im letzten Abschnitt beschrieben wurden, davon ausgehen, dass jedem Prozessor im Zuge einer TDMA-Round nur *einmal* die Gelegenheit geboten wird, auf das Bussystem zuzugreifen, geht die FlexRay-

[1] Bei mehreren Gateway-Knoten würde diese Formulierung ansonsten in der Analyse auf allen diesen Knoten Preemptionen verursachen, was natürlich nicht mit der Realität übereinstimmt. Eine Erweiterung auf mehrfache Gateways ist im Prinzip problemlos möglich, allerdings muss das Platzierungsverfahren dann noch zusätzlich den benutzten Gateway-Knoten bestimmen.

Spezifikation[55][1] hier weiter. Es wird eine zusätzliche Flexibilisierung der Nachrichtenübertragung dadurch erreicht, dass in jeder TDMA-Round beliebig viele Slots einem Prozessorknoten zugeordnet werden können. Weder deren Länge noch deren zeitliche Distanz zueinander sind vorgeschrieben, so dass nahezu beliebige Slot-Anordnungen verwendbar sind. Dies hat natürlich gravierende Auswirkungen auf die Antwortzeit-Analyse: Konnte in Systemen mit nur einem Slot pro TDMA-Round noch davon ausgegangen werden, dass die für die Analyse notwendige kritische Instanz als Startzeitpunkt äquivalent zu einem Taskssystem behandelbar ist, geht das hier nun nicht mehr. Stattdessen muss aus der Abfolge von Slots für einen Prozessorknoten die kritische Instanz errechnet werden, wobei diese Abfolge zwar gemäß der TDMA-Definition eine Periode besitzt, aber die Abfolge der Slots innerhalb dieser Periode keinerlei zeitliche Beziehung mehr aufweist.

Kritische Instanz besagt, dass der Start der Analyse die Annahme macht, dass jede Nachricht zum Zeitpunkt 0 bereit ist zu senden. Da in einer TDMA-Round der zur Nachricht gehörende Prozessorknoten $p = \Pi(\tau_i)$ nicht notwendigerweise am Beginn des TDMA-Zyklus stehen muss, wird in der folgenden Analyse von einer auf die kritische Instanz aus Sicht des betrachteten Prozessors verschobenen Periode ausgegangen[2]. Es wird eine Transformation $rot(TDMA, p)$ definiert, die eine TDMA-Round soweit nach links rotiert, bis der nächste von p benutzbare Slot gerade verstrichen ist. Abbildung 4.9 macht dies an einem Beispiel deutlich. Damit ist sichergestellt, dass die erreichte Verschiebung des in der Antwortzeitanalyse zu betrachtenden Startzeitpunktes einer möglichen kritischen Instanz entspricht: Die Blockierung durch das Verpassen eines möglichen Slots um einen unendlichen kleinen Zeitraum entspricht einer höher priorisierten interferrierenden Blockierung und ist demnach analog dem Schedulingproblem für Tasksysteme behandelbar.

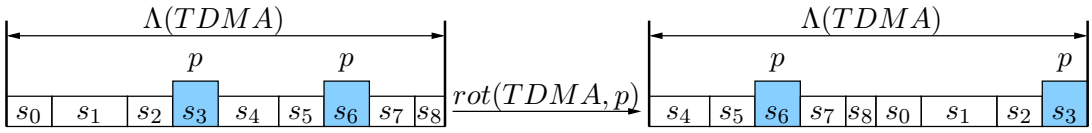


Abbildung 4.9: Eine Transformation $rot(TDMA, p)$ rotiert eine TDMA-Sicht so nach links, dass der nächste Slot, der dem betrachteten Prozessorknoten zugeordnet ist, gerade vorbei ist.

Sei $rot(TDMA, p) : S^{\leq N} \times P \rightarrow S^{\leq N}$ definiert durch:

$$rot(TDMA, p) = \begin{array}{l} \text{“}s_{i+1}s_{i+2}\ldots s_n s_1 \ldots s_i\text{“ mit } TDMA = \text{“}s_1 \ldots s_n\text{“} \\ \quad \wedge\, \zeta_{TDMA}(s_i) = p \;\wedge\; \forall 1 \leq j < i : \zeta_{TDMA}(s_j) \neq p \end{array} \tag{4.14}$$

Scheduling-Analysen für eine Nachricht $\gamma_s \ni g_t = (\tau_t, s_t)$ zwischen einem Task τ_s und τ_t beziehen sich im folgenden grundsätzlich auf einer durch $rot(TDMA, \Pi(\tau_s))$

[1] FlexRay ist ein TDMA-basiertes Busprotokoll, dass für den Einsatz in der Fahrzeugelektronik von einem Konsortium namhafter Automobil-Hersteller und Zulieferer entwickelt wurde und wird. Es bietet neben dem TDMA-Ansatz — wie TTP — weitreichende Möglichkeiten zur Fehlertoleranz und Ausfallsicherheit.

[2] Da es sich bei einer TDMA-Round um eine periodische Abfolge von $TDMA_i$ handelt, bleibt durch die Verschiebung die grundsätzliche Eigenschaft erhalten.

entstandenen Sicht auf die TDMA-Round. Dabei müssen alle möglichen verschiedene Verschiebungen betrachtet werden, die dafür sorgen, dass der jeweilige $\Pi(\tau_s)$ zugeordnete Slot gerade verstrichen ist. Die Gründe für die notwendige Betrachtung aller Verschiebungen sind in dem folgenden Beispiel motiviert:
Sei eine TDMA-Round mit mehrfacher Slotzuweisung zu einem Prozessorknoten gegeben – wie in Abbildung 4.10 dargestellt. Zwar ist die TDMA-Round an sich periodisch und somit auch die einzelnen Slots s_i, jedoch gibt es keine Periodenbeziehung der einem Prozessorknoten p zugeordneten Slots $s_{p_0}, \ldots, s_{p_4}$. Eine zentrale Annahme bei der in dieser Arbeit verwendeten Antwortzeit-Analysen ist die der kritischen Instanz, wie sie von [104] eingeführt und in Kapitel 2.3 beschrieben ist. Sie reflektiert die Frage, welche Situation der möglichen Startzeitpunkte von Tasks oder Nachrichten aus Echtzeitsicht den schlimmsten anzunehmenden Fall darstellen. Im Falle periodischer Nachrichten (oder sporadischer Nachrichten mit festgelegten minimalen Ankunftszeiten) ist dies trivialerweise das simultane Szenario, d.h. alle Nachrichten sind zum gleichen Zeitpunkt übertragungsbereit.

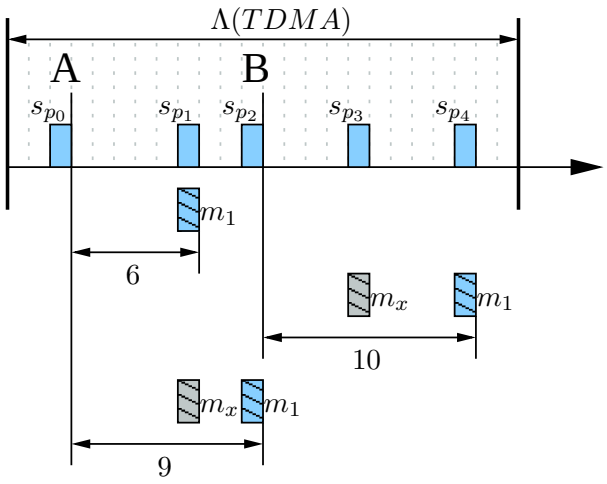


Abbildung 4.10: Zwei verschiedene mögliche kritische Instanzen A und B auf einer TDMA-Round mit mehrfacher Slotausnutzung am Beispiel von Nachrichtenübertragungen mit einem und mit zwei Frames.

In der in Abbildung 4.10 dargestellten multiplen Slotzuweisung ist dies aber nicht so klar zu erkennen. Aus der Annahme, eine maximale Blockierung zu suchen und den Zeitpunkt des Beginns dieser maximalen Blockierung als kritische Instanz zu definieren, folgt für das angegebene Beispiel das Szenario A. Sei nun eine Nachricht m_1 gegeben, die auf den zu p gehörenden Slots übertragen werden soll und die lediglich ein Frame zur Versendung der Daten benötigt. Seien weiterhin alle Slots $s_{p_0}, \ldots, s_{p_n}$ des TDMA-Verfahrens von der Breite exakt eines Frames. Wie man sehr einfach anhand des ersten Beispiels in Abbildung 4.10 erkennen kann, ist die Antwortzeit für die Versendung von m_1 unter der Annahme der kritischen Instanz A mit 6 Zeiteinheiten maximal für diese TDMA-Round. Wird auf p allerdings ein weiterer Task allokiert, der eine höher priorisierte Nachricht m_x versendet, so wird m_1 eine Blockierung erfahren und die Übertragung einen Slot weiter nach hinten verschoben (unteres Beispiel in Abbildung 4.10). Dies hat zur Folge, dass unter der kritischen

Instanz A die Antwortzeit der Nachricht auf 9 Zeiteinheiten erhöht wird. Betrachtet man hingegen statt des Szenarios A das Szenario mit B als kritischer Instanz, ergibt sich eine Antwortzeit von 10 Zeiteinheiten. Folglich kann A nicht die kritische Instanz sein, was sich aber mit der Situation widerspricht, in der nur ein Frame verschickt werden muss, da in diesem Fall die kritische Instanz B eine Antwortzeit von nur 5 Zeiteinheiten postuliert.
Man sieht recht schnell anhand dieses Beispiels, dass die Wahl der kritischen Instanz applikationsabhängig ist und ggf. für unterschiedliche Nachrichten differiert: So wäre beispielsweise für Nachricht m_x A die kritische Instanz, wohingegen für m_1 B zu nehmen ist. Es sind also mindestens alle mit (aus Sicht von p) startenden Blockierungen als kritische Instanz für die Nachrichtenübertragung zu betrachten. Diese Situationen werden mit dem Rotationsoperator $rot(TDMA)$ auf einer TDMA-Round hergestellt und ermöglichen es, alle möglichen startenden Blockierungen direkt nach einem p zugeordneten Slot zu analysieren. Hierzu soll im folgenden eine Sicht auf die TDMA-Round $TDMA_i$ verwendet werden, die einem i-fachen Verschieben auf den Slot darstellt, und damit i dem Prozessorknoten $\Pi(\tau_s)$ zugeordnete Slots gerade verstreichen läßt. Sei also

$$TDMA_i^{\Pi(\tau_s)} = \underbrace{rot(rot(\ \ldots\ \ rot(TDMA, \Pi(\tau_s))\ldots), \Pi(\tau_s))}_{i-mal}. \tag{4.15}$$

Slots, die für Zugriffe von Prozessorknoten $\Pi(\tau_s)$ auf das Bussystem vorgesehen sind, können gemäß der FlexRay-Spezifikation[55] mehrfach innerhalb einer TDMA-Round vorkommen. Um die zeitlichen Offsets dieser Slots zu bestimmen, wird zunächst die Menge aller Indizees $X^p_{TDMA_i} \subseteq \mathbb{N}$ über $TDMA_i$ gebildet, deren Slots s_i der entsprechende Prozessorknoten zugeordnet ist, also

$$X^{\Pi(\tau_s)}_{TDMA_i} = \left\{ l \ \middle|\ s_l \in \bigcup_{j \in \{1,\ldots,|TDMA_i^{\Pi(\tau_s)}|\}} pr_j(TDMA_i^{\Pi(\tau_s)}) \ \wedge\ \zeta_{TDMA_i^{\Pi(\tau_s)}}(s_l) = \Pi(\tau_s) \right\} \tag{4.16}$$

Die Mächtigkeit der Menge $X^{\Pi(\tau_s)}_{TDMA_i}$ gibt die Anzahl der Slots an, die Prozessor $\Pi(\tau_s)$ den Zugriff auf das Medium erlauben. Gleichzeitig schließt sich jedem s_{x_l} mit $x_l \in X^{\Pi(\tau_s)}_{TDMA_i}$ eine Reihe von Slots an, die $\Pi(\tau_s)$ den Zugriff verweigern. Aus Sicht einer zur Taskanalyse analogen Scheduling-Analyse sind diese Unterbrechungen als Preemptionen interpretierbar, wie in Abbildung 4.11 dargestellt.

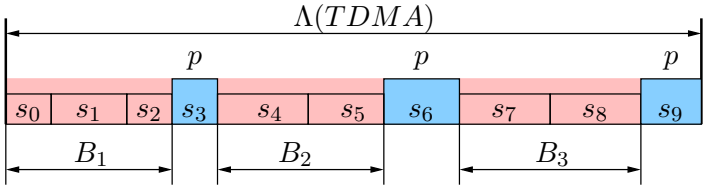


Abbildung 4.11: Nicht dem Prozessorknoten p zugeordnete Slots wirken in der Summe wie blockierende Interferenzen.

Die Anzahl dieser Preemptionen kann mit $|X^{\Pi(\tau_s)}_{TDMA_i}|$ angegeben werden, wobei jede Preemption eine Blockierung B_i^X erzeugt. Die Länge der Blockierung B_i^X kann aus

der Anordnung der Slots bezüglich s_{x_i} in der TDMA-Round und deren Längen angegeben werden:

$$B_l^{X_{TDMA_i}^{\Pi(\tau_s)}} = \begin{cases} \sum_{j=x_l+1}^{x_{l+1}-1} \psi(pr_j(TDMA_i^{\Pi(\tau_s)}) & \text{wenn } x_l + 1 < x_{l+1} \\ 0 & \text{sonst} \end{cases} \tag{4.17}$$

Aufgrund der strikten Einhaltung von Zeitbedingungen in Time-triggered Systemen, wie der FlexRay-Bus eines ist, verfügen die Startzeitpunkte der unterschiedlichen Slots über fest definierte Offsets zueinander. In der Scheduling-Analyse bewirkt die Einbeziehung dieser Offsets, wie eingangs dargelegt, eine bzgl. möglichen Preemptionen im physikalischen System genauere Vorhersage. Sie ist sogar zwingend notwendig, wenn über Zeit-basierte Tasks argumentiert wird. Die gleiche Art der Analyse ist hier notwendig, so dass die Offsets O_l^X der einzelnen Blockierungsslot-Mengen B_l^X berechnet werden müssen.

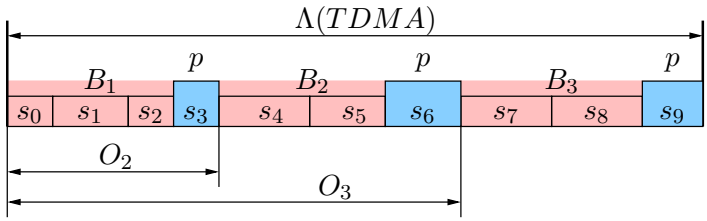


Abbildung 4.12: Blockierende Interferenzen zwischen zwei p zugeordneten Slots haben einen festen Offset bzgl. des Startzeitpunktes der betrachteten TDMA-Round-Verschiebung.

Die Größe der jeweiligen Offsets lässt sich leicht aus der bekannten Größe der Blockierungen und der p zugeordneten Slots errechnen, die vor der betrachteten Blockierung in einer Verschiebung der TDMA-Round liegen (vgl. Abbildung 4.12), wobei $O_1 = 0$ gilt:

$$O_{s_l}^{X_{TDMA_i}^{\Pi(\tau_s)}} = \sum_{j=1}^{l} \psi(pr_j(TDMA_i^{\Pi(\tau_s)})) \tag{4.18}$$

Damit läßt sich nun — basierend auf der Formalisierung von Zeit-basierten Kommunikationen aus dem letzten Abschnitt — das komplexe FlexRay Modell zur Analyse von Antwortzeiten konstruieren. Hierzu sei die Menge der auf dem Knoten p konkurierenden Nachricht $\breve{C}_k^p$ wiederum gegeben durch:

$$\begin{aligned} \breve{C}_k^p = \quad & \{g_i \mid \forall \tau_j \in \mathcal{T},\ \forall g_i \in \gamma_j : \Gamma(\tau_j, g_i) = \text{“}\ldots k \ldots\text{“} \wedge\ \Pi(\tau_j) = p\} \\ \bigcup\ & \{g_j \mid \forall \tau_h \in \mathcal{T},\ \forall g_j \in \gamma_h : \Gamma(\tau_h, g_j) = \text{“}\ldots k'k \ldots\text{“} \wedge p \in k' \cap k\} \end{aligned}$$

Sei weiterhin die Prioritätsabbildung $\phi(g_i, g_j)$ (analog dem Vorgehen beim CAN-Bus) definiert wie in Definition 4.3.1 angegeben. Dann ist die Antwortzeit einer Nachricht auf einem FlexRay-Bussystem k mit gegebener TDMA-Round und unter

Berücksichtigung der TDMA-Verschiebung $TDMA_i$ gegeben durch:

$$
\begin{aligned}
r_{g_t}^k(\Pi(\tau_s), TDMA_i) = \; & c_{g_t} \cdot \rho + B_{g_t} \\
& + \sum_{\forall g_j \in \check{C}_k^p : \phi_{g_j} > \phi_{g_t}} \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s), TDMA_i) + J_{g_j}}{t_{g_j}} \right\rceil \cdot c_j \cdot \rho \\
& + \sum_{j=1}^{|X_{TDMA_i}^{\Pi(\tau_s)}|} \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s)) - O_j^{X_{TDMA_i}^{\Pi(\tau_s)}}}{\Lambda(TDMA)} \right\rceil \cdot B_j^{X_{TDMA_i}^{\Pi(\tau_s)}}
\end{aligned}
\tag{4.19}
$$

Allerdings gilt diese Analyse der Antwortzeit nur bzgl. einer der möglichen Verschiebungen $TDMA_i$ der TDMA-Round. Nur dies zu betrachten ist nicht ausreichend, da durch die beliebige zeitliche Anordnung von Slots für Prozessor $\Pi(\tau_s)$ unterschiedliche Startzeitpunkte der Analyse innerhalb der TDMA-Round zu unterschiedlichen Antwortzeiten führen. Daher müssen für die Berechnung der schlimmsten anzunehmenden Übertragungsdauer alle möglichen Verschiebungen betrachtet und das Maximum bzgl. der Antwortzeit als Ergebnis propagiert werden. Dass es hierfür genügt, nur die unterschiedlichen Verschiebungen $TDMA_i$ zu betrachten und keine weiteren feingranulareren Verschiebung der TDMA-Round eine Erhöhung der Antwortzeit ergeben können, zeigt das folgende Lemma.

Lemma 4.3.1 (Verschiebung als obere Grenze). *Sei $TDMA_i$ eine beliebige Verschiebung einer TDMA-Round bzgl. eines Knotens p. Sei o_2 der Offset der ersten blockierenden Unterbrechung der Übertragung nach dem ersten Slot für p. Sei $TDMA_i^z$ die Sicht auf die TDMA-Round, die sich ergibt, wenn $TDMA_i$ um $0 \leq z < o_2$ Zeiteinheiten verschoben wird. Dann gilt:*

$$
r_{g_t}^k(\Pi(\tau_s), TDMA_i) \geq r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)
$$

Anschaulich bedeutet $TDMA_i^z$ jede beliebige Verschiebung um $z < o_2$ Zeiteinheiten, d.h. es wird nicht mehr davon ausgegangen, dass der letzte für p reservierte Slot gerade verstrichen ist, sondern dass dieser schon länger verstrichen ist und zur betrachteten Zeit gerade eine Blockierung aus Sicht von p auf dem Bus aktiv ist. Abbildung 4.13 zeigt diesen Sachverhalt exemplarisch an einem allgemeinen Wert für z.

Aufgrund der Verschiebung des Startzeitpunktes verändert sich auch die Antwortzeitberechnung, da die erste Blockierung in der Länge verringert wird und gleichzeitig nach dem letzten dem Prozessorknoten zugeordneten Slot der TDMA-Runde eine weitere Blockierung eingeführt wird. Letztere entspricht der Verschiebung z.

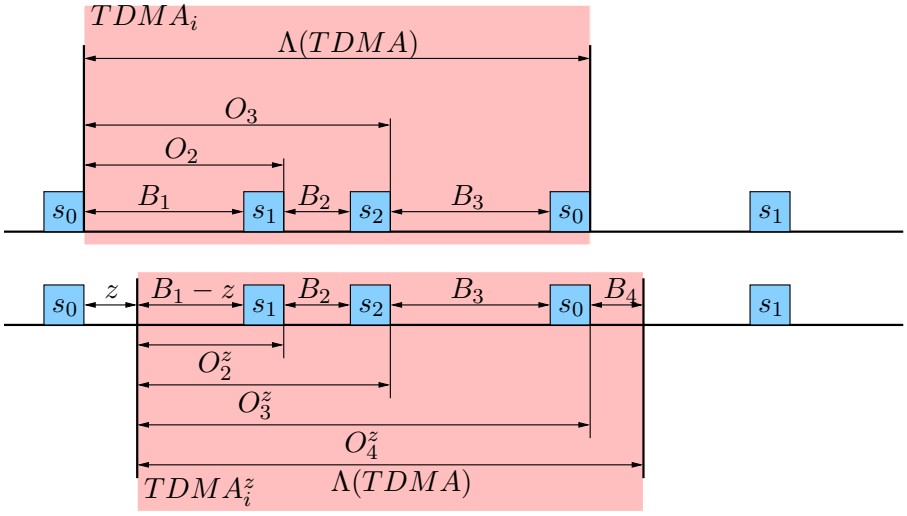


Abbildung 4.13: Die Betrachtung einer möglichen feingranularen Verschiebung einer TDMA-Sicht um z Zeiteinheiten. Dadurch entstehende andere Blockierungszeit B_i^X und andere Offsets O_i^z.

Daraus resultiert eine angepasste Antwortzeitberechnung:

$$
\begin{aligned}
r_{g_t}^k(\Pi(\tau_s), TDMA_i^z) = \; & c_{g_t} \cdot \rho + B_{g_t} \\
& + \sum_{\forall g_j \in \check{C}_k^p : \phi_{g_j} > \phi_{g_t}} \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s), TDMA_i^z) + J_{g_j}}{t_{g_j}} \right\rceil \cdot c_j \cdot \rho \\
& + \sum_{j=2}^{|X_{TDMA_i}^{\Pi(\tau_s)}|} \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s)) - (O_j^{X_{TDMA_i}^{\Pi(\tau_s)}} - z)}{\Lambda(TDMA)} \right\rceil \cdot B_j^{X_{TDMA_i}^{\Pi(\tau_s)}} \\
& + \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)}{\Lambda(TDMA)} \right\rceil \cdot (B_1^{X_{TDMA_i}^{\Pi(\tau_s)}} - z) \\
& + \left\lceil \frac{r_{g_t}^k(\Pi(\tau_s), TDMA_i^z) - (\Lambda(TDMA) - z)}{\Lambda(TDMA)} \right\rceil \cdot z
\end{aligned}
\tag{4.20}
$$

Eine Verschiebung um z Zeiteinheiten in Richtung des nächsten Slots bewirkt — wie in Gleichung 4.20 bereits eingeführt — auch eine Verkürzung der festen Offsets um den Betrag z, d.h. $O_i^z = O_i - z$ mit Ausnahme der zusätzlich entstandenen Blockierung O_{n+1} mit $n = |X_{TDMA_i}^{\Pi(\tau_s)}|$. Letztere beendet die Sicht der TDMA-Round $TDMA_i^z$ und hat die Breite z woraus folgt, dass der Offset aus der Länge der TDMA-Round und dieser Breite konstruiert werden kann: $O_{n+1} = \Lambda(TDMA) - z$. Alle blockierenden Interferenzen außer der ersten bleiben in ihrer Länge unverändert.

Da die allgemeine Form der Antwortzeit-Analyse, wie sie auch hier verwendet wurde, die Vorraussetzung einer kritischen Instanz (vgl. [104]) hat, bedeutet die Verschiebung der Sicht also eine Neudefinition der kritischen Instanz. Demzufolge weist Lem-

ma 4.3.1 nach, dass alle Beobachtungszeitpunkte zwischen $TDMA_i$ und $TDMA_{i+1}$ keine kritische Instanz darstellen. Bleibt also letztlich noch der Beweis dieser Aussage zu führen:

Beweis von Lemma 4.3.1:
Im folgenden werden verkürzte Schreibweisen verwendet: $\Lambda := \Lambda(TDMA)$, sowie $B_i := B_i^{X^{\Pi(\tau_s)}_{TDMA_i}}$ für alle blockierenden Interferenzen und $O_i := O_i^{X^{\Pi(\tau_s)}_{TDMA_i}}$ für alle Offsets. Ein hochgestelltes z an einer Variablen bezeichnet den Fall, dass $TDMA_i^z$ betrachtet wird, ansonsten $TDMA_i$. Sei $Y := |X^{\Pi(\tau_s)}_{TDMA_i}|$.
Beweis durch Widerspruch:
Sei W der Fixpunkt für $r^k_{g_t}(\Pi(\tau_s), TDMA_i)$. Dann soll mit Gleichung 4.19 gelten:

$$c_{g_t} \cdot \rho + B_{g_t} + \sum_{\forall g_j \in \check{C}^p_k : \phi_{g_j} > \phi_{g_t}} \left\lceil \frac{W + J_{g_j}}{t_{g_j}} \right\rceil \cdot c_j \cdot \rho + \sum_{j=1}^{Y} \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \cdot B_j$$

$$<$$

$$\begin{aligned} c_{g_t} \cdot \rho \quad & + B_{g_t} + \sum_{\forall g_j \in \check{C}^p_k : \phi_{g_j} > \phi_{g_t}} \left\lceil \frac{W + J_{g_j}}{t_{g_j}} \right\rceil \cdot c_j \cdot \rho \\ & + \left\lceil \frac{W}{\Lambda} \right\rceil \cdot (B_1 - z) + \sum_{j=2}^{Y} \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil \cdot B_j + \left\lceil \frac{W - \Lambda + z}{\Lambda} \right\rceil \cdot z \end{aligned}$$

Durch die Verwendung von Fixpunkt-Gleichungen bedeutet die Kleiner-Relation hier, dass es ein $W' > W$ gibt mit $W' = r^k_{g_t}(\Pi(\tau_s), TDMA_i^z)$, d.h. es existiert ein Fixpunkt für $r^k_{g_t}(\Pi(\tau_s), TDMA_i^z)$ der nach der Terminierung von $r^k_{g_t}(\Pi(\tau_s), TDMA_i)$ liegt.
Zunächst kann die Gleichung vereinfacht werden, indem alle gleichen Werte eleminiert werden (da hier ein konstantes W verwendet wird, werden beispielsweise beide Antwortzeitberechnungen die gleiche Anzahl an Preemptionen durch höher priorisierte Nachrichten auf dem betrachteten Knoten erfahren). Zudem können alle Offsets mit $O_j = 0$ beseitigt werden.

$$\left\lceil \frac{W}{\Lambda} \right\rceil \cdot B_1 + \sum_{j=2}^{Y} \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \cdot B_j$$

$$<$$

$$\underline{\left\lceil \frac{W}{\Lambda} \right\rceil \cdot (B_1 - z)} + \sum_{j=2}^{Y} \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil \cdot B_j + \underline{\left\lceil \frac{W - \Lambda + z}{\Lambda} \right\rceil \cdot z}$$

Anschließend können die beiden unterstrichenen Teilformeln aus der Antwortzeitbe-

rechung von $r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)$ vereinfacht werden zu

$$\left\lceil \frac{W}{\Lambda} \right\rceil \cdot (B_1 - z) + \left\lceil \frac{W - \Lambda + z}{\Lambda} \right\rceil \cdot z$$

$$\overset{\text{Def. } \lceil\cdot\rceil}{\Longleftrightarrow} \left\lceil \frac{W}{\Lambda} \right\rceil \cdot B_1 - \left\lceil \frac{W}{\Lambda} \right\rceil \cdot z + \left\lceil \frac{W + z}{\Lambda} \right\rceil \cdot z - z$$

und eingesetzt und vereinfacht ergibt:

$$\sum_{j=2}^{Y} \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \cdot B_j$$

$$<$$

$$\sum_{j=2}^{Y} \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil \cdot B_j + \underbrace{\left\lceil \frac{W + z}{\Lambda} \right\rceil \cdot z - \left\lceil \frac{W}{\Lambda} \right\rceil \cdot z - z}_{Z}$$

Der Wert für Teil Z der Gleichung kann entweder mit $Z = 0$ oder mit $Z = -z$ angegeben werden, da $z < \Lambda$. Somit vereinfacht sich die Ungleichung zu

$$\left(\sum_{j=2}^{Y} \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil - \sum_{j=2}^{Y} \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \right) \cdot B_j > -Z$$

Die nachzuweisende Eigenschaft besagte, dass wenn W der Fixpunkt für die $TDMA_i$-Sicht ist, $r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)$ nicht mit einem Wert kleiner als W terminieren kann. Dies soll nun zu einen Widerspruch geführt werden. Es gilt folglich:

$$\sum_{j=2}^{Y} \left(\left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil - \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \right) \cdot B_j > -Z \tag{4.21}$$

Weiterhin gilt gemäß der Definition der Ceiling-Funktion, da $z < \Lambda$, dass die Differenz der jeweiligen Anzahlen an blockierenden Unterbrechungen für $r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)$ nur 1 oder 0 sein kann:

$$\forall j \in \{2, \ldots, Y\} : \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil - \left\lceil \frac{W - O_j}{\Lambda} \right\rceil \in \{0, 1\}$$

Betrachtet man Abbildung 4.13, so wird deutlich, dass für die Sicht $TDMA_i$ auf jede unterbrechende Blockierung B_j ein Slot folgt, der dem Prozessor p zugeordnet ist. Da $r_{g_t}^k(\Pi(\tau_s), TDMA_i)$ mit dem Wert W als Fixpunkt terminiert, kann die Anzahl der zu belegenden Slots S für die zu übermittelnde Nachricht unter Berücksichtigung von höher priorisierten Nachrichten per Konstruktion direkt aus der Anzahl der interferierenden B_j abgeleitet werden. Laut Gleichung 4.21 und mit $-Z \in \{0, z\}$ muss zur Erfüllung der Ungleichung für mindestens ein B_j die Anzahl der Interferenzen durch diese Blockierung innerhalb der Zeitspanne W für $r_{g_t}^k(\Pi(\tau_s), TDMA_i^z)$ um mindestens 1 größer sein. Gleichzeitig ist bekannt, dass eine weitere Aktivierung dieses Blockes einen weiteren dem betrachteten Prozessor zugeordneten Slot für die Nachrichtenübertragung innerhalb von W zur Verfügung stellt. Daraus folgt, dass

für $r^k_{g_t}(\Pi(\tau_s), TDMA^z_i)$ mindesten $S+1$ Slots für die Nachrichtenverschickung benötigt werden. Da zum Zeitpunkt W in beiden Szenarien gleich viele höher priorisierte Nachrichten Blockierungen verursachen und damit die Anzahl der benötigten Slots bei gleicher Nachricht gleich sein müssen, kann die Antwortzeit $r^k_{g_t}(\Pi(\tau_s), TDMA^z_i)$ nicht mehr Slots umfassen als $r^k_{g_t}(\Pi(\tau_s), TDMA_i)$. Es muss also gelten, dass

$$\forall j \in \{2, \ldots, Y\} : \left\lceil \frac{W - O_j + z}{\Lambda} \right\rceil - \left\lceil \frac{W - O_j}{\Lambda} \right\rceil = 0$$

Dies steht im Widerspruch zur Annahme (da $0 \not> -Z$) und damit folgt:

$$r^k_{g_t}(\Pi(\tau_s), TDMA^z_i) \leq r^k_{g_t}(\Pi(\tau_s), TDMA_i)$$

□

Anschaulich bedeutet Gleichung 4.21 unter der Belegung von W, dass $r^k_{g_t}(\Pi(\tau_s), TDMA^z_i)$ mit einem Wert jenseits des Fixpunktes dieser Gleichung betrachtet wird, der sich jedoch durch eine Fixpunktiteration niemals einstellen wird.
Mithilfe von Lemma 4.3.1 läßt sich folglich nachweisen, dass bei der Analyse der Antwortzeit einer Nachricht auf einem FlexRay-Bussystem nur die möglichen Verschiebungen $TDMA_i$ zu betrachten sind, wie es in Abbildung 4.14 dargestellt ist.

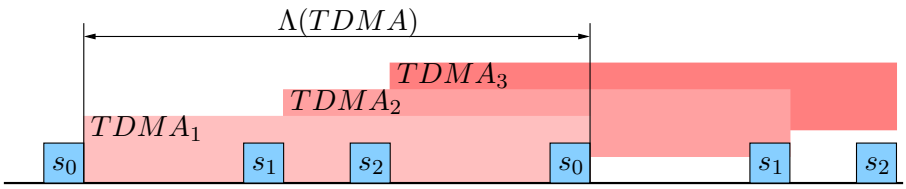


Abbildung 4.14: Alle für eine beispielhafte TDMA-Round möglichen Verschiebungen, die im Zuge der Antwortzeitanalyse betrachtet werden müssen.

Demzufolge ergibt sich für die größte anzunehmende Antwortzeit einer Nachricht:

$$r^{FlexRay}_{g_t}(\Pi(\tau_s)) = \max_{\forall i \in \{1,\ldots,|X^{\Pi(\tau_s)}_{TDMA}|\}} \left(r^{FlexRay}_{g_t}(\Pi(\tau_s), TDMA_i) \right) \tag{4.22}$$

Tokenring — Die flexible Variante

Zwecks Vergleichbarkeit der hier vorgestellten Methoden insbesondere mit den Arbeiten aus Arbeitsgruppe um Tindell (vgl. [183]) soll hier auch der in deren Ansätzen verwendete einfache Tokenring betrachtet werden. Ein Tokenring ähnelt dem TTP aus dem vorhergehenden Abschnitten insoweit, als dass es einen TDMA-Zugriff auf ein Medium verwendet, jedoch die Zeitpunkte des jeweilig zum Prozessorknoten gehörigen Slots nicht definiert sind. Vielmehr existiert eine spezielle Nachricht — das sogenannte Token — welches von einem Knoten zum nächsten gesendet wird. Solange ein Knoten Besitzer des Tokens ist, kann er Datenpakete auf dem Bus absetzen und alle anderen Busteilnehmer horchen den Bus ab. Ist die Datenübertragung beendet, so wird das Token zirkulär an einen Nachbarknoten weiter versendet.

Diese Art der Busarbitrierung bietet Fairness und Flexibiltät zugleich. Die Flexibilität zeichnet sich dadurch aus, dass ein Slot, über den keine Nachrichten versendet werden müssen, nicht den Bus blockiert, sondern das Token direkt weitergereicht werden kann. Dies kann sich in Systemen positiv auswirken, die neben harten Echtzeit-Tasks auch noch Tasksysteme ohne harte Deadlines parallel ausführen müssen. Ausgelassene Slots bieten diesen Tasks die Möglichkeit, ihre Nachrichten abzusetzen ohne die Deadlines des Tasksystems mit harten Echtzeitanforderungen zu beeinträchtigen. Für eine Scheduling-Analyse muss gleichwohl davon ausgegangen werden, dass es sich bei dem Tokenring um ein zeit-basiertes Bussystem handelt, da diese Art der Betrachtung im Tokenring den schlimmsten anzunehmenden Fall darstellt (jeder Knoten benötigt also die vollständige, maximal für ihn veranschlagte Übertragungszeit).

In [183] wird von einem sehr einfachen Bussystem ausgegangen, dass weder Frames noch eine Verteilung eines Datenpaketes auf mehrere Slots vorsieht. Stattdessen wird von einer reinen Datenübertragung ausgegangen, die von genaueren Protokollbestandteilen abstrahiert: Jeder Knoten bekommt den Zugriff auf den Bus solange, wie er benötigt, anfallende Nachricht abzusetzen. Diese Zeit wird Token Hold Time (THT) genannt. Dementsprechend sind auch keine Prioritäten für Nachrichten notwendig, weder auf dem Kommunikationsmedium noch auf dem sendenden Knoten. Auf letzterem werden Nachrichten bis zum Empfangen des Tokens aufgesammelt und dann versendet. Die Zeit die das Token maximal benötigt, um einmal von einem Knoten abgesendet wieder bei eben diesem Knoten zu landen, nennt sich Token Rotation Time (TRT). Die TRT bestimmt sich aus den THT-Zeiten aller angeschlossenen Knoten und der Übertragungszeit ξ für das Token selbst:

$$TRT = \sum_{p_j \in k_{Tokenring}} (THT_p + \xi) \tag{4.23}$$

Die THT_p bestimmt sich aus der Summer der Übertragungszeit aller Nachrichten auf dem Knoten p. Dies bedeutet aber eine weitere Einschränkung bzgl. der TRT: Sie darf nicht größer werden als die kleinste Periode im System, da ansonsten eine Nachricht womöglich mehrfach versendet werden muss, wenn das Token das nächste Mal den Knoten erreicht, d.h. es gilt

$$TRT \leq \min_{\forall \tau_i in \mathcal{T}} (t_i)$$

und daraus folgt dann, dass die Tokenhaltezeit nur von der Anzahl der Nachrichten, die den betrachteten Knoten p verlassen, determiniert wird:

$$THT = \sum_{\tau_s \in \mathcal{T}: \Pi(\tau_s) = p} \left(\sum_{g_t = (\tau_t, s_t) \in \gamma_s : \Pi(\tau_t) \neq p} \frac{s_t}{\upsilon_{Tokenring}} \right) \tag{4.24}$$

Aus Sicht der Scheduling-Analyse bedeuten diese Bedingung, dass im schlimmsten anzunehmenden Fall die Übertragung einer Nachricht gerade TRT Zeiteinheiten benötigt, da sie gerade den Sendeplatz des zu ihrem Knoten gehörenden Tokens verpasst hat. Somit ist die Antwortzeitberechnung einfach

$$r_{g_t}^{Tokenring} = TRT \leq \Delta_\gamma(g_t),$$

unabhängig vom ausgehenden Knoten.
Erweitert man dieses Modell beispielsweise um Frames und TRT-Werte oberhalb von Perioden, so führt dies automatisch zu dem Modell der zeit-basierten Kommunikation, wie es weiter oben für TTP vorgestellt wurde.

4.3.3 Gemischte zeit- und ereignis-basierte Bus-Systeme

Gemischte Zeit- und Ereignis-basierte Bussysteme verwenden beide Varianten gleichzeitig, wobei die Ereignis-basierte Variante in einen Slot der TDMA-Round eingebunden ist (vergleiche Abbildung 4.15).

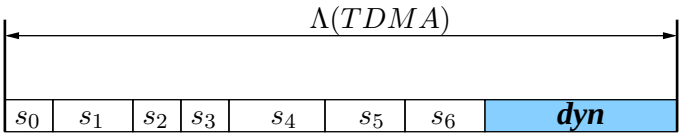


Abbildung 4.15: Zeit-getriebenes TDMA-Protokoll, dass einen Slot besitzt, in dem eine dynamische, prioritäten-orientierte Arbitrierung erfolgt, die allen angeschlossenen Knoten zugänglich ist.

Der Sinn dieser auf den ersten Blick merkwürdig erscheinenden Kopplung zweier vollkommen unterschiedlicher Methodiken zur Nachrichtenverschickung liegt in der steigenden Flexibilität. Es ist bekannt (siehe z.B. die Messungen in [178]), dass Ereignis-basierte Systeme deutlich mehr Flexibilität und damit eine bessere Auslastung des Bussystems erlauben. Auf der anderen Seite sind Zeit-basierte Systeme deutlich deterministischer, da sie keinen Jitter auf den Taskketten erzeugen. Eine Mischung beider Varianten verspricht daher in solchen großen Systemen einen erheblichen Gewinn, in denen die Beschränkung auf nur eines dieser Verfahren eine nicht hinnehmbare Leistungsdämpfung des Systems bewirken würde. Zudem sind Zeit-basierte Ansätze gerade in inkrementellen Entwurfsprozessen kaum oder nur unter großen Schwierigkeiten einzusetzen, wenn, so wie es heutzutage industrielle Praxis ist, beispielsweise die Slots und deren Länge schon während der Spezifikationsphase mit einem Zulieferer ausgehandelt werden müssen[1].
Aus diesen Gründen ist bei der Definition beispielsweise des FlexRay-Standards bereits eine Mischform vorgesehen. FlexRay erlaubt eine Konfiguration des Systems durch die folgenden Verfahren:

- Gemischter Betrieb: Bis zu 1028 Slots können dem TDMA-Verfahren zugeordnet werden. Ein Slot wird als sogenannter dynamischer Slot verwendet, der eine festgelegte Größe hat. Innerhalb dieses Slots können konkurierende Nachrichten in einem Protokoll ähnlich dem CAN-Bus-Arbitrierungsverfahren Ereignis-basiert ihre Kommunikation abwickeln.[2]

[1] Kapitel 6.2 widmet sich diesem Thema eingehender.

[2] Die FlexRay-Spezifikation sieht hier streng genommen keinen wirklichen dynamischen Anteil des Protokolles vor, sondern es wird ein auf sogenannten Minislots basierenden TDMA-Verfahren verwendet. In diesem Verfahren ist die Anzahl der Slots für eine Nachricht allerdings nicht vorgeschrieben, so dass ein dynamisierenderes Verhalten entsteht als bei einem reinem TDMA-Verfahren.

- Rein statischer Betrieb: Alle 1028 Slots sind dem TDMA-Verfahren zugeordnet, es gibt keinen Ereignis-basierten Anteil.

- Rein dynamischer Betrieb: Es gibt keinen TDMA-Anteil. Stattdessen wird der Bus komplett Ereignis-basiert betrieben. Das Protokoll wird auf die *byteflight*-Spezifikation [19] umgestellt.

Die letzteren beiden Betriebsarten können mit den Ansätzen aus den beiden vorangegangenen Abschnitten beschrieben werden, nicht jedoch der gemischte Ansatz, so dass hier nun eine Betrachtung dieser Betriebsart folgen soll.

Eine vereinheitlichte Theorie eines gemischten Bussystems ist nicht einfach zu finden und auch gar nicht notwendig. Vielmehr soll im folgenden jeder Teil des gemischten Mediums isoliert betrachtet und durch entsprechende Maßnahmen erweitert werden, so dass die Zusammenfassung beider Analysen ein korrektes Abbild des Echtzeitverhaltens des gesamten gemischten Kommunikationsmediums ergibt. Die wesentliche Idee hierbei ist, den gemischten Bus nicht als *ein* Kommunikationsmedium zu betrachten, sondern als *2 voneinander getrennte und parallel in der Architektur liegende* Medien. Jedes Medium gehorcht den jeweilig für die Versendung von Nachrichten festgelegten Bedingungen, verfügt aber zusätzlich zu den weiter oben isoliert dargestellten Verfahren noch über eine zusätzliche Blockierung. Diese Blockierung entspricht dem Aktivieren des jeweils anderen Teil des Bussystems und der eigenen Suspendierung. Folglich müssen die entsprechenden Analysen um diesen Faktor erweitert werden.

Der Zeit-basierte Anteil

Für den Zeit-basierten Anteil legt Gleichung 4.12 die Grundlage für die Antwortzeitberechnung von Nachrichten. Eine Einführung eines zusätzlichen Slots, in dem die dynamische Nachrichtenarbitrierung stattfindet, wird aus Sicht des TDMA-Verfahrens lediglich eine zusätzliche Blockierung. FlexRay legt den dynamischen Slot per Definition an den Anfang einer TDMA-Round. so dass hier oBdA davon ausgegangen wird, dass die Blockierungszeit B_{dyn} für die Analyse die Blockierung B_1 verlängert[1]. Demnach ergibt sich die folgende erweiterte Antwortzeitberechnung:

Wir werden jedoch im folgenden der Einfachheit halber davon ausgehen, dass es sich bei dem dynamischen Anteil um einen dem CAN-Bus ähnelnden Protokollabschnitt handelt.

[1] Es wird hier davon ausgegangen, dass die TDMA-Round grundsätzlich ohne den dynamischen Slot definiert wurde. Andernfalls ist diese Erweiterung der Antwortzeitberechnung gar nicht notwendig.

$$
\begin{aligned}
r^k_{g_t}(\Pi(\tau_s), TDMA_i) = \; & c_{g_t} \cdot \rho + B_{g_t} \\
& + \sum_{\forall g_j \in \check{C}^p_k : \phi_{g_j} > \phi_{g_t}} \left\lceil \frac{r^k_{g_t}(\Pi(\tau_s), TDMA_i) + J_{g_j}}{t_{g_j}} \right\rceil \cdot c_j \cdot \rho \\
& + \left\lceil \frac{r^k_{g_t}(\Pi(\tau_s)) - O_i^{X^{\Pi(\tau_s)}_{TDMA_i}}}{\Lambda(TDMA)} \right\rceil \cdot (B_i^{X^{\Pi(\tau_s)}_{TDMA_i}} + B_{dyn}) \\
& + \sum_{j=1, j\neq i}^{|X^{\Pi(\tau_s)}_{TDMA_i}|} \left\lceil \frac{r^k_{g_t}(\Pi(\tau_s)) - O_j^{X^{\Pi(\tau_s)}_{TDMA_i}}}{\Lambda(TDMA)} \right\rceil \cdot B_j^{X^{\Pi(\tau_s)}_{TDMA_i}},
\end{aligned}
\tag{4.25}
$$

wobei die Länge der Blockierung durch den dynamischen Slot dessen Länge entspricht, also gilt $B_{dyn} = \psi(s_{dyn})$. Weitere Maßnahmen sind nicht zu treffen, da aus Sicht eines TDMA-Verfahrens eine Blockierung immer auch als nicht diesem Knoten zugeordneter Slot wahrgenommen werden kann. Weitere Blockierungsslots können notwendig werden, wenn beispielsweise beim Übergang in den dynamischen Anteil des Protokolls bestimmte Sequenzen auf den Bus gelegt werden müssen (etwa Start-of-Cycle-Sequenzen, etc.).

Der Ereignis-basierte Anteil

Im dynamischen Anteil verhält es sich ähnlich: Es gibt eine Blockierung B_{TT} durch den TDMA-Anteil des Systems, der zyklisch wiederkehrt. Dementsprechend ist Gleichung 4.9 als Referenz der Antwortzeitberechung um eben diesen Blockierungsfaktor zu erweitern:

$$
r^n_{g_t} = \left(c^k_{g_t} - 1\right) \cdot \rho_k + \tilde{c}^k_{g_t} \cdot \rho_k + B_{g_t} + I^k_{g_t}(r^n_{g_t}) \cdot \rho_k + \left\lceil \frac{r^n_{g_t}}{\Lambda(TDMA)} \right\rceil \cdot B_{TT} \tag{4.26}
$$

Auch hier sind keine weiteren Maßnahmen erforderlich. Etwaige Start-of-Cycle-Sequenzen sind einmalig zu Beginn des dynamischen Slots auftretende Ereignisse, die in den Analysen des TDMA-Teils berücksichtigt werden und somit hier implizit in der Blockierungszeit B_{TT} enthalten sind. Die Blockierungszeit B_{TT} selbst setzt sich damit aus denjenigen Teilen der TDMA-Round zusammen, die nicht Teil des dynamischen Slots sind. Zusätzlich wird die Länge des dynamischen Slot noch weiter verkürzt: Da Nachrichten nur Frame-weise verschickt werden könnnen und eine einmalig begonnene Versendung nicht unterbrechbar ist, der dynamische Slot aber eine feste Länge besitzt, gibt es ab einem bestimmten Bereich vor dem Ende des dynamischen Slots keine Erlaubnis für ein weiteres Versenden. Frames, die nach diesem Zeitpunkt verschickt werden würden, könnten den Zeitrahmen des dynamischen Slots sprengen und würden das gesamte TDMA-Verfahren ad absurdum führen. Üblicherweise wird dieser Zeitpunkt so gesetzt, dass gerade noch ein maximaler Frame versendet werden kann. Aus Sicht der Antwortzeit-Analyse wird folglich der dynamische Slot um diese Zeit verkürzt und die Blockierungszeit B_{TT} ergibt sich durch:

$$B_{TT} = \Lambda(TDMA) - (\psi(s_{dyn}) - \rho) \tag{4.27}$$

Insgesamt obliegt es nun dem Optimierungsverfahren, welche Nachrichten welchem Teil des gemischten Kommunikationsmediums zugeordnet wird. Dabei ist sicherzustellen, dass es kein "Bus-Hopping" gibt, d.h. dass auf dem Weg von Prozessorknoten p_1 zu Knoten p_2 nicht durch mehrfaches Besuchen von anderen Knoten p_i[1] an dem gemischten Bussystem das jeweils andere Medium mehrfach benutzt wird. In Kapitel 5 sind Maßnahmen erläutert, die diesen Effekt in der Platzierungsoptimierung verhindern.
Gleichzeitig sind auch die Längen der jeweiligen Slots, also auch die Länge des dynamischen Slots, unter der Kontrolle des Optimierungsverfahrens. Hierbei werden sich interessante Rückschlüsse in den Messungen ergeben, wann welche Nachrichten besser im Ereignis-basierten Teil und wann besser im Zeit-basierten Teil des Mediums platziert werden.

4.3.4 Erhöhung der Flexibilität: Multiple dynamische Slots

Wie schon im vorhergehenden Abschnitt erläutert, leiden Zeit-basierte Kommunikationsmedien an der mangelnden Flexibität und führen daher zu schlechteren Busauslastungen als dies mit Ereignis-basierten Bussystemen möglich wäre. Trotzdem ist es aus Gründen der Vorhersagbarkeit in einigen Domänen notwendig, diese Art der Bussysteme einzusetzen. Die kombinierte Variante, wie es beispielsweise die FlexRay-Spezifikation vorsieht, führt zu Verbesserungen hinsichtlich der Flexibilität, hat aber dennoch Probleme bei sehr hochfrequenten, Ereignis-getriebenen Nachrichten. Die Länge einer TDMA-Round ist hier der begrenzende Faktor, da Tasks mit einer Periode kürzer als die Länge der TDMA-Round keine Möglichkeit haben, ihre Nachrichten in der vorgesehenen Frequenz abzusetzen. Um an dieser Stelle eine weitere Flexibilisierung zu ermöglichen, kann man das Konzept der multiplen Slot-Zuweisungen an einen Prozessorknoten innerhalb einer TDMA-Runde insofern erweitern, als dass man auch mehrere dynamische Slots in einer TDMA-Round erlaubt. Damit genügt ein solches System auch den Anforderungen von sehr hochfrequenten Tasks und weist aus Sicht der Zeit-basierten Nachrichtenverschickung die gleiche Qualität der zeitlichen Vorhersagbarkeit auf. Abbildung 4.16 macht dies an einem Beispiel deutlich.

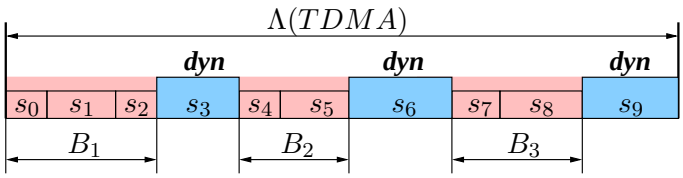


Abbildung 4.16: TDMA-basiertes Protokoll mit mehreren dynamischen Slots.

[1] In diesem Fall würden diese Knoten p_i als Gateways zwischen den beiden Teilsystemen genutzt.

In der praktischen Erprobung befindet sich zur Zeit der sogenannte Time Triggered CAN Bus (TTCAN)[61]. TTCAN basiert auf einem normalen CAN Bus, auf den in mehreren Ausbaustufen Zeit-basierte Anteile aufgesetzt werden. Die Idee ist hierbei, ein normales TDMA-Verfahren mit multipler Slotzuweisung zu erlauben. Jeder Slot ist entweder ein Zeit-getriebener, oder ein sogenannter Arbitrierungsslot. Im Ersteren hat nur ein bestimmter, zur Systemerstellung fest definierter, Prozessorknoten Zugang zum Bus. Im Letzteren verhält sich der Slot wie ein normaler CAN-Bus, vollzieht also eine Arbitrierung (daher der Name des Slots) von anstehenden Nachrichten. Diese maximale Flexibilität kann durch entsprechende Optimierungsverfahren ausgenutzt werden, um eine optimale Zuordnung von Nachrichten zu den verschiedenen Slots und ihren Betriebsarten zu wählen. Vorbedingung ist auch hierbei eine sichere Vorhersage des zeitlichen Verhaltens. Die auf CAN basierende Implementierung eines Kommunikationsmedium mit multiplen dynamischen Slots innerhalb einer TDMA-Round ermöglicht es nun, die in den verherigen Abschnitten hergeleiteten Berechnungsvorschriften für die Analyse der Antwortzeiten von Ereignis-basierten Nachrichten auf diesen Fall zu übertragen.

Aus Sicht der statischen Slots ist das zeitliche Verhalten der TDMA-Round mit multiplen dynamischen Slots identisch zu dem Verhalten, welches in Kapitel 25 eingehend erläutert wurde. Zusätzliche Slots für den dynamischen Anteil des Bussystems sind von anderen statischen Slots, die nicht dem unter Beobachtung stehenden Prozessorknoten zudeordnet sind, nicht zu unterscheiden. Für die Antwortzeit-Analyse kann folglich direkt auf Gleichung 4.19 zurückgegriffen werden.
Bezüglich der Ereignis-basierten Anteile verhält sich dies jedoch anders. Zum einen wurde bei der Analyse gemischter Kommunikationsmedien in Kapitel 4.3.3, Gleichung 4.26 nur von einem dynamischen Slot ausgegangen und folglich wird die Berechnung der Antwortzeiten von Nachrichten nicht von Blockierungen mit Offsets beeinflusst. Vielmehr existiert nur eine einzige Blockierung, die gemäß des Theorems der kritischen Instanz ohne Offset berücksichtigt wird. Eine Übertragung auf den nun hier vorliegenden Fall mehrerer Blockierungen, die das Zeitfenster für Übertragungen unterbrechen können, verlangt nach einer den multiplen Slots in den Zeit-basierten Anteilen analogen Vorgehensweise. Zum anderen sind die dynamischen Anteile nicht gänzlich nutzbar, da — wie in Gleichung 4.27 ausgeführt — es in jedem dynamischen Slot einen Zeitpunkt vor dem Ende des Slots gibt (LPOS, Last Point Of Sending), ab dem keine weitere Nachricht übertragen werden darf. Antwortzeit-Analysen auf den dynamischen Anteilen müssen diese Teile der Ereignis-basierten Slots folglich als zusätzliche Blockierung betrachten; sie dienen nicht der Nachrichtenübertragung[1]. Letztlich können diese Anteile eines dynamischen Slots als zusätzlich statische Slots angesehen werden, die eine Blockierung auslösen. Abbildung 4.17 macht dies an einem Beispiel deutlich:

[1]Streng genommen ist dies eine Überapproximation, da innerhalb dieses Zeitintervalls zwar keine neuen Nachrichten mehr versendet werden können, jedoch die Antwortzeit von Nachrichten, deren Versendung vor dem Start des verbotenen Intervalls liegen, sich durchaus in den verbotenen Bereich erstrecken kann. Demzufolge ist die Blockierung bei einer exakteren Analyse als bedingte Blockierung zu betrachten, die sich nur auf den Startzeitpunkt einer Nachrichtenversendung bezieht. Im folgenden soll dies allerdings nicht weiter betrachtet werden, sondern es wird die skizzierte Überapproximation verwendet.

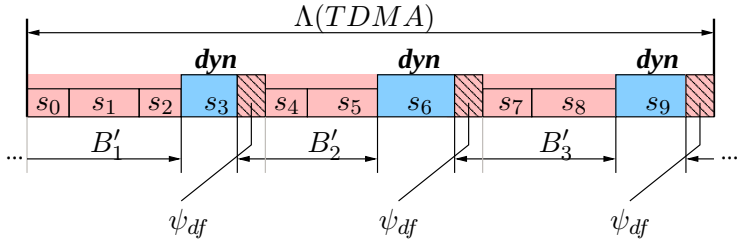


Abbildung 4.17: Umwandlung der verbotenen Sendezonen eines dynamischen Slots in synthetische Slots der TDMA-Round.

Für die Übertragung der in Kapitel 25 dargestellten Verfahren auf die Antwortzeitberechnung im Ereignis-basierten Teil des gemischten Bussystems mit multiplen dynamischen Slots wird daher zunächst aus der gegebenen TDMA-Round eine um diese synthetischen Slots angereicherte TDMA-Round erzeugt. Dabei wird davon ausgegangen, dass die verbotenen Intervalle in den einzelnen dynamischen Slots jeweils die identische Länge ψ_{df} besitzen. Die Erzeugung der neuen TDMA-Round erfolgt einfach auf der Veränderung der jeweiligen Slotlängen: Jeder dynamische Slot wird in seiner Länge um ψ_{df} Zeiteinheiten gekürzt, während jeder direkt einem dynamischen Slot folgende Slot (der kein dynamischer sein kann) um ψ_{df} Zeiteinheiten verlängert wird. Sei n die Anzahl der Slots einer TDMA-Round. Dann ergeben sich für die veränderte TDMA-Round die folgenden Längen der Slots s_i mit $0 \leq i < n$:

$$\psi'(s_i) = \begin{cases} \psi(s_i) - \psi_{df} & \text{wenn } type(s_i) = \text{dyn} \\ \psi(s_i) + \psi_{df} & \text{wenn } i > 0 \wedge type(s_{i-1}) = \text{dyn} \\ \psi(s_i) + \psi_{df} & \text{wenn } i = 0 \wedge type(s_{n-1}) = \text{dyn} \\ \psi(s_i) & \text{ansonsten} \end{cases} \tag{4.28}$$

Diese Umstellung ist nur für die Sicht der dynamischen Slots erlaubt, verändert aber ansonsten die TDMA-Runde nicht. Da in der Analyse der Antwortzeit ohnehin auf rotierten Instanzen der TDMA-Round operiert wird, hat diese Veränderung keine weiteren Auswirkungen als die gewünschte Verkürzung der dynamischen Slots und die notwendige Verlängerung der Blockierungen durch statische Slots.
Mithilfe dieser Umstellung kann nun das Instrumentarium aus Abschnitt 25 benutzt werden, wobei in den Definitionen 4.16 für $X^{\text{dyn}}_{TDMA_i}$, 4.17 für $B_i^{X^{\text{dyn}}_{TDMA_i}}$ und 4.18 für $O_{s_i}^{X^{\text{dyn}}_{TDMA_i}}$ jeweils ψ durch ψ' zu ersetzen ist. Durch die Verwendung mehrfacher Blockierungen mit Offsets müssen zur Berechnung der Antwortzeit auch in diesem Fall alle möglichen Verschiebungen der TDMA-Round betrachtet werden. Mithilfe der Gleichung 4.26 ergibt sich demnach für eine mögliche Verschiebung $TDMA_i^{\text{dyn}}$ der TDMA-Round die folgende Antwortzeit:

$$\begin{aligned} r^k_{g_t}(TDMA_i^{\text{dyn}}) = \; & \left(c^k_{g_t} - 1\right) \cdot \rho_k + \tilde{c}^k_{g_t} \cdot \rho_k + B_{g_t} + I^k_{g_t}(r^n_{g_t}) \cdot \rho_k \\ & + \sum_{j=1}^{|X^{\text{dyn}}_{TDMA_i}|} \left\lceil \frac{r^k_{g_t}(TDMA_i^{\text{dyn}}) - O_j^{X^{\text{dyn}}_{TDMA_i}}}{\Lambda(TDMA^{\text{dyn}})} \right\rceil \cdot B_j^{X^{\text{dyn}}_{TDMA_i}} \end{aligned} \tag{4.29}$$

Auch hier muss über die Fomulierung eines entsprechenden Lemma sichergestellt werden, dass die betrachteten Verschiebungen nur an den Grenzen der dynamischen Slots hinreichend sind, die Antwortzeitberechnung zu maximieren. Der Beweis dieses Lemma ist trivialerweise vollkommen analog zu dem Beweis von Lemma 4.3.1, da bei der Aufstellung der Beweisverpflichtung alle für den Ereignis-basierten Fall notwendigen Anteile herausfallen und sich der Beweis somit auf den Beweis von 4.3.1 reduziert.
Demzufolge lässt sich die Antwortzeit einer Ereignis-basierten Nachricht auf den dynamischen Anteilen eines gemischten Kommunikationsmediums mit multipler dynamischer Slotverwendung durch die folgende Gleichung berechnen:

$$r_{g_t}^k(TDMA^{\text{dyn}}) = \max_{\forall i \in \{1,\ldots,|X_{TDMA}^{\text{dyn}}|\}} \left(r_{g_t}^k(TDMA_i^{\text{dyn}}) \right) \tag{4.30}$$

4.3.5 Gemischt-betriebene Echtzeitbetriebssysteme

Die im vorangegangenen Abschnitt herausgearbeitete Antwortzeitanalyse auf Kommunikationsmedien mit mehreren dynamischen Slots eröffnet ein weiteres Anwendungsszenario: Die Verwendung von gemischten Echtzeit-Betriebssystemen, in denen ein Zeit-basiertes Schedulingverfahren mit einem Ereignis-basierten RTOS kombiniert wird. Die Vorteile einer solchen Kopplung unterschiedlicher Paradigmen liegt in der Flexibilität bzgl. heterogener Systemanforderungen. So ist es beispielsweise meist nicht möglich, alle Tasks gemäß ihrer Requirements streng periodisch auszulegen, wie dies für Zeit-basierte Schedulingverfahren aber unumgänglich ist, da es in nahezu jeder Anwendungsdomäne eine Reihe von sporadisch auftretenden Ereignissen gibt, die von Tasks bearbeitet werden müssen. Ein Beispiel hierfür sind etwa von der Drehzahl des Motors abhängige Aktionen im Automobil, die zwar aufgrund der physikalischen Gegebenheiten eine minimale Zwischenankunftzeit garantieren können, deren wirkliches zeitliches Auftreten aber nicht vorhersehbar ist. Derartige Anforderungen können in Zeit-basiertem Scheduling nur abgedeckt werden, wenn die verursachenden Sensoren durch ein zyklisches Pollen abgefragt werden und im Falle einer Wertänderung diese anschließend in dem dann statisch auftretenden Slot behandelt werden kann. Polling Tasks verursachen aber anderseits eine höhere Systemlast. Basierend auf [90] konnte in [115] gezeigt werden, dass die Periode einer pollenden Tasks der Hälfte der Deadline des das Ereignis bearbeitenden Tasks betragen muss. Selbst für den Fall, dass die bearbeitende Task nicht über eine, aufgrund ihrer Zugehörigkeit zu einer langen Taskkette notwendigen, sehr kurzen Deadline verfügt, bedeutet dies zumindest eine Verdopplung der eigentlich notwendigen Auslastung des Prozessors. Da dies unter Umständen nicht gewünscht ist, kann man sich hier mit einer gemischten Betriebssystemvariante behelfen, indem die sporadischen Tasks im Ereignis-basierten Teil bearbeitet werden und alle streng periodischen auf den Zeit-basierten Teil abgebildet werden.

Beispielhaft für ein derartiges gemischtes Zeit/Ereignis-basiertes Echtzeitbetriebssystem soll hier eine Implementierungsvariante von OSEKtime[134] dargestellt werden. Abbildung 4.18 stellt einen Abschnitt aus der OSEKtime Spezifikation vor, in der der gemischte Betriebsmodus dargestellt ist.

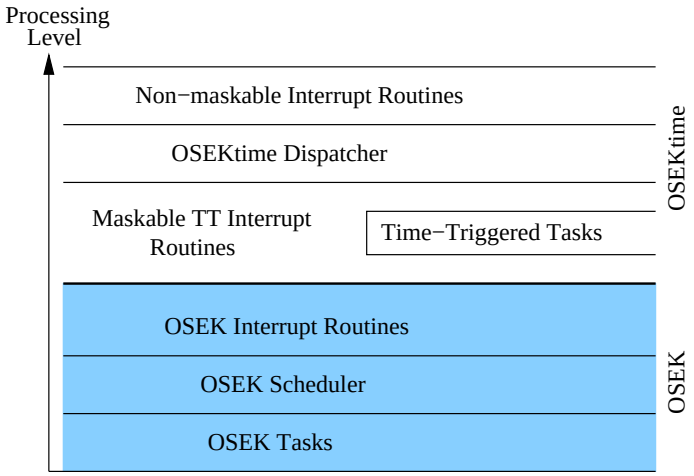


Abbildung 4.18: Processing-Level einer gemischt-betriebenen OSEK[133] Variante, die aus einem übergeordneten OSEKtime[134] und einem untergeordneten Standard-OSEK RTOS besteht.

Die grundlegende Idee hierbei ist, den Zeit-basierten Anteil dem Ereignis-basierten Standard-OSEK übergeordnet zu implementieren. Der Zeit-basierte Schedule kann nun so gewählt werden, dass in den Zeitintervallen, in denen keine Zeit-basierten Tasks laufen, der preemptive, prioritätengesteuerte Scheduler Ausführungszeit erhält und Ereignis-basierte Tasks abarbeitet, jedoch jederzeit von OSEKtime unterbrochen werden kann.
Aufgrund der strengen Festlegung der zeitlichen Punkte, an denen OSEKtime Tasks dispatched, entsteht ein zu gemischten Bussystemen des letzten Abschnitts äquivalentes Bild: Feste statische Slots sind gemäß einer TDMA-Strategie vergeben, und die freien Zwischenintervalle entsprechen dynamischen Slots, die ebenfalls an festen Zeitpunkten initiiert werden.
Ebenso wie für gemischte Kommunikationsmedien gibt es in der Literatur nahezu keinen Ansatz, der sich mit der Antwortzeitanalyse von Ereignis-basierten Tasks in solchen Systemen beschäftigt. Lediglich in [145] wird eine Erweiterung der in [181] vorgenommenen Formalisierung vorgeschlagen, die jedoch voraussetzt, dass alle dynamischen Anteile zu einem einzigen dynamischen Slot verbunden werden. Ist diese Vorbedingung erfüllt, können die Analysen aus Kapitel 4.3.3[1] angewendet werden. Allerdings widerspricht dieses Vorgehen zum einen den Beobachtungen, die zur Einführung multipler Slots in einer TDMA-Round führten und zum anderen bildet es das reale Verhalten eines gemischt betriebenen Betriebssystems nicht korrekt ab, da ebenjene Zusammenfassung der dynamischen Anteile dort nicht passieren kann. Stattdessen soll hier die Analysemethodik für Kommunikationssystemen mit multiplen dynamischen Slots übertragen werden auf Antwortzeitanalysen für das Scheduling von Tasks.
Zunächst kann festgestellt werden, dass Tasks im Gegensatz zu Nachrichten preemptiv sind, es also keine Dispatch-freien Zeitintervalle innerhalb eines dynamischen Zeitabschnitts des Scheduling gibt. Demzufolge können die Ergebnisse aus

[1] [145] verwendet eine andere Art der Notation, die aber in ihrer Aussagekraft identisch zu den in Kapitel 4.3.3 notierten Analysen sind

4.3.4 direkt mit den Analysen für Tasks in preemptiven, prioritäten-gesteuerten Schedulingverfahren kombiniert werden. Letztlich ergibt sich also für die Antwortzeitberechnung wiederum die Notwendigkeit, alle notwendigen Verschiebungen der TDMA-Round zu betrachten und mittels Maximumsbildung die schlimmste anzunehmende Antwortzeit zu ermitteln:

$$r_i = \max_{\forall j \in \{1,\ldots,|X_{TDMA^{TT}}|\}} \left(r_i(TDMA_j^{TT}) \right) \tag{4.31}$$

mit

$$\begin{aligned} r_i(TDMA_i^{TT}) = \; & c_i(\Pi(\tau_i)) + \sum_{\tau_h : \phi_h > \phi_i \wedge \Pi(\tau_h) = \Pi(\tau_i)} \left\lceil \frac{r_i(TDMA_i^{TT}) + J_h}{t_h} \right\rceil \cdot c_h(\Pi(\tau_h)) \\ & + B_i + \sum_{j=1}^{|X_{TDMA_i^{TT}}|} \left\lceil \frac{r_i(TDMA_i^{TT}) - O_j^{X_{TDMA_i^{TT}}}}{\Lambda(TDMA)} \right\rceil \cdot B_j^{X_{TDMA_i^{TT}}} \end{aligned} \tag{4.32}$$

wobei eine TDMA-Round $TDMA^{TT}$ aus den Startzeitpunkten s_j^{TT} der Zeit-basierten Tasks τ_j^{TT} bezüglich einer periodisch wiederkehrenden Sequenz und deren Laufzeiten, also der WCET einer Task, erzeugt wird. Damit gilt für die Länge eines statischen Slot des Zeit-basierten Teiles:

$$\psi(\tau_j^{TT}) = c_i(\Pi(\tau_j^{TT})) \tag{4.33}$$

Selbst wenn das zeit-basierte Scheduling preemptiv organisiert ist, kann aus Sicht der ereignis-basierten Tasks im schlimmsten Fall tatsächlich von einer festen Slotlänge ausgegangen werden. Alle innerhalb der periodischen Abfolge von Slots freien Zeitintervalle werden damit zu dynamischen Slots, deren zeitliche Längen eindeutig (und sicher im Sinne einer Worst-Case-Analyse) approximiert werden können.

4.3.6 Experimentelle Ergebnisse für multiple Slot-Systeme

Zur Einschätzung des Vorteils der in den letzten Kapiteln dargestellten Analyse-Methoden für Systeme, die gemischte Formen von Ereignis- und Zeit-getriebenen Subsystemen verwenden, soll hier eine Evaluation typischer Systemkonfigurationen folgen. Wie bereits eingehend dargelegt, ähneln sich die Analyseverfahren für z.B. FlexRay, TT-CAN oder gemischte Betriebssysteme, so dass wir uns hier für die experimentelle Messung auf ein gemischtes Betriebssystem beschränken. Die Ergebnisse sind jedoch aufgrund eben dieser Ähnlichkeiten ohne weiteres auch auf die anderen erwähnten Systeme übertragbar. Im Fokus dieser Experimente liegt der dynamische Teil, also Ereignis-basierte Tasks, die beispielsweise unter einem gemischt betriebenen OSEK-Time ihre Ausführung versehen müssen.
Im folgenden wird die in den vorangegangenen Kapiteln hergeleitete Analyse für Systeme mit multiplen Slots *MSA*-Analyse[1] genannt werden. Zwecks Vergleichbarkeit muss zunächst betrachtet werden, wie Systeme dieser Art mit der herkömmlichen

[1] Multiple Slot Access

Analysetechnik berechnet werden können. Angesichts der Probleme beim Finden der kritischen Instanz (vgl. Abbildung 4.10 auf Seite 73) bedeutet dies für eine traditionelle Analyse, dass alle Blockierungen ohne Offsets betrachtet werden müssen (ein Weg, der auch in [145] beschritten wird): Alle Tasks *und* alle Blockierungen durch den Zeit-basierten Anteil starten im worst case zum selben Zeitpunkt. Dies kann man mit der klassischen Analyse für die Antwortzeit von Tasksystemen unter Berücksichtigung von Blockierungen modellieren. Das Ergebnis einer solchen Analyse ist eine sichere, aber überapproximierte Antwortzeit. Die folgenden Messungen für typische Systemlasten sollen nun klären, in welchen Größenordnungen die Überapproximation im Vergleich zu der genaueren MSA-Analyse liegt.

Abbildung 4.19 zeigt die Ergebnisse für ein gemischt-betriebenes Betriebssystem, welches 50% der Auslastung Zeit-basierten Tasks und die anderen 50% Ereignis-basierten Tasks zuordnet. Die Größe und Verteilung der einzelnen Slots wurde zufällig gewählt, die Anzahl der Slots ist parametrisiert (und durch $|TDMA|$ angegeben, wobei z.B. eine Mächtigkeit von 20 die entsprechende Anzahl an Slots meint und damit folglich 10 Zeit-basierte Tasks im System existieren, auf deren Slot jeweils ein dynamischer Slot folgt). Das dynamische Taskset besteht aus 5 Tasks mit einer Periode zwischen 200ms und 600ms und die WCETs jeder Task liegt zwischen 15ms und 35ms. Deadlines entsprechen der Periode und die Priorisierung wurde entsprechend dem Ratenmonotonen Verfahren vorgenommen. Die Länge der TDMA-Round variiert zwischen 20ms und 200ms, was der minimalen Periode der Ereignis-getriebenen Tasks entspricht.

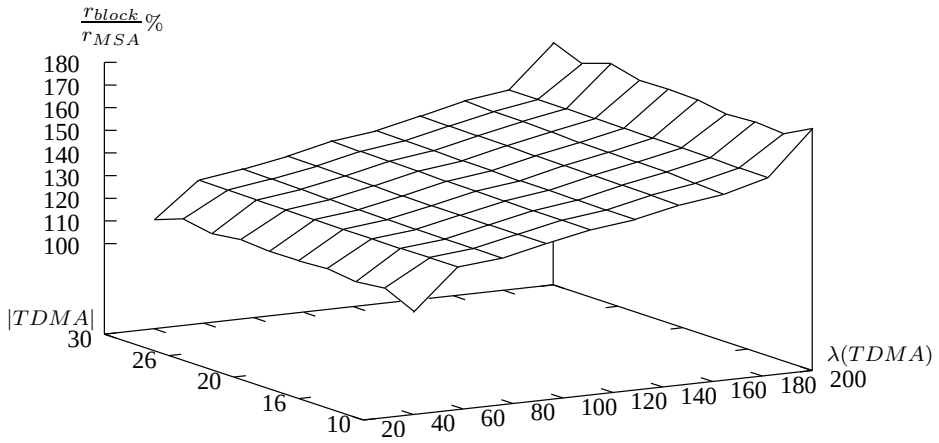


Abbildung 4.19: Verhältnis der Antwortzeiten für die traditionelle, auf Blockierung basierende, Analyse und das im letzten Kapitel dargestellte Analyse-Verfahren.

Man kann in Abbildung 4.19 deutlich erkennen, dass die traditionelle Analyse mit steigender Länge der TDMA-Round deutliche Überapproximationen liefert. Dargestellt ist das durchschnittliche Verhältnis der Antwortzeiten der Tasks in Prozent, die einmal mittels der traditionellen Analysemethode und einmal mittels des MSA-Verfahrens errechnet wurden. Beispielsweise sagt die traditionelle Analyse für eine typische TDMA-Round der Länge 160ms und mit 20 Slots im Durchschnitt eine 1,5 mal größere Antwortzeit voraus, als die MSA-Analyse ermittelt. Bzgl. der Anzahl

der Slots lässt sich eine sehr leichte Steigung der Überapproximation beobachten, wenn die Slotanzahl anwächst. Dieser Effekt ist aber mit Vorsicht zu genießen, da — wie wir im nächsten Experiment noch sehen werden — die Größe und Verteilung der einzelnen Slots einen wesentlich höheren Einfluss auf die Größenordnung der Überapproximation hat als die Anzahl der Slots selbst.
Die Interpretation dieses Ergebnisses bestätigt die Erwartungen, da mit einer ansteigenden Länge der TDMA-Round die Anzahl der TDMA-Runden, die innerhalb der Antwortzeitberechnung betrachtet werden, abnimmt. Dies wiederum bedeutet, dass der Einspareffekt der MSA-Analyse (nicht grundsätzlich immer die komplette TDMA-Round in der Analyse berücksichtigen zu müssen) stärker ausgeprägt ist: Aufgrund der Formelstruktur der Berechnungen ist klar, dass mehrfaches Vorkommen der kompletten TDMA-Round innerhalb der Antwortzeit in beiden Analysverfahren den gleichen Anteil liefert. Die letzte TDMA-Round jedoch muss nicht komplett aufgenommen werden, so dass dies der überapproximierende Anteil in der traditionellen Analyse ist. Dementsprechend wird dies bei einer längeren TDMA-Round einen größeren Effekt hervorrufen. Liegen die Längen der TDMA-Round über den Perioden der Tasks, wird es zunehmend schwieriger, Tasksysteme mittels der traditionellen Analysemethode als Feasible zu deklarieren, während dies mittels der MSA-Methode noch gelingt.
Insgsamt zeigen die Daten aus Abbildung 4.19, dass für das betrachtete Tasksystem die MSA-Analyse auch noch den Nachweis der Feasibility erbringen kann, wenn die Deadlines der Tasks deutlich kleiner sind, als dies für die traditionelle Analyse möglich wäre.

Ein wichtiger Faktor, wie oben angesprochen, ist auch die Struktur einer TDMA-Round selbst, d.h. wie lang sind welche Slots und wie regelmäßig sind sie angeordnet. Um diesen Effekt zu untersuchen, wird im folgenden ein Beispiel mit einer typischen Systemlast als Basissystem genommen und nach und nach die Varianz der Slotlängen erhöht. Diese Erhöhung bedeutet, dass die Slots unterschiedlicher in ihrer Länge und Verteilung werden, ohne jedoch die Slotanzahl selbst zu erhöhen. In Tabelle 4.1 ist der Einfluss der Varianz der Slots bezüglich der Überapproximation des traditionellen Verfahrens dargestellt.

	$\sigma = 0.0$	$\sigma = 2.4$	$\sigma = 4.5$	$\sigma = 8.0$	$\sigma = 32.0$	$\sigma = 80.0$	$\sigma = 120.0$
160/20	169%	156%	154%	149%	136%	120%	112%

Tabelle 4.1: Abhängigkeit der Überapproximation von der Varianz der TDMA-Round bzgl. der Slotlängen bei konstanter Slotanzahl und konstanter Länge der TDMA-Round.

Basissystem dieser Messung ist eine TDMA-Round mit 20 Slots (entspricht 10 Zeitbasierten Tasks) und einer Länge von 160ms. Man kann deutlich erkennen, dass die Überapproximation des traditionellen Analyseverfahrens abnimmt, wenn die Varianz der Slots ansteigt. Dieser Effekt lässt sich leicht durch die Form der MSA-Analyse erklären: Bei steigender Varianz mit gleichzeitig gleichbleibender Slotanzahl steigt die Länge einiger weniger Slots drastisch an, während es auf der anderen Seite viele kur-

ze Slots gibt. Da die kritische Instanz über alle Verschiebungen der TDMA-Round ermittelt wird, wird sich das Verhalten mit steigender Varianz dem der traditionellen Analyse angleichen, also diejenige Instanz gewählt werden, die sicherstellt, dass die größten Blockierungen am Anfang zu betrachten sind. Würde man die Länge der kleiner werdenden Slots asymptotisch gegen 0 laufen lassen, so würde die MSA-Analyse auf exakt die traditionelle Analyse reduziert werden. Bei der Planung der Slotpositionen eines gemischten Echtzeitbetriebssystems ist folglich einer der Optimierungsparamter, eine möglichst geringe Varianz in den Slotlängen und deren Verteilung zu erzeugen.

Die letzte hier betrachtete Experimentenreihe soll den Einfluss einer steigenden Auslastung der Ereignis-basierten Tasks auf die Feasibility-Aussage der beiden Verfahren zeigen. In Tabelle 4.2 ist dieser Effekt wiederum für das oben schon verwendete Beispiel dargestellt. Die Auslastung $U_{\mathcal{T}}$ wurde durch das Hinzufügen weiterer Ereignis-basierter Tasks erreicht und es zeigt sich, dass sich durch die Verwendung der MSA-Methode in diesem Beispiel eine etwa 10% höhere Auslastung erzielen lässt.

	$U_{\mathcal{T}} = 0.31$	$U_{\mathcal{T}} = 0.38$	$U_{\mathcal{T}} = 0.40$	$U_{\mathcal{T}} = 0.42$	$U_{\mathcal{T}} = 0.44$
block	✓	✓	✗	✗	✗
MSA	✓	✓	✓	✓	✗

Tabelle 4.2: Abhängigkeit der Feasibility unterschiedlicher Auslastungen von der traditionellen (*block*) und der MSA-Analyse.

Letztlich zeigt sich, dass die MSA-Analyse, wie wir sie in den vorangegangenen Kapitel vorgestellt haben, deutlich exaktere Vorhersagen bzgl. des Antwortzeitverhaltens liefern kann. Zudem wird auch deutlich, dass die Dimensionierung der einzelnen Slots weitreichende Einflüsse auf die Feasibility eines Ereignis-basierten Sub-Systems haben kann, auch wenn der Anteil an Auslastung, der dem Ereignis-basierten Sub-System zugebilligt wird, konstant ist. Alle diese Ergebnisse lassen sich natürlich mit den entsprechenden Anpassungen auch auf die Antwortzeitanalyse von Ereignis-basierten Nachrichten in MSA-Bussen, wie FlexRay oder TT-CAN übertragen.

4.3.7 Zusammenfassung

Insgesamt liegt damit nun eine umfangreiche Scheduling-Analyse für alle möglichen verschiedenen Arten der Nachrichtenübermittlung über Kommunikationsmedien innerhalb einer Architektur vor. Die formale Natur dieser Analysen erlaubt es, Ereignis-basierte Kommunikationen exakt in einem Optimierungsverfahren zu behandeln und stellt damit den erforderlichen Mechanismus zur Verfügung, um komplexe Systeme in ihrer Ganzheit einer automatischen Platzierung zuzuführen. Zusätztlich wurde gezeigt, dass die Methodik auch und gerade auf den Bereich der gemischten Zeit-/Ereignis-basierten Echtzeitbetriebssysteme übertragbar ist. Die Fragen des Routings (also über welchen Medien und Knoten laufen Nachrichten in komplexen, hierarchisch organisierten Bussystemen) ist vom Optimierungsverfahren zu leisten und wird in Kapitel 5 eingehend dargestellt. Bevor dies näher erläutert

wird, wollen wir uns aber zunächst dem Zusammenspiel von Taskausführung und Nachrichtenübermittlung in verteilten Systemen widmen.

4.4 Einfluss der Verteilung und Topologie

4.4.1 Wechselseitige Abhängigkeit von Jitter und Antwortzeit

Obwohl in den Abschnitten 4.2 und 4.3 vollständige Formalisierungen der Antwortzeitberechnungen von Tasksystemen und Kommunikationen angegeben wurden, reicht es jedoch nicht aus, diese Analysen isoliert und vor allem unabhängig von der Platzierung zu betrachten. Der wechselseitige Einfluss von Task- und Kommunikationslaufzeiten auf den Jitter abhängiger und unabhängiger Tasks in ereignisbasierten Schedulingmechanismen[1] macht eine dynamisch wiederkehrende globale Systemanalyse notwendig. Diese findet entweder einen Fixpunkt für alle zu betrachtenden Zeitkriterien oder aber überschreitet eine der gegebenen Deadlines. Dieser Effekt ist erstmals durch Tindell in [181] durch die sogenannte holistische Analyse quantifiziert worden. In [178] ist die holistische Analyse beispielsweise mit einer automatischen Platzierungsberechnung verbunden worden.
Zusätzlich bedeutet die globale Systemanalyse, dass der Vorteil der rein lokalen Analyse, wie er beispielsweise durch die lokale Deadlinesynthese erzeugt wurde, nicht direkt ausgenutzt werden kann.
Wechselseitige Abhängigkeiten der Antwortzeiten von Tasks und Nachrichten manifestieren sich im Jitter abhängiger und unabhängiger Tasks. Zur Illustration soll das folgende Beispiel aus Abbildung 4.20 dienen.

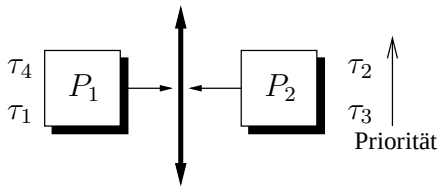


Abbildung 4.20: Beispielarchitektur mit einer gegebenen Verteilung von 4 Tasks, die sich miteinander über ein gemeinsames Kommunikationsmedium im Sinne einer kausalen Aktivierungskopplung Nachrichten austauschen.

Vier Tasks, von denen jeweils 2 über eine Nachricht aneinander gekoppelt sind, werden gemäß der Abbildung auf einer verteilten Architektur platziert. Die Kommunikationsstruktur sei definiert durch: $\tau_1 \overset{m_1}{\rightarrow} \tau_2$ und $\tau_3 \overset{m_3}{\rightarrow} \tau_4$. Die Platzierung sei mit $\Pi(\tau_1) = \Pi(\tau_4) = P_1$ und $\Pi(\tau_2) = \Pi(\tau_3) = P_2$ gegeben. Prioritäten werden gemäß der Abbildung 4.20 vergeben, d.h. für den jeweiligen Knoten gilt: $\phi_4 > \phi_1$ und $\phi_2 > \phi_3$.

[1] Zeit-getriebene Mechanismen leiden nicht an diesem Problem, da dort der Jitter aufgrund der strengen Kopplung einer Taskaktivierung an einen festen Zeitpunkt grundsätzlich nicht vorhanden ist

Betrachtet man die Antwortzeiten ω_i[1] der niedriger priorisierten Tasks auf den jeweiligen Knoten gemäß der Analyse aus Gleichung 4.3 und vernachlässigt dabei Kommunikationen und Blockierungsfaktoren, so ergibt sich:

$$\omega_1 = \text{fixpoint}\left(r_1 = c_1 + \left\lceil \frac{r_1 + J_4}{t_4} \right\rceil \cdot c_4\right) := f_1(J_4)$$
$$\omega_3 = \text{fixpoint}\left(r_3 = c_3 + \left\lceil \frac{r_3 + J_2}{t_2} \right\rceil \cdot c_2\right) := f_3(J_2)$$

D.h. die Antwortzeit der Tasks kann als Funktion über den Jitter der höher priorisierten Tasks dargestellt werden. Vom Jitter selbst ist bekannt, dass er direkt von der Antwortzeit abhängt, da diese das Verschmieren des Aktivierungszeitpunktes einer abhängigen Task bestimmt (die genauen Zusammenhänge dazu sind weiter unten beschrieben, hier genügt diese abstrakte Information zum Verständnis). Daraus folgt, dass die jeweiligen Jitter als Funktionen von den Antwortzeiten der Vorgängertasks aufgefasst werden können.

$$J_4 = j_4(\omega_3) \Rightarrow r_1 = f_1(j_4(\omega_3)) = \ldots = f_1(f_3(f_2(f_1)))$$
$$J_2 = j_2(\omega_1) \Rightarrow r_3 = f_3(j_2(\omega_1)) = \ldots = f_3(f_1(f_4(f_3)))$$

Die Auflösung der Zusammenhänge dieses Beispiels liefert schließlich für die Funktion der Antwortzeiten der niedriger priorisierten Tasks eine zyklische Abhängigkeit. Zyklische Abhängigkeiten dieser Gestalt können mathematisch nur mittels einer Fixpunktberechnung gelöst werden. Diese Fixpunktberechnung ist systemweit, außerhalb der lokalen Fixpunktberechnungen für die Antwortzeiten gemäß Gleichung 4.3, durchzuführen.
Insbesondere wird damit deutlich, dass das Hinzufügen neuer Tasks oder Kommunikationen nicht lokal betrachtet werden kann, da sie das gesamte System durch den oben dargestellten Effekt beeinflussen und die holistische Analyse (also die systemweite zusätzliche Fixpunktberechnung) das Einschwingen des Gesamtsystems auf die lokalen Veränderungen berechnen muss.
In letzter Konsequenz bedeutet dies, dass zumindest Anteile des Jitters von Tasks nicht fest vorgegeben sind, sondern von der der aktuellen Verteilung von Tasks und Kommunikationen bestimmt werden. Gerade im Entwicklungsprozess von Zulieferer-dominierten Systemen ist aber eine inkrementelle Systementwicklung mit der Möglichkeit zur Kapselung bestimmter Anteile der Funktionalitäten wünschenswert. Dies war auch einer der wesentlichen Gründe, in Abschnitt 4.2 die Synthese von lokalen Deadlines einzuführen und somit auch den ereignis-basierten Systemen einen Zugang zum inkrementellen, komponenten-basierten Entwurf zu ermöglichen. Für eine weitergehende und detaillierte Argumentation bezüglich der inkrementellen Integration sei der Leser hier auf Kapitel 6 verwiesen; An dieser Stelle soll zunächst die Information genügen, dass es einen inkrementellen Integrationsprozess gibt und dieser unter Entwurfsprozess-Gesichtspunkten notwendig ist.
Um dem Ideal des inkrementellen Entwurfes auch für ereignis-basierte Systeme

[1] ω_i bedeutet hier der Fixpunkt der Gleichung 4.3 für Task τ_i.

zu genügen, wird im folgenden statt einer exakten Jitterberechnung eine Überapproximation eingeführt, die sich direkt aus den abgeleiteten lokalen Deadlines ergibt. Diese Überapproximation stellt eine sichere obere Grenze des jeweiligen Jitter-Wertes dar und liefert damit eine worst-case Betrachtung des Systems. Überapproximierte Jitter-Werte sind zudem platzierungsunabhängig und können somit direkt als Schnittstelle in einem komponenten-basierten Entwurf genutzt werden. Allerdings bedeutet die Berücksichtigung einer Worst-Case-Annahme auch, dass ggf. eine zu pessimistische Betrachtung stattfindet und somit Platzierungs- und Scheduling-Spielraum vergeben werden.
Jitter in abhängigen Taskketten stellt unterschiedliche Aktivierungszeitpunkte einer Task τ_i dar und wird durch die verschiedenen möglichen Laufzeiten der Vorgängertasks und Nachrichten des Kontrollflusses $\tau_0 \overset{m_0}{\rightarrow} \ldots \overset{m_{i-2}}{\rightarrow} \tau_{i-1} \overset{m_{i-1}}{\rightarrow}$ einer Taskkette verursacht. Unterschiedliche Bedingungen zum Aktivierungszeitpunkt der Vorgängertasks und Nachrichten können dazu führen, dass die jeweiligen Antwortzeiten aus den Scheduling- und Kommunikationsanalysen unterschritten werden. Dies ist möglich, weil diese Analysen den schlimmsten auftretenden Fall untersuchen und nur darüber verlässliche Aussagen machen. Eine exakte Berechnung des Jitters ist aber nur unter Verwendung des bestmöglichen Falles möglich, kann jedoch durch Über- bzw. Unterapproximationen angenähert werden. So ist es beispielsweise durchaus üblich (vgl. auch [181]), die bestmögliche Laufzeit einer Task in der Jitterberechnung gleich 0 zu setzen. Der daraus resultierende zu große Wert für den Jitter einer Task ist unter worst-case Gesichtspunkten, wie sie die den Jitter verwendenden Scheduling-Analysen darstellen, trotzdem korrekt, liefert aber ggf. eine zu große Anzahl an Preemptionen. Allgemein kann der Jitter einer Task τ_i durch eine Vorgängertask τ_j durch die folgende Gleichung berechnet werden, wenn $\omega(m)$ die worst-case Laufzeit einer Nachricht, $\beta(m)$ die best-case Laufzeit und b_t die Best-Case Antwortzeit einer Task ist:

$$J_i = \big(r_j - b_j\big) + \big(\omega(m_j) - \beta(m_j)\big) \tag{4.34}$$

Wie oben schon beschrieben, sind diese Werte abhängig von der Platzierung der Tasks und Nachrichten innerhalb der Topologie des verteilten Systemes. Überapproximationen bzw. Unterapproximationen dieser Werte lassen sich aber leicht finden:

- In einem System, dass den Nachweis der Feasibility erbracht hat, ist bekannt, dass die Antwortzeit r_j einer Task τ_j immer kleiner oder gleich ihrer (synthetischen) Deadline $\Delta_\tau(\tau_j)$ ist.

- Die Best-Case Antwortzeit ist abhängig von der Platzierung weiterer Tasks auf dem Prozessorknoten, kann aber von unten durch die BCET des Task approximiert werden.

- Nachrichten sind unter der Worst-Case-Analyse analog den Tasks zu behandeln, so dass als Überapproximation ihrer Laufzeit ebenfalls die Deadline $\Delta_\gamma(m_j)$ benutzt werden kann.

- Best-Case Laufzeiten von Nachrichten hängen von den verwendeten Bussystemen ab und können für die kürzeste Version einer Nachricht unter der Annahme, den schnellsten Bus alleine zu verwenden, berechnet werden.

Wendet man diese Annäherungen auf die Jitterberechnung in Gleichung 4.34 an, so resultiert eine platzierungs-unabhängige Wertermittlung für den Jitter von Tasks. Dieses Verfahren kann vollkommen analog auch für den Jitter von Nachrichten eingesetzt werden. Für allgemeine Taskketten ist zu berücksichtigen, dass Jitter-Effekte nicht an streng zeitaktivierten Tasks ($tr_j = TIME$) auftreten. Das bedeutet für die Jitterberechnung einer Task τ_i, dass nur diejenigen Vorgängertasks inkl. der Nachrichten betrachtet werden müssen, die seit der letzten zeitaktivierten Task vor τ_i in der Taskkette liegen. Diese Einschränkung ist in der nun folgenden Definition der Jitter-Überapproximation berücksichtigt.

Definition 4.4.1 (Überapproximation des Jitters). *Sei $\tau_i \in \mathcal{T}$ mit $tr_i = TRIGGER$. Sei $b_i(\Pi(\tau_i))$ die BCET von τ_i auf dem Prozessorknoten $\Pi(\tau_i)$. Dann ist der jeweilige Jitter einer Task τ_i einer Taskkette $d = (\text{“}\tau_1\tau_2 \ldots \tau_s\tau_{s+1} \ldots \tau_i \ldots \tau_n\text{“}, \lambda)$ mit $tr_s = TIME \wedge \forall l \in \{s+1, \ldots, i-1\} : tr_l = TRIGGER$ gegeben durch:*

$$J_i = \sum_{j=s}^{i-1} \left(\Delta_\tau(\tau_j) - \min_{\forall p \in P}\{b_j(p)\}\right) + \sum_{j=s}^{i-1} \Delta_\gamma(g_{j+1}),$$
$$mit\ g_{j+1} = (\tau_{j+1}, s_{j+1}, a_{j+1}) \in \gamma_j$$

Da aufgrund der Platzierung Kommunikationen zwischen Tasks in sich zusammenfallen können, wenn die beiden Kommunikationspartner-Tasks dem selben Knoten zugewiesen werden, ist die best-case Ausführungszeit für Kommunikationen hier mit 0 angegeben. In Definition 4.4.1 ist dementsprechend kein subtraktiver Faktor bei den Deadlines der Kommunikationen $\Delta_\gamma(g_{j+1})$ angegeben.

In der gleichen Art und Weise läßt sich auch der Jitter von Nachrichten einer Taskkette überapproximieren. In der Definition 4.4.1 wird übrigens keine Rücksicht auf die Tatsache genommen, dass Zeit-basiert verschickte Nachrichten keinen Jitter hervorrufen bzw. bisher bestehenden Jitter “schlucken”, wie dies auch die sehr wohl berücksichtigten Zeit-basierten Tasks tun. Dies liegt an der fehlenden Spezifikation der Art des Nachrichtenversands: Da Nachrichten prinzipiell über mehrere, auch unterschiedlichen Paradigmen gehorchenden, Bussystemen verschickt werden, ist zum Zeitpunkt der Jitterapproximation nicht festzustellen, ob eine Nachricht Zeit- oder Ereignis-basiert versendet wird (Die Jitterapproximation wird vor der eigentlichen Platzierung erstellt.). Würden die Nachrichten streng als Zeit-basiert charakterisiert werden, so könnte dies ohne Probleme analog zum Vorgehen bei Tasks geschehen, das Platzierungsverfahren dürfte aber dann auch nur entsprechende Kommunikationsmedien zum Versand auswählen.

Da die Definiton des Jitters in 4.4.1 aber eine Überapproximation verwendet, können Schedulinganalysen ein zur Wirklichkeit differierendes Abbild des Echtzeitverhaltens erzeugen. Diese Differenz soll im folgenden für den schlimmsten anzunehmenden Fall quantifiziert werden.
Betrachtet man den Einfluss des Jitters in einer vereinfachten Form der Schedulinganalyse aus Gleichung 4.3, so zeigt sich, dass eine Erhöhung des Jitters zusätzliche Preemptionen durch höher priorisierte Tasks vorhersagt, die in der realen Implementierung nicht auftreten würden. Geht man beispielsweise von 2 Tasks τ_i und τ_j mit

$\phi_i < \phi_j$ aus, so ergibt sich abseits von Blockierungen und ähnlichem das folgende Echtzeitverhalten:

$$r_i = c_i + \left\lceil \frac{r_i + J_j}{t_j} \right\rceil \cdot c_j$$

Überapproximation des Jitters bedeutet, dass ein Fehlerfaktor von ζ eingeführt wird, der sich aus der Differenz der realen Antwortzeit (beispielsweise der τ_j initiierenden Nachricht) und der (synthetischen) Deadline der Nachricht g_j von τ_j nach τ_i zusammensetzt, also gilt

$$\zeta = \Delta_\gamma(g_j) - \omega(g_j), \text{ mit } \exists k : g_j = (\tau_j, s_j) \in \gamma_k$$

Damit wird der reale Jitter J_j^* inkrementiert, so dass gilt $J_j = J_j^* + \zeta$. Betrachtet man nur die Anzahl der Unterbrechungen einer Task τ_i durch einen mit Jitter behafteten Task τ_j, so ergibt sich die Anzahl durch:

$$\left\lceil \frac{r_i + J_j}{t_j} \right\rceil = \left\lceil \frac{r_i + J_j^*}{t_j} + \frac{\zeta}{t_j} \right\rceil$$

Die Anzahl der durch den Fehlerfaktor ζ eingeführten Unterbrechungen einer Task τ_i durch τ_j hängt folglich direkt mit dem Verhältnis der Deadline zur Periode der unterbrechenden Task zusammen, wenn davon ausgegangen wird, dass kausal über Nachrichten abhängende Tasks mit derselben Periode ausgestattet sind[1]. Im allgemeinen Fall kann die Deadline $\Delta_\gamma(g_j)$ bis zu einem Vielfachen der Periode t_j betragen, so dass gilt

$$\Delta_\gamma(g_j) = n \cdot t_j - \epsilon$$

mit einem beliebigen ϵ kleiner als $n \cdot t_j$. Im schlimmsten anzunehmenden Fall gilt $\epsilon = 0$. D.h. die Anzahl der Unterbrechungen durch τ_j ergeben sich zu

$$\left\lceil \frac{r_i + J_j^*}{t_j} + \frac{\zeta}{t_j} \right\rceil = \left\lceil \frac{r_i + J_j^*}{t_j} + \frac{n \cdot t_j - \omega(g_j)}{t_j} \right\rceil$$

Geht man ferner davon aus, dass die Antwortzeit der Task genügend klein ist, wird der maximal zu erreichende Fehler durch das Setzen von $\omega(g_j) = 0$ angenähert. Damit ergibt sich für die Anzahl an Unterbrechungen unter Berücksichtigung der Definition der Ceiling-Funktion:

$$\left\lceil \frac{r_i + J_j^*}{t_j} + \frac{n \cdot t_j}{t_j} \right\rceil = \left\lceil \frac{r_i + J_j^*}{t_j} \right\rceil + n$$

Es folgt also, dass bei einer maximalen Deadline einer Task vom n-fachen seiner Periode im schlimmsten Fall gerade n zusätzliche Preemptionen durch den Fehler berücksichtigt werden, die im realen System nicht auftreten. Bei der in dieser Arbeit vorausgesetzten Einschränkung auf $\Delta_\tau(\tau_i) \leq t_i$ kann somit maximal eine Preemption pro höher priorisiertem Task fälschlicherweise angenommen werden. Dabei ist

[1] Eine Annahme, die in der Tat die Grundlage der Methodiken dieser Arbeit bildet.

allerdings zu berücksichtigen, dass der Fehler nicht unbedingt tatsächlich zu einer zusätzlichen Preemption führt, sondern dieses überapproximierte Verhalten nur dann induziert wird, wenn die Summe aus Antwortzeit, realem Jitter und Fehler ζ geteilt durch die Periode gerade eine ganzzahlige Grenze überschreitet, die ohne ζ nicht überschritten worden wäre. Zudem ist die Approximation von $\omega(g_j) = 0$ gerade in Systemen mit Deadlines kleiner oder gleich der Periode nicht sehr wahrscheinlich, so dass eine tatsächliche Preemption nur in seltenen Fällen vorhergesagt wird.
Abgesichert wird diese Einschätzung durch Messungen an einem Satz von synthetischen Benchmarks mit unterschiedlichen Eigenschaften, die jeweils typische Systemkriterien abdecken. Ausgehend von einem Platzierungsverfahren, wie es in Kapitel 5 vorgestellt wird, wurde unter Verwendung der Überapproximation des Jitters eine Verteilung von Tasksystemen mit jeweils 20-25 Tasks auf eine aus 3 Prozessorknoten und einem Tokenring bestehenden Architektur vorgenommen. Anschließend ist das Ergebnis dieser Platzierung um dynamische Jitter-Anteile erweitert worden, und in einer Nachberechnung wurden die zuviel vorhergesagten Preemptionen und deren Auswirkungen auf die Antwortzeit der jeweiligen Tasks quantifiziert. Die Parameter bei der Erzeugung der Benchmarks wurden so gewählt, dass typische Systemlast-Situationen auf den jeweiligen Knoten abgebildet werden können. Bezüglich des Jittereinflusses sind dies die Ähnlichkeit der Ausführungszeiten und Perioden und die Länge der jeweiligen Taskketten. Die folgenden charakteristischen Taskeigenschaften sind jeweils verteilt und evaluiert worden:

KN Kurze Taskketten mit der maximalen Länge von 3 Tasks. Ausführungszeiten und Perioden genügen einer gleichmäßigen Verteilung über einem gegebenen Intervall.

MN Taskketten mit einer mittleren Länge, die maximal 5 beträgt. Ausführungszeiten und Perioden sind wiederum gleichmäßig verteilt.

MD Taskketten mit einer mittleren Länge, die maximal 5 beträgt. Ausführungszeiten und Perioden der Tasks liegen alle in einem sehr engen Intervall und ähneln sich somit.

LN Lange Taskketten mit der maximalen Länge von 7 Tasks. Ausführungszeiten und Perioden sind gleichmäßig verteilt.

Alle Tasksysteme wurden mit einer Deadline ausgestattet, die kleiner oder gleich ihrer jeweiligen Periode ist. Abbildung 4.21 zeigt einen Überblick der Evaluationen dieser 4 Szenarien.

Auffällig ist die deutlich höhere Anzahl der überapproximierten Preemptionen in den Tasksystemen mit kleinen Taskketten. Dies liegt im wesentlichen an der Verteilung der jeweiligen Tasks einer Taskkette auf die 3 Knoten der Architektur: Das Optimierungsverfahren hat eine bzgl. der Kommunikationslast minimierte Platzierung errechnet, die versucht, Tasks einer Taskkette zusammen auf einen Knoten zu allokieren. Das erniedrigt zum einen die Anzahl der über das Kommunikationsmedien zu sendenden Nachrichten und führt auf der anderen Seite zu weniger Preemptionen auf dem Knoten. Letzteres lässt sich aus den im folgenden Kapitel 4.5.1 aufgestellten

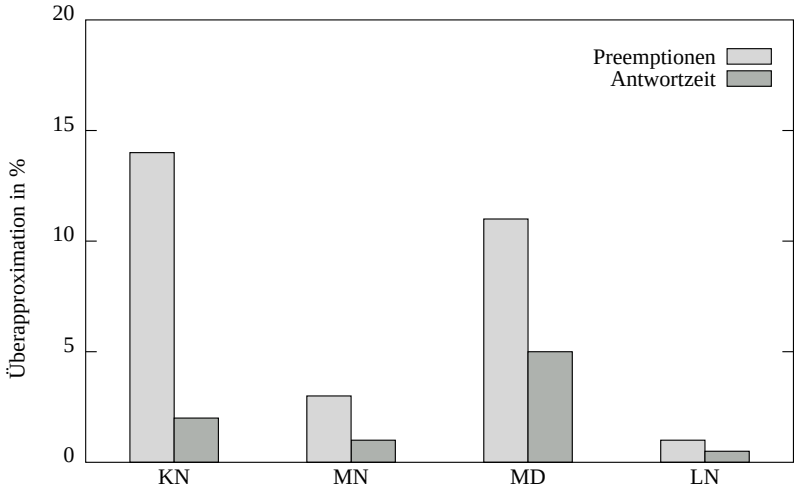


Abbildung 4.21: Prozentuale Überapproximation gemessen an typischen Tasksystemen bzgl. Antwortzeit und Anzahl der Preemptionen im System unter der gegebenen Platzierung.

sogenannten preemptionsfreien Taskfenstern ableiten. Vorweggenommen bedeutet dies, dass Tasks der gleichen Taskkette sich nur eingeschränkt gegenseitig unterbrechen können. Dies hat zur Folge, dass bei langen Taskketten die Prozessorknoten nur durch wenige sich unterbrechende Tasks belegt sind. Bei kurzen Taskketten hingegen wird sich eine Vielzahl potentiell unterbrechender Tasks auf einem Knoten behindern, so dass in diesen Fällen auch die Anzahl der überapproximiert vorhergesagten Preemptionen größer ist.
Generell ist die Auswirkung auf die systemweit zusammengefasste Antwortzeit selbst im Falle einer hohen Überapproximation mit unter 5% eher gering. Hier hängt der Wert entscheidend von dem Verhältnis der Ausführungszeiten der unterbrechenden und unterbrochenen Tasks ab[1]. Im Szenario MD wird dies deutlich: Die Ausführungszeiten der Tasks liegen sehr nah beieinander, so dass eine zusätzliche Preemption einen prozentual höheren Vorhersagefehler induziert als dies in Systemen mit stark differierenden Ausführungszeiten erfolgt.
Zusammenfassend lässt sich für eine gemittelte Systemlast (die eher einem realen System entspricht als die isoliert analysierten Szenarien) sagen, dass mit etwa 5% zuviel vorhergesagten Preemptionen zu rechnen ist, die sich allerdings im Mittel nur um etwa 1,5% zu Ungunsten der Antwortzeiten der Tasks auswirken. Letzteres ist ein gemittelter Wert über alle Antwortzeiten eines Taskystems. Einzelne Antwortzeiten wurden in den Evaluationen mit einer maximalen Abweichung von 8% gemessen. Dies ist allerdings sehr stark abhängig vom diskret behandelten Tasksystem und kann in Einzelfällen deutlich höher liegen. Insgesamt ist jedoch festzustellen, dass die durch die Festlegung des Jitters erzeugte Überapproximaton der Antwortzeiten nur eine sehr gering schlechtere Vorhersage liefert. Dies ist in Systemen, in denen die Deadlines der Tasks und Nachrichten kleiner oder gleich der Periode sind, darauf

[1] Auswirkungen auf die Antwortzeit von Nachrichten sind im hier betrachteten einfachen Zeitbasierten Tokenring-Bussystem nicht möglich. Gleichwohl sind die Nachrichten mit dynamischen Jitter ausgestattet, der sich durch die Übertragung zusätzlich zum von der Nachricht erzeugten Jitter auf die nachfolgenden Tasks einer Taskkette überträgt.

zurückzuführen, dass nur jeweils maximal eine zusätzliche Preemption pro unterbrechenden Task durch den Fehler induziert wird, unabhängig von der Länge der Antwortzeit oder Periode aller Tasks. Gleichzeitig ist die Scheduling-Analyse durch die Festlegung des Jitters aber wieder berechenbar und ermöglicht eine effiziente Platzierungsbewertung.

4.4.2 Dynamische sub-lokale Deadline- und Jittergenerierung für Kommunikationen

Kommunikationen können im Gegensatz zu Tasks nicht nur ein Medium nutzen, sondern aufgrund eines Pfades zwischen zwei Knoten auch durchaus mehrere, die in sequentieller Reihenfolge die Übertragung der Nachricht abwickeln. Bei der Betrachtung eines solchen Szenarios, siehe z.B. Abbildung 4.22, wird deutlich, dass die synthetisierte lokale Deadline $\Delta_\gamma(g_j)$ in diesem Zusammenhang wiederum als End-to-End-Deadline angesehen werden kann, da sie sich über verschiedene Übertragungs-Instanzen auf einem Pfad erstreckt.

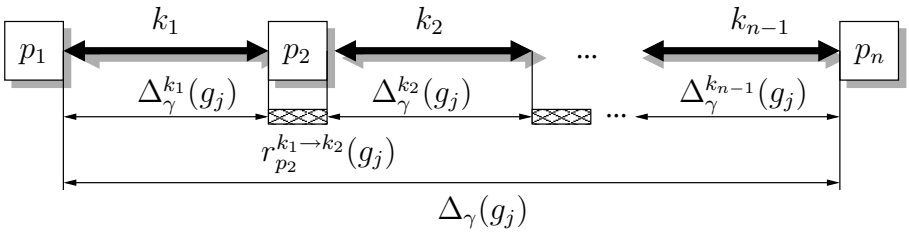


Abbildung 4.22: Aufteilung einer synthetisierten lokalen Deadline $\Delta_\gamma(g_j)$ einer Kommunikation g_j in sub-lokale Deadlines für jedes benutzte Kommunikationsmedium.

Eine Verteilung der lokalen Deadline auf sogenannte sub-lokale Deadlines für Kommunikationen ist die notwendige Konsequenz. Bedingt wird dies wiederum durch die Notwendigkeit, eindeutige Komponentenschnittstellen zu bilden (siehe Kapitel 4.2 und 4.4.1). Würde an dieser Stelle auf eine Synthese der sub-lokalen Deadlines verzichtet werden, wäre wiederum, wie im vorigen Abschnitt dargestellt, eine holistische Analyse des Gesamtsystems erforderlich und der inkrementelle Charakter der Platzierung ginge verloren.
Wenig sinnvoll ist in diesem Zusammenhang eine a priori Synthese der sub-lokalen Deadlines in Anlehnung an das Zusammenspiel von Task- und Kommunikationsdeadline, wie sie in Gleichung 4.4 dargestellt ist. Aufgrund potentiell mehrfacher zusammenfallender Teilkommunikationen ist die Zuordnung des damit freiwerdenden Zeitraumes auf die verbleibenen sub-lokalen Deadlines im allgemeinen nicht optimal berechenbar. Eine Zuteilung nur auf den Kommunikationsvorgänger ist insofern problematisch, als in einer freien Verteilung der Sender- und Empfängertasks im Vorherein nicht bestimmt werden kann, welches Kommunikationsmedium in einer sequentiellen Abfolge zur Übertragung der Nachricht g_j vor einem anderen liegt. Zudem bedingen die unterschiedlichen Eigenschaften der Kommunikationsmedien (beispielsweise die sehr unterschiedliche Flexibilität bzgl. kleiner Deadlines von

zeitbasierten oder ereignisbasierten Systemen) eine differenzierte Bestimmung unter Kenntnis der aktuellen Verteilung.
Stattdessen sollen die sub-lokalen Deadlines hier Teil des Optimierungsprozesses bei der Findung einer möglichst optimalen Platzierung sein. Dabei ist zu beachten, dass der Übergang von einem Kommunikationsmedium k_i auf ein nachfolgendes Medium k_{i+1} über einen Gateway-Knoten p_G erfolgt und dieser ebenfalls eine Verzögerung der Übertragungslatenz von $r_{p_G}^{k_i \to k_{i+1}}(g_j)$ erzeugt. Dies ist bei der Wahl der sub-lokalen Deadlines für die betroffenen Kommunikationen $g_j \in \gamma_i$ zu berücksichtigen, so dass sich insgesamt die folgende Bedingung für einen Pfad $\Gamma(\tau_i, g_j) =$ "$k_1 k_2 \dots k_n$" ergibt:

$$\Delta_\gamma(g_j) = \sum_{\forall k \in \{k_1,\dots,k_n\}} \Delta_\gamma^k(g_j) + \sum_{\forall k_l \in \{k_1,\dots,k_{n-1}\}} r_{k_l \cap k_{l+1}}^{k_l \to k_{l+1}}(g_j) \tag{4.35}$$

Gleichung 4.35 erfordert das Vorhandensein lediglich eines einzigen Gateway-Knotens $p_G = k_l \cap k_{l+1}$ zwischen zwei Kommunikationsmedien. Dies ist durch eine entsprechend formulierte Wohlgeformtheitseigenschaft für die zu betrachtenden Architekturen sicher zu stellen. Gemäß den Ausführungen in Kapitel 4.3 können die Annahme und das Versenden von Kommunikationen durch eine knotenspezifischen Grundlast (Tasks τ_{IN} und τ_{OUT}) modelliert werden, die jeweils mit einer sehr hohen Priorität und einer konstanten Deadline $\Delta_{IN}(p_G)$ bzw. $\Delta_{OUT}(p_G)$ für alle möglichen Nachrichtengrößen ausgestattet sind. Damit lässt sich die Latenzzeit auf den Gateway-Knoten in Gleichung 4.35 verkürzt darstellen:

$$r_{k_l \cap k_{l+1}}^{k_l \to k_{l+1}}(g_j) = \Delta_{IN}(k_l \cap k_{l+1}) + \Delta_{OUT}(k_l \cap k_{l+1}) \tag{4.36}$$

Die eigentliche Determination der jeweiligen sub-lokalen Deadlines bleibt hierbei vollständig dem jeweiligen Platzierungs- und Optimierungsverfahren überlassen. Aufgrund der unterschiedlichen Vorgehensweise von verschiedenartigen Optimierungsverfahren, wie beispielsweise heuristische/stochastische und optimale Verfahren, sind grundsätzlich unterschiedliche Ansätze zur Bestimmung der sub-lokalen Deadlines anzuwenden. Dies wird in Kapitel 5 detailliert dargestellt.
Jede Kommunikation $g_j \in \gamma_i$ ist mit einem aus dem Pfad der vorangegangenen Tasks und Kommunikationen stammenden Jitter J_{g_j} behaftet, der gemäß Definition 4.4.1 generiert wurde. Bei der sequentiellen Verteilung der Nachrichtenübertragung auf eine Menge von Kommunikationsmedien entsteht folglich auch auf diesen Medien $k \in K$ ein jeweils sub-lokaler Jitter $J_{g_j}^k$, der sich aus dem eingehenden Jitter der Nachricht J_{g_j} und dem Zeitverhalten der vorgeschalteten Kommunikationsmedien ergibt. Analog zu Definition 4.4.1 wird auch der sub-lokale Jitter hier durch eine Überapproximation aus den Deadlines der vorangegangenen Übertragungen gebildet. Für eine Nachricht g_j von τ_i nach τ_j mit der Platzierungsabbildung für Kommunikationen $\Gamma(\tau_i, g_j) =$ "$k_1 k_2 \dots k_n$" und der Best-Case-Übertragungszeit $\beta_k(g_j)$ für g_j auf dem Kommunikationsmedium k gilt somit:

$$\forall l \in \{1,\dots,n\} : J_{g_j}^{k_l} = \begin{cases} J_{g_j} & \text{wenn } k_l = k_1 \\ J_{g_j} + \sum_{s=1}^{l-1} \left(\Delta_\gamma^{k_s}(g_j) - \beta_{k_s}(g_j)\right) & \text{sonst} \end{cases} \tag{4.37}$$

Hier wird anders als in Definition 4.4.1 die Best-Case-Übertragungszeit genutzt, da garantiert eine Kommunikation auf dem Medium stattfindet. Der Wert $\beta_k(g_j)$ ist

einfach statisch für jeden Bus und jede Nachricht anhand der jeweiligen Charakteristika, wie Paketgröße und Busgeschwindigkeit, ermittelbar und hilft, die Überapproximation des sub-lokalen Jitters zu minimieren. Gleichwohl ist Wert für $J^k_{g_j}$ selbst kein statischer Wert, der vor der Platzierung ermittelt werden kann, da — wie oben beschrieben — die jeweiligen sub-lokalen Deadlines dynamisch vom Optimierungsverfahren bestimmt werden. Trotzdem ist eine Überapproximation notwendig, um im inkrementellen Entwurfsprozess dem Kommunikationsmedium entsprechende eindeutige Schnittstellen zur Verfügung zu stellen.

Im Sinne eines inkrementellen Entwurfes stellt die Verlagerung der sub-lokalen Deadlines in den Bereich der Optimierungsziele keine Einschränkung dar. Für die Komponente "Kommunikation g_j" besteht aus Prozesssicht weiterhin nur die lokale Deadline $\Delta_\gamma(g_j)$ und der dazugehörige Jitter. Die sub-lokalen Deadlines und Jitter für die Kommunikationsmedien auf dem Pfad durch die Topologie der Architektur werden hingegen für jedes Medium im entsprechenden Repository der aktuellen Systemkonfiguration vorgehalten und stehen damit weiteren inkrementellen Integrationsschritten zur Verfügung, sind aber nicht als Eigenschaft der Komponente "Kommunikation g_j" selbst sichtbar.

4.5 Preemptionen innerhalb von Taskketten

Preemptionen einer Task τ_i durch andere Tasks, wie sie in der Antwortzeitberechung in Gleichung 4.3 mittels des Interferenzfaktors berücksichtigt werden, hängen von der Priorität der jeweiligen Tasks ab. Werden die Prioritäten z.B. nach dem Deadline-monotonen Ansatz vergeben, führt das aufgrund der synthetisierten Deadlines $\Delta_\tau(\tau_j)$ zu unterschiedlichen Prioritäten auch von Tasks, die über eine Taskkette kausal aneinander gekoppelt sind und deren Zeitbedingung mittels einer End-to-End-Deadline angegeben ist. Eine von dem kausalen Zusammenhang unabhängige Berechnung des Interferenzfaktors, wie sie in Gleichung 4.3 durchgeführt wird, ist demzufolge eine Überapproximation des realen Verhaltens: Invokationen des dem Task τ_i direkt nachfolgenden Tasks können τ_i (bei entsprechender Prioritätenbeziehung) nur dann unterbrechen, wenn es sich dabei um eine ältere oder jüngere Invokation der gesamten Taskkette handelt. Zur Begrenzung dieser Überapproximation des Preemptionsverhaltens wird im folgenden ein sogenanntes preemptionsfreies Taskfenster eingeführt. Dieses Fenster enthält alle Tasks τ_j einer Taskkette, zu der eine Task τ_i gehört, die aufgrund der zeitlichen Randbedingungen und der jeweiligen Abhängigkeiten keinen preemptiven Beitrag zur Antwortzeit von τ_i leisten kann. Preemptionsfreie Fenster für eine Task werden gebildet, indem jeweils der Vor- und den Nachbereich von Invokationen (also frühere und spätere Invokationen der Taskkette) einer Task bzgl. des Zeitverhaltens analysiert werden. In Abbildung 4.23 ist dies beispielhaft dargestellt.

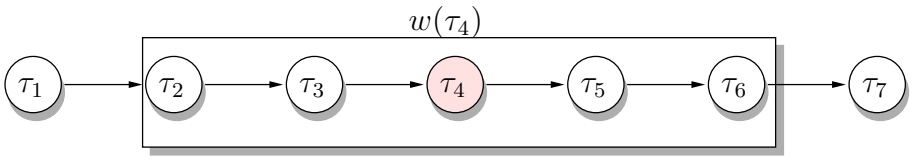


Abbildung 4.23: Taskkette mit dem preemptionsfreien Taskfenster w für den Vorbereich und Nachbereich einer Task τ_4. Alle Tasks innerhalb des gekennzeichneten Fensters können τ_4 nicht unterbrechen, wohingegen τ_1 und τ_7 Preemptionen verursachen.

4.5.1 Disjunkte Taskketten

Zunächst soll der Fall betrachtet werden, dass alle Taskketten eines Tasksystems vollständig disjunkt sind, d.h. kein Task ist gleichzeitig Teil mehrerer Taskketten.
Seien im folgenden aufeinanderfolgende Invokationen $l-n$ bis $l+n$ mit $n \in \mathbb{N}^+$ von Tasks τ_i einer Taskkette durch τ_i^{l-n} bzw. τ_i^{l+n} gekennzeichnet.
Im Vorbereich $w_V(\tau_i)$ der Task τ_i^l werden alle Tasks τ_j^{l-n} der Taskkette *nicht* aufgenommen, die in früheren Invokationen die aktuelle Ausführung von τ_i^l überlappen und damit Preemptionen hervorrufen. Diese Situation tritt ein, wenn der Abstand, der durch die Periode der Taskkette gegeben ist, nicht ausreicht, um eine Überlappung zu verhindern. Abbildung 4.24 macht dies an einem Beispiel deutlich.

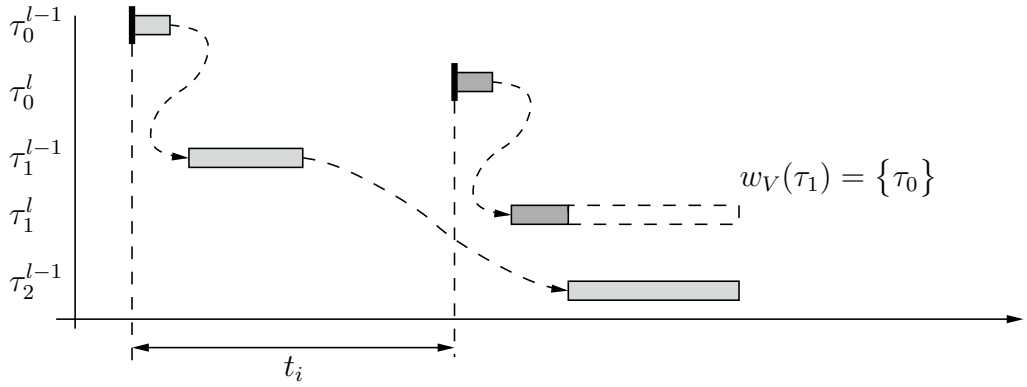


Abbildung 4.24: Preemptionen durch Tasks des Vorbereichs einer hier betrachteten Task τ_1. Task τ_2 kann aufgrund des Zusammenspiels der zeitlichen Bedingungen in einer früheren Invokation preemptiv in den Ablauf von Task τ_1 eingreifen, Task τ_0 jedoch nicht. Letztere wird demzufolge in das preemptionsfreie Taskfenster $w_V(\tau_1)$ für den Vorbereich aufgenommen.

Analysen zur Determinierung der preemptionsfreien Taskfenster des Vorbereiches von Tasks einer Taskkette müssen alle potentiell möglichen Kombinbationen vom Eintreffen der Startereignisse der beiden betrachteten Tasks berücksichtigen. Dies bedeutet insbesondere, dass im Prinzip unendlich viele frühere Invokationen der Taskkette auf ihren preemptiven Einfluss überprüft werden müssen[1]. Formal ist die

[1] In der Praxis ergibt sich eine obere Grenze, die durch die gegebene End-to-End-Deadline der Taskkette und der Periode bestimmt wird.

Menge der möglichen Zeitpunkte für das Eintreffen der Startereignisse von Tasks in einem Intervall anzugeben, das durch die frühest mögliche und die spätest mögliche Invokation einer Task beschränkt ist (Startintervall). Soll für eine Task τ_i^l ein preemptionsfreies Taskfenster erzeugt werden, so wird zunächst für jede Task τ_j der Taskkette die Vereinigung aller Startintervalle für alle möglichen früheren Invokationen der Taskkette gebildet:

$$E_{\tau_j}^{-\infty} = \bigcup_{r=l-\infty}^{l-1} \left[r \cdot t_j + \sum_{s=0}^{j-1} \min_{p \in P}\{b_s(p)\}, r \cdot t_j + \sum_{s=0}^{j-1} \big(\Delta_\tau(\tau_s) + \Delta_\gamma(g_{s+1})\big) + \sigma_j \right]$$

Zeit-basierte Tasks innerhalb der Taskkette erzeugen zusätzliche Verschiebungen der spätest möglichen Startzeit. Dies ist durch den σ_j-Faktor analog zum Vorgehen in Abschnitt 4.2 abgedeckt, der das infinitisimale Verpassen der Invokation von zeitbasierten Tasks auf dem Pfad zur Task τ_j aufsummiert:

$$\sigma_j = \sum_{s \in \{1,\ldots,j\}: tr_s = TIME} t_s$$

Das Startintervall für die betrachtete Task in der l-ten Invokation τ_i^l wird ähnlich gebildet. Da Preemptionen auch während der Laufzeit von τ_i^l auftreten können, ist das unterbrechungsfähige Intervall entsprechend um die Antwortzeit der Task zu erweitern. Eine Überapproximation dieser Antwortzeit um die synthetisierte Deadline $\Delta_\tau(\tau_i)$ wird hier zur Verhinderung einer dynamisch von der Platzierung abhängigen Bestimmung des Intervalls genutzt. Damit ergibt sich das Unterbrechungsintervall für den Vorbereich zu

$$E_{\tau_i}^{V} = \left[l \cdot t_i + \sum_{s=0}^{i-1} \min_{p \in P}\{b_s(p)\}, l \cdot t_i + \sum_{s=0}^{i-1} \big(\Delta_\tau(\tau_s) + \Delta_\gamma(g_{s+1})\big) + \Delta_\tau(\tau_i) + \sigma_i \right]$$

Mithilfe dieser Intervallmengen kann nun das preemptionsfreie Taskfenster für den Vorbereich w_V erzeugt werden:

$$w_V(\tau_i) = \left\{ \tau_j \;\middle|\; E_{\tau_j}^{-\infty} \cap E_{\tau_i}^{V} = \emptyset \right\}$$

Entsprechend der Analyse des Vorbereiches kann der Nachbereich einer Task τ_i aufgebaut werden. Entscheidend sind hierbei die Überlappungen von Tasks aus späteren Invokationen der Taskkette. Abbildung 4.25 macht dies an einem Beispiel deutlich.

Formal bedeutet dies auf der einen Seite alle möglichen Startzeitpunkte von späteren Invokationen $(l+1), \ldots, l+\infty$ und auf der anderen Seite das Intervall der Startzeitpunkte normiert auf die Invokation $l = 0$.

$$E_{\tau_j}^{+\infty} = \bigcup_{r=1}^{\infty} \left[r \cdot t_j + \sum_{s=0}^{j-1} \min_{p \in P}\{b_s(p)\} \quad , \quad r \cdot t_j + \sum_{s=0}^{j-1} \big(\Delta_\tau(\tau_s) + \Delta_\gamma(g_{s+1})\big) + \sigma_j \right]$$

$$E_{\tau_i}^{N} = \left[\sum_{s=0}^{i-1} \min_{p \in P}\{b_s(p)\}, \sum_{s=0}^{i-1} \big(\Delta_\tau(\tau_s) + \Delta_\gamma(g_{s+1})\big) + \Delta_\tau(\tau_i) + \sigma_i \right]$$

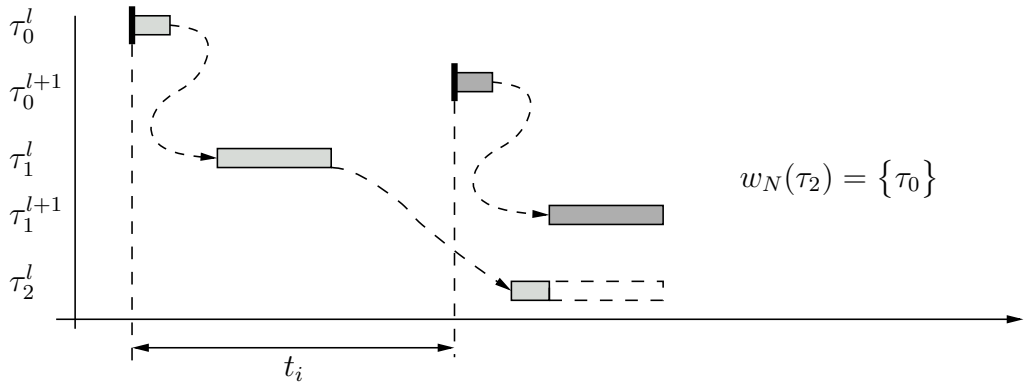


Abbildung 4.25: Preemptionen durch Tasks des Nachbereichs einer betrachteten Task τ_2. Task τ_1 kann aufgrund des Zusammenspiels der zeitlichen Bedingungen in einer späteren Invokation preemptiv in den Ablauf von Task τ_2 eingreifen, Task τ_0 jedoch nicht. Letztere wird demzufolge in das preemptionsfreie Taskfenster $w_N(\tau_2)$ für den Nachbereich aufgenommen.

Mithilfe dieser Intervallmengen kann nun das preemptionsfreie Taskfenster für den Nachbereich w_N erzeugt werden:

$$w_N(\tau_i) = \left\{\tau_j \;\; | E_{\tau_j}^{+\infty} \cap E_{\tau_i}^{N} = \emptyset\right\}$$

Preemptionen von τ_i^l durch Tasks τ_j^l derselben Invokation der Taskkette sind aufgrund der kausalen Abhängigkeiten nur für Tasks beobachtbar, die zeit-basiert angestoßen werden. Dies kann nicht auf die gleiche Art und Weise analysiert werden, wie Vor- und Nachbereiche, da die Offsets der Startereignisse des Starttasks der Taskkette und des zeit-basierten Tasks nicht bekannt sind und somit keine gemeinsame Relation etwaiger Startintervalle zu finden sind. Stattdessen muss als obere Abschätzung davon ausgegangen werden, dass zeit-basierte Tasks grundsätzlich Preemptionen verursachen.

Letztlich können die preemptionsfreien Taskfenster bzgl. einer Tasks τ_i aus einer Taskkette $d_r = (\text{“}\tau_0\tau_1 \ldots \tau_n\text{”}, \lambda)$ gebildet werden, indem für alle Tasks die Schnittmenge der oben entwickelten Teilfenster verwendet wird, d.h. diejenigen Tasks aus der Menge genommen werden, die im Vor- oder im Nachbereich Preemptionen verursachen können oder die zeit-basiert gestartet werden:

$$\forall l \in \{0, \ldots, n\} : w^{d_r}(\tau_l) = \left(w_V^{d_r}(\tau_l) \cap w_N^{d_r}(\tau_l)\right) \backslash \{\tau_i \;\; | \;\; tr_i = TIME\}$$

Zeit-basierte Tasks τ_i innerhalb einer Taskkette können aufgrund der fehlenden zeitlichen Relation zu den Aufrufen der Vorgängertasks keine preemptionsfreie Taskfenster aufbauen. Demzufolge gilt entsprechend:

$$\forall d_r : d_r = (\text{“}\tau_0\tau_1 \ldots \tau_n\text{”}, \lambda) : tr_i = TIME \Rightarrow w^{d_r}(\tau_i) = \emptyset \tag{4.38}$$

Für den in dieser Arbeit betrachteten Fall $d_i = (\text{“}\tau_0 \ldots \tau_n\text{”}, \lambda)$ mit $\lambda \leq t_i$ gilt

$$\sum_{s=0}^{n-1} \left(\Delta_\tau(\tau_j) + \Delta_\gamma(g_{j+1})\right) + \Delta_\tau(\tau_n) + \sigma \leq t_i$$

Startintervalle für Vor- und Nachbereiche bezeichnen die Zeitpunkte, zu denen die jeweils betrachtete Task ihr Startereignis bekommen kann, ohne dabei die Berechnungszeit der Task zu berücksichtigen. Mit der obigen Formel kann also gefolgert werden, dass für die jeweils rechte Begrenzung der Einzelintervalle in $E_{\tau_j}^{-\infty}$ bzw. $E_{\tau_j}^{+\infty}$ im Fall $\lambda \leq t_i$ gilt:

$$r \cdot t_j \sum_{s=0}^{j-1} \big(\Delta_\tau(\tau_j) + \Delta_\gamma(g_{j+1})\big) + \sigma_j = r \cdot t_j + (t_j - \varepsilon),$$

wobei ε die Differenz zwischen möglicher Startzeit und der End-to-End-Deadline ist, der die Berechnungszeit der Task subsumiert. Gleichermaßen lässt sich über die rechte Intervallgrenze des Unterbrechungsintervalls $E_{\tau_i}^V$ und $E_{\tau_i}^N$ argumentieren, so dass letzlich die folgenden beiden Intervallmengen mit entsprechenden Konsequenzen entstehen.

Vorbereich: Hier gilt es, die Schnittmenge der früheren Invokationen mit dem Unterbrechungsintervall $E_{\tau_i}^V$ zu bilden. Die $E_{\tau_i}^V$ am nächsten kommende Invokation ist die $l-1$-te. Damit ergibt sich der folgende Schnitt (ν_x bezeichnet den Offset der durch die Betrachtung des frühest möglichen Startpunktes entsteht):

$$C_V = [(l-1)t_j + \nu_j, (l-1)t_j + (t_j - \varepsilon_j)] \cap [lt_i + \nu_i, lt_i + t_i]$$

Alle früheren Invokationen der Taskkette müssen nicht betrachtet werden, da die rechten Begrenzungen ihrer Startintervalle aufgrund der obigen Gleichung immer kleiner als die hier betrachtete Invokation sind. Aus Gründen der Feasibility des Gesamtsystems müssen ν_x und ε_j kleiner als die jeweilige Periode sein. Dann kann man folgern, dass gilt

$$t_j \leq t_i \Rightarrow C_V = \emptyset,$$

d.h. es gibt keine Überlappung von Tasks im Vorbereich und damit können alle Tasks der Taskette in das preemptionsfreie Taskfenster des Vorbereiches aufgenommen werden.

Nachbereich Vollkommen analog zum Vorbereich wird bei der Betrachtung des Nachbereiches die folgende Gleichung entstehen:

$$C_N = [t_j + \nu_j, t_j + (t_j - \varepsilon_j)] \cap [\nu_i, t_i]$$

Wiederum kann aus Gründen der Feasibility gefolgert werden, dass gilt:

$$t_j \leq t_i \Rightarrow C_N = \emptyset$$

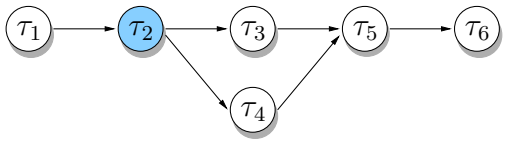
Zusammengefasst enthält das preemptionsfreie Taskfenster für den Fall $\lambda \leq t_i$ die folgenden Tasks der Taskkette:

$$\forall l \in \{0, \ldots, n\} : w_{\lambda \leq t_i}^{d_r}(\tau_l) = \{\tau_0, \ldots \tau_n\} \backslash \{\tau_j \mid t_j > t_l \vee tr_j = TIME\}$$

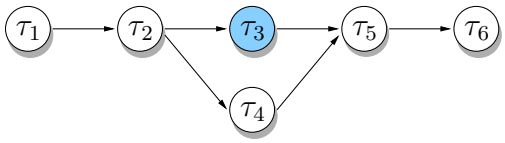
Sind innerhalb einer Taskkette zusätzlich noch keine zeit-basierten Tasks enthalten, so ergibt sich eine vollständige Preemptionsfreiheit bzgl. Tasks der eigenen Taskkette.

4.5.2 Zusammenhängende Taskketten

Stärker verwobene Tasksysteme, in denen einzelne Tasks mehreren Taskketten angehören, müssen gesondert behandelt werden, da für eine Task aus jeder zugehörigen Taskkette eine preemptionsfreies Taskfenster entstehen kann. Zudem ergeben sich möglicherweise unterschiedliche Behandlungsszenarien abhängig davon, wie die Stellung eines Tasks in unterschiedlichen Taskketten ist. Die zu betrachtenden Szenarien sind in den Abbildungen 4.26 und 4.27 dargestellt.

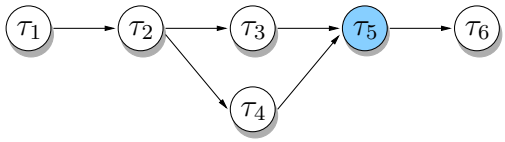


(a) Vor einer Verzweigung in Teilpfade

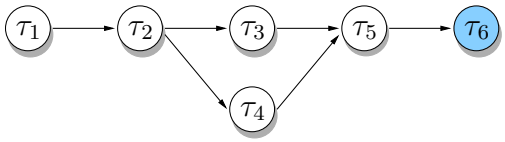


(b) Nach einer Verzweigung, aber vor einer Vereinigung

Abbildung 4.26: Stellung einer Task in mehreren Taskketten vor einem Zusammenlaufen der Ketten auf einen gemeinsamen Teil



(a) Vereinigungstask von Teilpfaden



(b) Nach einer Vereinigung

Abbildung 4.27: Stellung einer Task in mehreren Taskketten nach einem Zusammenlaufen der Ketten auf einen gemeinsamen Teil

Sowohl in Abbildung 4.26(a) als auch in 4.27(b) können Tasks in den jeweiligen preemptionsfreien Taskfenstern der Taskketten liegen, die in allen anderen preemptionsfreien Fenstern der betrachteten Task nicht liegen, da sie nicht Teil der Pfade sind.
Trotzdem ist dann sicherzustellen, dass diese Tasks mit in die preemptionsfreie Taskmenge über allen Pfaden aufgenommen wird. Zu diesem Zweck wird das im folgenden beschriebene Vorgehen verwendet:
Sei D_{τ_i} die Menge der Taskketten, in denen die Task τ_i enthalten ist, wobei für alle $d_r \in D_{\tau_i}$ gilt : $d_r = \text{“}\tau_{r_0}\tau_{r_1}\ldots\tau_i\tau_{r_{i+1}}\ldots\tau_{r_n}\text{”}$. Basierend auf den Analyseschritten im Abschnitt 4.5.1 kann zunächst für jede Taskkette $d_r \in D_{\tau_i}$ das preemptionsfreie Taskfenster $w^{d_r}(\tau_i)$ konstruiert werden. Anschließend werden die Taskfenster um alle Tasks angereichert, die sich auf allen anderen Taskketten $d_k \in D_{\tau_i}$ befinden, aber nicht Teil von d_r sind. Als letzter Schritt wird nun der Schnitt über all diese Mengen von Tasks gebildet. Mithilfe dieser Methode wird zum einen erreicht, dass alle Tasks, die in einem preemptionsfreien Taskfenster nur einer Taskkette liegen, aber

ansonsten in keiner Taskkette zu finden sind, in das preemptionsfreie Taskfenster über allen Taskketten $d_r \in D_{\tau_i}$ aufgenommen werden (vgl. z.B. Task τ_4 in Abbildung 4.26(a)). Zum anderen können diejenigen Tasks entfernt werden, die Teil aller Taskketten sind, aber nicht in allen preemptionsfreien Taskfenstern erscheinen. Als Referenz diene hier Task τ_5 in Abbildung 4.26(a). Formal lässt sich dieses Vorgehen folgendermaßen beschreiben:

$$w(\tau_i) = \bigcap_{d_r \in D_{\tau_i}} \tilde{w}^{d_r}(\tau_i) \tag{4.39}$$

$$\text{mit } \forall d_r, \tau_i : \tilde{w}^{d_r}(\tau_i) = w^{d_r}(\tau_i) \bigcup_{d_k \in D_{\tau_i} \setminus \{d_r\}} \{\tau_{k_0}, \ldots, \tau_{k_n}\} \setminus \{\tau_{r_0}, \ldots, \tau_{r_n}\}$$

Betrachtet man die Situation, die in Abbildung 4.26(b) dargestellt ist, unter dieser Definition, wird schnell deutlich, dass diese Vorgehensweise auch in diesem Szenario gilt. Abbildung 4.28 macht dieses an einem komplexeren Beispiel deutlich.

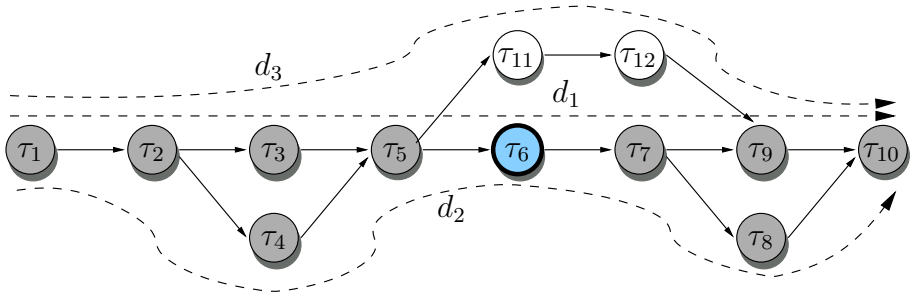


Abbildung 4.28: Stellung einer Task in mehreren Taskketten mit parallelen Anteilen. Die grau markierten Tasks stellen die Menge der Tasks in Taskketten aus D_{τ_6} dar.

Gemäß Gleichung 4.39 werden nur Tasks derjenigen Tasketten aufgenommen, von denen die betrachtete Task τ_6 ein Teil ist. Parallel ablaufende Tasks liegen bzgl. τ_6 in disjunkten Taskketten, d.h. es gilt: $d_3 \notin D_{\tau_6}$. Somit befinden sich τ_{11} und τ_{12} nicht im preemptionsfreien Taskfenster $w(\tau_6)$ und könnten prinzipiell mit τ_6 interferieren.

Schnittpunkte verschiedener Taskketten in Form eines gemeinsamen Tasks, wie es beispielhaft in Abbildung 4.27(a) dargestellt ist, können prinzipiell zwar auch mit dem Algorithmus gemäß Gleichung 4.39 behandelt werden, bedürfen aber im Rahmen der Darstellung dieser Arbeit einer Sonderbehandlung. Das Ansteuern durch unterschiedliche Taskketten bedeutet, dass es zwischen den verschiedenen Startevents eines Schnitttasks (im Abbildung 4.27(a) Task τ_5) keine strenge Beziehung gibt; lediglich ist bekannt, wieviele Aufrufe des Tasks in welchem Zeitraum maximal zu erwarten sind. Tasks dieser Eigenschaft werden üblicherweise *Burst*-Tasks[90] genannt und sind relativ einfach über sogenannte sporadische Server[169] behandelbar, werden im Rahmen dieser Arbeit jedoch nicht weiter betrachtet. Stattdessen wird gefordert, dass alle Tasks, die aufgrund der Mitgliedschaft im Schnittpunkt mehrerer Taskketten ein Bursty-Verhalten zeigen, als *zeit-getriebene* Tasks implementiert sein müssen. Dementsprechend sind an diesen Schnittpunkten asynchrone Kopplungen im Taskgraph erforderlich, die übrigens gleichzeitig auch alle hinter dieser Task

liegende Tasks von ihrem Bursty-Verhalten befreien[1]. Letztlich können diese Tasks damit mit den Methoden aus Abschnitt 4.5.1 (Gleichung 4.38) behandelt werden.

4.5.3 Messung der verbesserten Vorhersagbarkeit

Preemptionsfreie Taskfenster können die vorhergesagte Performanz eines Echtzeitsystems deutlich steigern, hängen aber vom Verteilungsgrad des Systems ab. Letzteres bezeichnet die Freiheit in der Wahl von Prozessorknoten für die einzelnen Tasks. In Systemen mit einem hohen Grad an Verteilungsspielraum wird die Differenz zwischen der zeitlichen Vorhersage mit und ohne preemptive Taskfenster geringer ausfallen, als in einer durch Lokalität geprägten Verteilung: Sind die Tasks einer Kette weit über viele Prozessorknoten gestreut, ist eine vorhergesagte Preemption in der klassischen Analyse eher selten. Die Messungen aus Abbildung 4.29 fundieren diesen Effekt.

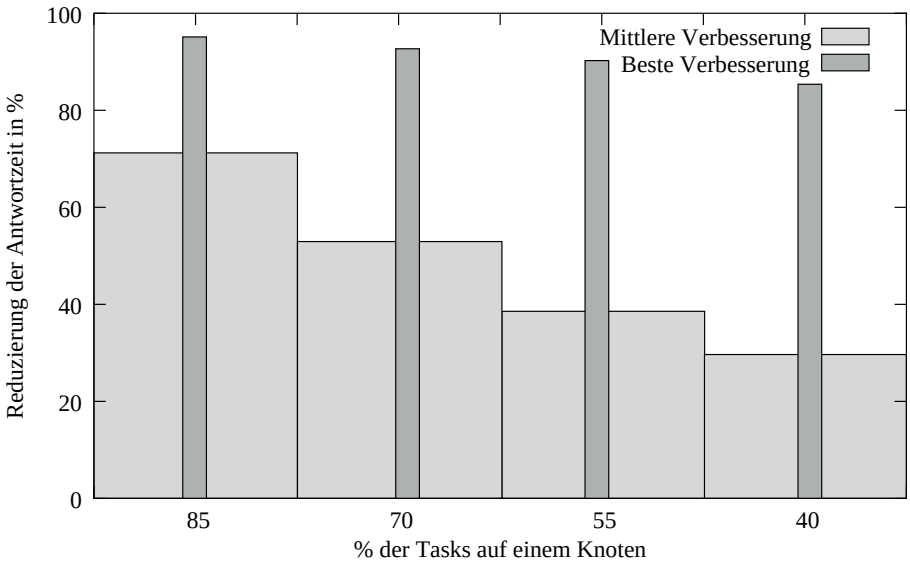


Abbildung 4.29: Mittlere und beste Reduzierung der vorhergesagten Antwortzeit von Tasks in einer Taskkette unter der Annahme unterschiedlicher Verteilungsgrade. Der Verteilungsgrad gibt an, wieviel Tasks der Taskkette auf einem Knoten verbleiben.

Basis der Evaluation sind eine Reihe von Taskketten mit Größen von 10 bis 20 Tasks, deren Prioritäten gemäß der Deadline Monotonic[100] Strategie vergeben sind (bei Gleichheit der Deadlines wird eine zufällige, eindeutige Priorität erzeugt). End-to-End Deadlines sind jeweils kleiner als die Periode der beteiligten Tasks gewählt worden. Der Verteilungsgrad ist für vier Bereiche analysiert worden und sorgt dafür, dass zwischen 15% und 60% der Tasks einer Taskkette auf hier nicht betrachtete Knoten der Architektur verteilt werden. Bei der in Abbildung 4.29 gezeigten Reduzierung der Antwortzeiten wird davon ausgegangen, dass diese abseits verteilten Tasks keine Reduktion der Antwortzeit erfahren.

[1] Somit können alle hinter dem Schnittpunkttask liegenden Tasks auch event-basiertes Verhalten aufweisen.

Es wird deutlich, dass bei einem durch eine hohe Lokalität geprägten System die mittlere Effektivität der Echtzeitvorhersage um bis zu 76% gesteigert werden kann. Einzelne Tasks der Taskkette, hier diejenigen mit einer niedrigen Priorität, leiden in der klassischen Analyse besonders unter zusätzlich, nicht real auftretenden, Preemptionen. In Einzelfällen kann die Reduktion der Antwortzeit bis zu 96% betragen. Entsprechend der Erwartung nimmt die mittlere Differenz in der Vorhersagegenauigkeit mit einer steigenden Anzahl an nicht-lokalen Platzierungen ab, ist aber auch bei nur noch 40% der Tasks auf einem Knoten mit 28% signifikant. Einzelne Tasks können ihre Antwortzeit auch in dieser Konfiguration um bis zu 86% exakter vohersagen.

4.5.4 Übertragung auf Nachrichten-Scheduling

Betrachtet man eine mögliche Übertragung der preemptionsfreien Taskfenster auf das Scheduling von Nachrichten auf den Kommunikationsmedien einer verteilten Architektur, so fällt zunächst die vollkommen analoge Behandlungsweise von Nachrichten und Tasks auf. Dies liegt natürlich daran, dass die Analyse der Nachrichtenverschickung sich ganz maßgeblich auf Scheduling-Analysen abstützen, wie sie aus dem Bereich von Tasksystemen bekannt sind. Folglich sollte die Bildung von preemptionsfreien Taskfenstern für Nachrichten auch vollkommen analog aufgebaut werden können.
Problematisch ist bei diesem Ansatz die Behandlung von Zeit-basierten Kommunikationen. Da Nachrichten, anders als Tasks, im Rahmen der unter Kapitel 4.1 angegebenen Formalisierung nicht eindeutig als Zeit-basiert charakterisiert werden können, ist auch nicht a priori feststellbar, ob sie durch die Platzierung auf bestimmte Medien Ereignis- oder Zeit-basiert verschickt werden. Diese Charakterisierung ist aber zwingend notwendig, um entscheiden zu können, ob ein preemptionsfreies Taskfenster aufgebaut werden muss oder aber ob diese Nachricht generell Preemptionen verursachen kann. Selbst wenn eine Kommunikation als Ereignis-basiert spezifiziert werden könnte, hilft dies auch nicht, da sie durchaus im Rahmen des Routings auf Basis der Topologie der Architektur Zeit-basierte Bussysteme nutzen muss um den Zieltask zu erreichen. Stattdessen muss davon ausgegangen werden, dass jede Nachricht potentiell einen Zeit-basierten Bus nutzen kann und damit generell eine Preemption verursachen kann[1].

[1]Natürlich könnte man aufgrund von Einschränkungen der Platzierung und des Aufbaus der Architektur bzgl. Topologie und Kommunikationsmedien entscheiden, ob eine Nachricht ggf. gar keine Zeit-basierten Medien benutzen kann. In diesem Fall wäre es dann möglich, preemptionsfreie Taskfenster aufzubauen. Diese Optimierung wird jedoch zum jetzigen Zeitpunkt von dem im nächsten Kapitel dargestellten Platzierungsverfahren noch nicht unterstützt.

Kapitel 5

Platzierungswahl

5.1 Komplexität

In den folgenden Abschnitten werden unterschiedliche Herangehensweisen beschrieben, das Platzierungsproblem zu modellieren. Dabei wird von ereignis-basierten, preemptiven Scheduling-Mechanismen ausgegangen. Andere Arten des Scheduling können, wie in Kapitel 4.2 dargestellt, auf einfache Art und Weise integriert werden. Als Blockierungsfaktoren werden aus Gründen der Übersichtlichkeit nur die durch Kommunikationen verursachte Blockierungen betrachtet. Auf weitergehende Verwendungen von beispielsweise Semaphoren für kritische Bereiche wird verzichtet, wobei deren Integration ebenfalls einfach machbar ist.

Zunächst soll jedoch die Komplexität des Problems erörtert werden und anschließend werden die üblicherweise eingesetzten Optimierungsverfahren für derartige Problemklassen diskutiert. Generell ist bekannt, dass das Platzierungs- und Schedulingproblem zur Klasse der NP-harten Probleme gehört[33]. Das damit verbundene exponentielle Wachstum des Lösungsraumes lässt sich einfach erklären: Ginge es nur um die Verteilung von n Tasks auf eine Architektur mit k Prozessorknoten, so ist die Anzahl der möglichen Lösungen mit k^n anzugegeben. Da die betrachteten Systeme heterogene Eingenschaften aufweisen, gibt es keine Symmetrie in den Platzierungen, die sich, wie in der Lehre der Statistik durch die sogenannten Urnenmodelle[23], reduzierend auf die Größe des Lösungsraumes auswirkt; Wird eine identische Taskmenge nicht auf Prozessor p_i, sondern auf p_j platziert, so ist aufgrund der unterschiedlichen Eigenschaften von p_i und p_j die Kostenfunktion nicht mit dem gleichen Wert zu erwarten. Hier kommt zusätzlich zum Tragen, dass nicht nur das Platzierungsproblem der Tasks bestimmt werden muss, sondern ebenso das Routing der Nachrichten über komplexe Topologien oder die Bestimmung der Prioritäten von Tasks auf einem Knoten. Diese Faktoren erzeugen neue Freiheitsgrade der Platzierung und vergrößern damit den Lösungsraum noch einmal signifikant.
Auf der anderen Seite haben aber gerade in den hier betrachteten Anwendungsdomänen, wie der Automobil-Industrie, viele der Tasks vordefinierte Einschränkungen der Platzierungsfreiheit (verbotenen Knoten durch Domain-Zugehörigkeiten, Redundanzen durch Fehlertoleranz, etc.). Dies verkleinert wiederum die Größe des zu durchsuchenden Lösungsraumes. Doch trotz dieser Reduzierung, die jeweils spezi-

fisch zur betrachteten Probleminstanz zu bewerten ist, sind die aufgespannten Lösungsräume realistischer Beispiele, wie das System aus [183], gigantisch groß und bedürfen einer Optimierung mittels effektiver Verfahren.

Im folgenden sollen zunächst die bekanntesten Optimierungsverfahren im Einzelnen beschrieben werden, bevor in Kapitel 5.2 der Stand der Technik analysiert wird. Die Liste der Publikationen der letzten 30 Jahre in den vielen Bereichen der Informatik, Naturwissenschaften oder Ingenieurwissenschaften, die sich mit dem Thema Optimierungsverfahren beschäftigen, umfasst mehrere Hundert. Daher kann die nun folgende kurze Zusammenfassung einzelner Algorithmen nicht vollständig sein. Wir haben stattdessen diejenigen Verfahren herausgegriffen, die am bekanntesten sind und die in ähnlichen Fragestellungen bereits eingesetzt wurden. Andere Verfahren leiten sich zumeist aus den hier dargestellten Grundmethoden ab oder sind problemspezifisch.
Grundsätzlich können die unterschiedlichen Algorithmen in zwei Klassen gruppiert werden, den exakten Verfahren und den heuristischen Verfahren. Abbildung 5.1 macht dies graphisch deutlich.

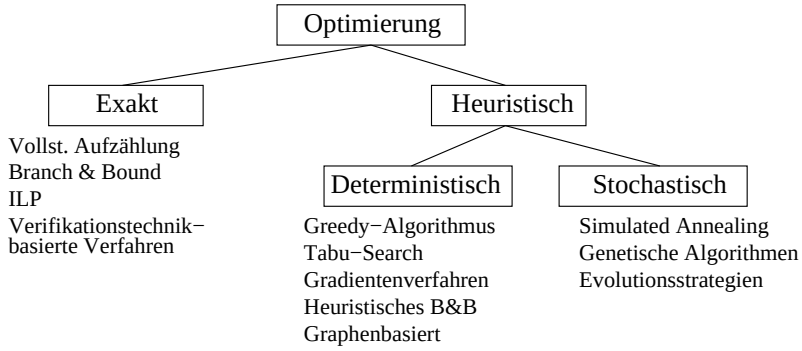


Abbildung 5.1: Klassifizierung der generellen Optimierungsverfahren, ausgenommen problem-spezifische Algorithmen.

Die Differenzierung in exakt und nicht exakt bzw. heuristisch kennzeichnet die Qualität der Lösung der jeweiligen Optimierungsmethodik: Exakte Verfahren liefern immer das globale Optimum des Lösungsraumes, wenn sie überhaupt etwas liefern. Heuristische Verfahren hingegen garantieren nicht, das globale Optimum zu finden, sondern womöglich nur ein Lokales, garantieren aber die Terminierung mit einer — möglicherweise nicht gültigen — Lösung. In den nun folgenden beiden Abschnitten werden die einzelnen Ansätze jeweils weiter verfeinert. Problem-spezifische Optimierungsverfahren sind hingegen in Abbildung 5.1 nicht enthalten, sie lassen sich aber aufgrund ihrer zugrundeliegenden Methodik in einen der verschiedenen Bereiche einordnen.

5.1.1 Exakte Verfahren

Exakte Verfahren liefern das tatsächliche Optimum des Verteilungsproblems, leiden aber inhärent an der exponentiellen Komplexität des Problems. So sind exakte Ver-

fahren durchaus gewinnbringend bei kleinen Problemgrößen anwendbar, scheitern aber meist an einer verfahrensspezifischen Grenze. Im folgenden sollen die bekanntesten vorgestellt und auf ihre Anwendbarkeit auf das im vorangegangenen Kapitel dargestellte Problem der Platzierungswahl untersucht werden.

Vollständige Aufzählung

Die vollständige Aufzählung durchmustert den kompletten Suchraum, indem entweder Tiefen- oder Breitentraversierung auf dem Baum der möglichen Kombinationen einer Platzierung angewendet wird. Da alle Lösungen betrachtet werden, garantiert dieses Verfahren als Ergebnis die global optimale Lösung zu finden. Allerdings ist aufgrund des expliziten Absuchens des Lösungsraumes dieses Verfahren nur mit sehr kleinen Probleminstanzen effektiv einsetzbar, da ansonsten die Laufzeit nicht mehr tolerierbar ist.

Branch & Bound

Das Lösungsprinzip Branch & Bound[159] basiert auf der schrittweisen Zerlegung des Gesamtproblems in Teilprobleme und die Untersuchung der möglichen besten und schlechtesten Werte einer Zielfunktion des jeweiligen Teilpropblems. Im Verzweigungsschritt (Branching) wird dazu das zu lösende Problem in zwei oder mehr Teilprobleme zerlegt. Dies geschieht wiederum mit jedem entstandenen Teilproblem, so dass sich insgesamt eine Baumstruktur von Teilproblemen ergibt. Durch das Beschränken (Bounding,Pruning) kann dann ein Teil der Probleme aussortiert werden: Jene Teilprobleme, deren beste geschätzte mögliche Lösung schlechter als eine schon bekannte Lösung des Gesamtproblems ist, brauchen nicht weiter berücksichtigt zu werden und können aussortiert werden. Dadurch kann der zu untersuchende Baum von Teilproblemen beschränkt werden.

Integer Linear Programming

Integer Linear Programming[21] basiert zunächst auf sogenannten linearen Programmen, eine Form von linearen arithmetischen Formeln über reellwertigen Variablen, dessen Lösungen durch numerische Methoden effizient ermittelt werden können. Zur Optimierung wird die Optimierungsfunktion in den Satz linearer Formeln integriert. Ganzzahlige Parameter werden eingeführt, indem zunächst eine sogenannte relaxierte Lösung errechnet wird, die eine reellwertige Interpretation der ganzzahligen Parameter verwenden. Anschließend werden Rundungsfunktionen benutzt, die üblicherweise auf Heuristiken beruhen, um diese Parameter wieder in den ganzzahligen Bereich zu überführen, und gleichzeitig wird die Gültigkeit dieser ganzzahligen Lösung ermittelt. Eine iterative Anwendung dieses Verfahrens sorgt dafür, dass es sich schrittweise der optimalen Lösung annähert, indem durch die gültigen gefundenen ganzzahligen Lösungen der Lösungsraum beschränkt wird. Lineare Gleichungssysteme bilden dabei immer ein konvexes Polyeder als Lösungsraum, aus dem die gefundenen ganzzahligen Lösungen in Form von Ebenen im $\mathbb{R}^n$ Teile herausschneiden, die aufgrund der ganzzahligen Natur der Parameter keine gültigen Lösungen darstellen.

ILP-Techniken stellen für kleine Anzahlen an ganzzahligen Variablen eine effiziente Optimierungsmethodik zur Verfügung, allerdings müssen die Rundungsfunktionen sorgfältig ausgewählt werden. Zusätzlich kann dieses Verfahren, wie der Name es schon sagt, nur mit linearen Gleichungssystemen umgehen, da nicht-lineare Anteile die konvexe Eigenschaft des Lösungsraum gegebenenfalls verändern. In konkaven Polyedern können aber Teilräume des Lösungsraumes nicht einfach durch Ebenen herausgeschnitten werden.

Verifikationstechnik-basierte Verfahren

Verifikationsbasierte Techniken stellen zunächst einmal an sich kein Optimierungsverfahren zur Verfügung, sondern liefern im allgemeinen Antworten auf reine Erfüllbarkeitsprobleme. Um große Zustandsräume analysieren zu können, werden unterschiedliche Techniken der Komprimierung der Darstellung des Zustandsraumes oder der Steigerung der Effizienz der Durchmusterung desselben eingeführt. Die beiden wesentlichen Vertreter der automatischen Verifikationsmethodiken[1] sind das sogenannte Model Checking[112] und das sogenannte SAT-Checking[122]. Model Checking für große Zustandsräume benutzt die komprimierenden Eigenschaften von BDDs[113] im sogenannten symbolischen Model Checking. Hierbei wird der Zustandsraum nicht explizit dargestellt, sondern symbolisch durch eine Transitionsrelation der Menge der Zustände, deren Kodierung mithilfe von BDDs im allgemeinen sehr effizient durchgeführt werden kann. SAT-Verfahren auf der anderen Seite versuchen auf einem Satz konjugierter Boolescher Formeln die Erfüllbarkeit nachzuweisen. Effiziente heuristische Algorithmen, wie beispielsweise der Davis-Putnam-Algorithmus[11], versuchen, während der sukzessiven Belegung der Variablen des Gleichungssatzes möglichst früh Widersprüche zu entdecken, um ganze Teilbäume des Lösungsraumes abzuschneiden.

Beide Verfahren liefern zunächst nur eine Lösung oder einen Widerspruch. Eingebettet in ein iteratives Verfahren können sie jedoch, ähnlich wie ILP, genutzt werden, die optimale Lösung einer Probleminstanz zu finden. Hierzu wird ein dem Abstraction-Refinement-Ansatz äquivalentes Vorgehen benötigt, dass die Optimierungsfunktion als zu erfüllende Proposition kodiert und diese schrittweise verfeinert. Sowohl SAT-Techniken als auch symbolisches Model Checking leidet jedoch unter der im allgemeinen exponentiellen Berechnungszeit, die zwar für Modelle beachtlicher Größe aufgrund von Komprimierung und effizienter Verfahren tolerierbar ist, aber ab einer gewissen Schranke die Verfahren nicht mehr anwendbar machen.

5.1.2 Heuristische Verfahren

Heuristische Verfahren verwenden — teilweise problemspezifische — Heuristiken, die ein ziel-orientiertes Durchsuchen des Lösungsraumes ermöglichen. Die Hoffnung hierbei ist es, auch für große Probleminstanzen innerhalb erträglicher Rechenzeiten mehr oder weniger optimale Lösungen erreichen zu können. Optimalität kann dabei

[1] Nicht- oder Semi-automatische Verfahren können in Bezug auf eine Optimierung nicht angewendet werden, da dies dem generellen Prinzip einer automatischen Platzierung widerspricht.

allerdings nicht garantiert werden und es ist auch i.a. nicht möglich, Konfidenzintervalle zur optimalen Lösung anzugeben. Allerdings zeigen Versuche, dass sich mittels dieser Strategien durchaus annehmbare Lösungen für eine ganze Reihe von Problemen erzielen lassen.
Bei heuristischen Ansätzen wird grundsätzlich zwischen deterministischen und stochastischen Verfahren unterschieden, die sich im wesentlichen in der Reproduzierbarkeit der Lösungen auszeichnen. Deterministische Verfahren verfolgen immer den selben Lösungsweg und kommen deshalb auch immer zur selben Lösung. Stochastische Ansätze hingegen verwenden den Einfluss des Zufalls, um eine Lösung zu finden. Sie basieren oftmals auf Beobachtungen des Verhaltens natürlicher (z.B. biologischer oder physikalischer) Prozesse. Wir werden im folgenden beide Varianten kurz beschreiben.

Deterministische Verfahren

Ein bekannter Vertreter der deterministischen Verfahren ist der sogenannte *Greedy*-Algorithmus. Er benutzt eine Folge von lokal optimalen Entscheidungen um damit eine gute Lösung des Gesamtproblems zu ermitteln. Bei einigen Problemen liefern Greedy-Algorithmen auch die globalen Optima, allerdings ist dies in der Regel nicht der Fall. Aufgrund ihrer normalerweise sehr einfachen Struktur zeichnen sie sich durch eine sehr kurze Laufzeit aus, leiden aber massiv darunter, dass lokal optimale Entscheidungen sehr schnell zu Lösungen führen, die vom globalen Optimum einen sehr großen Abstand haben. Aus diesem Grund werden sie vorwiegend zur Bestimmung von Anfangslösungen eingesetzt, die anschließend durch andere Verfahren verbessert werden.

Ein weiterer deterministischer Algorithmus untersucht die Nachbarschaft einer bekannten Ausgangslösung, die beispielsweise mit einem Greedy-Algorithmus erzeugt wurde. Unter allen möglichen Nachbarschaftslösungen wird diejenige ausgewählt, die den besten Wert der Optimierungsfunktion aufweist. Diese Nachbarschaftslösung wird als Ausgangspunkt für eine erneute Iteration des Verfahrens genommen. Dabei werden unter Umständen auch schlechtere Lösungen akzeptiert, wenn keine besseren gefunden wurden. Auf diese Art und Weise ist es möglich, lokale Optima zu verlassen. Um zu vermeiden, dass bei einer erneuten Iteration wiederum die zuvor besuchte Lösung ausgewählt wird, weil diese gegebenfalls besser sein kann, werden derartige Lösungen verboten. Daher wird dieses Verfahren *Tabu-Search*[63] genannt. Die Liste der verbotenen Lösungen ist naturgemäß in ihrer Länge beschränkt und bildet das entscheidende Kriterium für die Eigenschaft, aus lokalen Optima zu entfliehen.

Ein weiterer heuristischer Ansatz ist das sogenannte Gradientenverfahren[105]. Auch dieses Verfahren untersucht, ähnlich dem Tabu-Search, Nachbarschaftslösungen, wobei es die Optimierungsfunktion als Potentialfeld interpretiert und die Steigungen auf dem Weg zu einer Lösung in diesem Feld benutzt, um zu entschieden, welche Nachbarschaftslösung akzeptiert wird und welche nicht.

Neben diesen generellen deterministischen Verfahren gibt es speziall auf die jeweili-

ge Problemklasse zugeschnittene Verfahren. Aus dem Bereich der hier betrachteten Problemklasse des Scheduling und der Platzierung sind hier vorwiegend Graphen-basierte Verfahren zu nennen, z.B. [6]. Es existieren aber auch Ansätze, das Platzierungsproblem mit Methoden der künstlichen Intelligenz zu lösen[74].

Simulated Annealing

Simulated Annealing[1] ist ein Vertreter der stochastischen Methoden. Es erlaubt die Suche nach einem globalen Optimum auch bei mehreren sich widersprechenden Optimierungskriterien. Als naturanaloges Verfahren basiert es auf einem physikalischen Prozess der Thermodynamik: Ein geschmolzenes Material wird dabei so langsam abgekühlt, dass die Moleküle ein Kristallgitter minimaler Energie ausbilden können. Der Energiegehalt einer Kristallstruktur entspricht hierbei dem Wert der Kostenfunktion für eine bestimmte Lösung. Die Temperatur steht in Analogie zu einem Parameter, der die Optimierung steuert.
Das Verfahren besteht in erster Linie aus einem stochastischen Absuchen des Lösungsraumes. In jedem Temperaturschritt werden mehrere mögliche Lösungen berechnet und bewertet. Verbesserungen gegenüber bereits gefundenen Lösungen werden immer akzeptiert. Bringt die untersuchte Nachbarschafts-Lösung aber keine Verbesserung, wird sie trotzdem mit einer gewissen Wahrscheinlichkeit angenommen, die mit fallender Temperatur abnimmt. Die Wahrscheinlichkeitsfunktion hierfür lautet:

$$P = e^{-\frac{\Delta E}{kT}},$$

wobei k die Boltzmann-Konstante und T die Temperatur ist. $\Delta E = E_{new} - E_{best}$ ist die Energiedifferenz zwischen der neuen und der besten bisher bekannten Lösung. Ist $\Delta E < 0$, wird die Wahrscheinlichkeit größer 1 und damit die neue Lösung als bessere Lösung akzeptiert. Ist hingegen $\Delta E \geq 0$, so hängt die Akzeptanz der dann schlechteren neuen Lösung von einem Zufallswert ab, der mit der Wahrscheinlichkeitsfunktion verglichen wird. In jedem Temperaturschritt wird eine festgelegte Anzahl an Lösungen untersucht und die Temperatur anschließend abgesenkt. Der Abkühlungsplan der Temperatur wird üblicherweise durch eine geometrische Folge der Form $T^{i+1} = \alpha T^i$ festgelegt, wobei α meist im Bereich $0.9 \leq \alpha < 1.0$ gewählt wird. Ausgangspunkt für dieses Verfahren ist eine Anfangslösung, von der aus der Nachbarschaftslösungsraum abgesucht werden kann. Es ist dabei zu bemerken, dass die Anfangslösung keine gültige Lösung sein muss, um zu einer gültigen Lösung zu gelangen; Die Energiefunktion muss nur so beschaffen sein, dass sie ungültige Lösungen bestraft.
Simulated Annealing leidet unter einer starken Problemabhängigkeit, die dazu führt, dass zwar bei hohen Temperaturen der gesamte Suchraum abgesucht werden kann, bei fallender Temperatur aber das lokale Optimum in der Nähe der bisher gefundenen besten Lösung annähert. Im Idealfall ist dies gerade das globale Optimum, dazu müsste aber lange bei hohen Temperaturen gesucht werden. Dies wiederum hat sehr lange Laufzeiten zur Folge, so dass hier ein Kompromiss gefunden werden muss, der lange genug bei hohen Temperaturen sucht, um lokalen Optima zu entkommen, der sich jedoch bei niedrigen Temperaturen schnell an das Optimum annähert. Eine diesbezügliche Erweiterung dieses Algorithmus ist in [102] vorgeschlagen, das soge-

nannte Localized Simulated Annealing. Die grundlegende Idee dieses Verfahrens ist die Aufteilung des Lösungsraumes in disjunkte Teilräume, die jeweils eine lokalen Simulated Annealing Suche durchführen, aber an einen globalen Abkühlungsplan der Temperatur angekoppelt sind. Auf diese Weise wird versucht, mehrere Punkte im Lösungsraum zu finden mit einer insgesamt höheren Wahrscheinlichkeit, das globale Optimum zu finden.

Genetische Algorithmen

Ein weiteres populäres stochastisches Verfahren sind die Genetischen Algorithmen [161]. Sie basieren auf der Beobachtung, dass sich Lebewesen in der Natur über Generationen hinweg immer besser an ihre Umwelt anpassen. Dies wird als Evolution bezeichnet und Genetische Algorithmen versuchen, diesen biologischen Prozess auf eine Anwendung in der Technik zu übertragen.
Die gesamte genetische Information eines Lebewesens oder Individuums ist in Chromosomen enthalten. Chromosomen selbst sind aus einzelnen Bausteinen, den sogenannten Genen zusammengesetzt. Die Gesamtheit der Gene ist für die Ausprägung eines bestimmten Individuums verantwortlich, wobei mehrere Individuen eine sogenannte Population bilden. Die Strategie der Evolution benutzt nun drei wesentliche Prinzipien zur Bildung neuer Populationen: Die Rekombination der genetischen Information der beiden Eltern eines Individuums, die Modifikation oder Mutation der Gene und die Selektion von Individuen.
Bei der Rekombination werden die Chromosomen der beiden Eltern zu neuen Chromosomen kombiniert, wobei die Kombination nur an der Grenze von Genen möglich ist. An diesen Stellen findet dann durch das sogenannte Crossover auch zufällig ein paarweise Tausch der genetischen Informationen statt. Völlig neue Eigenschaften lassen sich auf diese Weise nicht aus der genetischen Information der Eltern erzeugen, stattdessen werden sie lediglich neu kombiniert. Erst die Mutation sorgt für neue Eigenschaften des Individuums, indem ein oder mehrere Gene zufällig verändert werden. In der Natur geschieht dies beispielsweise durch energiereiche Strahlung, chemische Prozesse, etc. Sporadisch entstehen hierbau auch Individuen, die nicht lebensfähig sind.
Die Selektion der Individuen basiert auf dem Prinzip “Survival of the fittest”, wodurch die am besten an die Umwelt angepassten Individuen innerhalb einer Population die größten Chancen erhalten, sich zu reproduzieren. Weniger gut angepasste Individuen können für die nächste Population entweder nur wenige oder gar keine Nachkommen erzeugen.
Um das Verfahren für beliebige Optmierungsprobleme anwenden zu können, müssen die Lösungen in Form von Chromosomen dargestellt werden. Dazu eignet sich eine binäre Darstellung in Form von Bitketten mit Hilfe des Gray-Codes besonders gut, da diese Art der Kodierung garantiert, dass zwei Lösungen, die in unmittelbarer Nachbarschaft zueinander liegen, sich nur in einem einzigen Bit unterscheiden. Damit ist der Einfluss der Mutation für alle Bitpositionen gleich groß. Zusätzlich muss für die Selektion eine Kostenfunktion vorhanden sein, mit der sich die Fitness eines Individuums bestimmen lässt. Diese kann ähnlich wie im Simulated Annealing Verfahren gewählt werden und sollte ebenfalls ungültige Lösungen bestrafen.

Die Schwierigkeiten bei den Genetischen Algorithmen ist die geeignete Auswahl der Populationsgröße sowie die Wahrscheinlichkeit von Crossover und Mutation. Ebenfalls sehr schwierig ist die Wahl einer geeigneten Problemkodierung, mit der möglichst viele überlebensfähige Individuen erzeugt werden und bei der der Einfluss von Mutation über alle Gene gleichverteilt ist.

Eine vereinfachene Abwandlung genetischer Algorithmen sind die sogenannten Evolutionsstrategien[150], die das natürliche Vorbild der Evolution nicht mittels komplexer Chromosomen abbilden, sondern stattdessen Vektoren reellwertiger Zahlen zu Genen vereinen und mit diesen den biologisch-analogen Auswahlprozess durchführen. Evolutionsstrategien konzentrieren sich nicht so sehr auf die Art und Weise der Informationskodierung, wie dies genetische Algorithmen tun, sondern legen den Schwerpunkt auf die Optimierung selbst und sind daher nicht so anfällig bezüglich der Kodierung von Informationen, können aber auch nur eine beschränkte Menge von Problemklassen lösen.

Dieses letzte Verfahren soll an dieser Stelle die Beschreibung der üblicherweise angewandten Verfahren beenden und im folgenden soll anhand der Literatur herausgearbeitet werden, welche Ansätze sich mit der Problemstellung der Platzierung und des Scheduling verteilter eingebetteter Echtzeitsysteme beschäftigen, welche Schwächen und welche Stärken diese Ansätze haben.

5.2 Stand der Technik

Basierend auf der Vielzahl zur Verfügung stehenden Optimierungsmethoden existieren zahlreiche Arbeiten, die sich bereits seit Ende der Siebziger Jahre mit dem Thema Task Allokation auf verteilten Systemen beschäftigen. Die Ursprünge sind im Bereich der Multiprozessor-Systeme zu suchen und versuchen unter dem Aspekt des Load-Balancing eine möglichst optimale Ausnutzung der Prozessorresourcen zu berechnen, die zu einer gesteigerten durchschnittlichen Antwortzeit von Prozessen führen soll. Stellvertretend seien hierfür die Arbeiten in [155], [20] sowie [105] genannt. Letzteres verwendet beispielsweise das in Kapitel 5.1.2 vorgestellte Gradientenverfahren um eine Verbesserung der Prozess-Allokation zu erreichen. Für die hier betrachteten harten Echtzeitsysteme eignen sich diese Verfahren jedoch nicht, da sie zumeist von einer sich während der Laufzeit ändernden Zuweisung der Prozesse zu Prozessoren ausgehen. Dies macht in klassischen Multiprozessor-Systemen mit ihren auf mittleren Durchsatz zugeschnittenen Architekturen auch durchaus Sinn, ist jedoch für Echtzeitsysteme aufgrund der notwendigen Techniken zur Migration von Tasks ungeeignet; Letztlich müssten diese Task-Migrationen in die Echtzeitanalyse integriert werden und würden dazu führen, dass im schlimmsten anzunehmenden Fall grundsätzlich von einer Migration der zu analysierenden Task ausgegangen werden muss. Die dafür entstehenden Kosten (Übertragung der Daten und evtl. Übertragung des Programmcodes) dürften in harten Echtzeitsystemen mit teilweise sehr engen Zeitschranken in nahezu allen Fällen zu keiner validen Lösung führen und verursachen zudem immense Verzögerungen auf den zur Übertragung benutzten Kommunikationsmedien.

Erste Arbeiten zur Task Allokation auf dem Gebiet der harten Echtzeit-Systeme geben die Autoren von [43] an, in der sie die in [104] präsentierten Analysen zur Bestimmung der Feasibility in einem System mit raten-monotonem Scheduling Algorithmus für eine Reihe von problemspezifischen heuristischen Taskzuweisungen betrachten. Allerdings werden dort keine Nachrichtenübermittlungen über Kommunikationsmedien akkurat betrachtet, ebensowenig wie Antwortzeiten, sondern es wird stattdessen lediglich die Feasibility anhand der hinlänglich bekannten Kriterien der raten-monotonen Analyse verwendet. Abhängig vom Optimierungskriterium können Ansatze zur Task Allokation sich deutlich unterscheiden. Generell ist bei den meisten frühen Arbeiten auf diesem Gebiet die Minimierung der Ausführungskosten von Tasks und deren Nachrichten Ziel einer Optimierung. Dieses erlaubt allerdings zunächst keine Aussage darüber, ob die Deadlines von Tasks eingehalten werden können oder nicht.
Graphen-theoretische Optimierungsverfahren in diesem Kontext sind beispielsweise in [175], [176] und [6] betrachtet worden. Letzteres führt die sogenannte Slack Method ein, mit der ein Taskgraph in einen Graphen kritischer Pfade überführt wird. Anschließend wird dieser Graph auf ein Grid gleicher Prozessorknoten platziert. Die Transformation des Taskgraphen fasst zwecks Minimierung des Suchraumes Taskpaare mit hohem Kommunikationsaufwand sukzessive in zusammenhängend zu platzierende Verbünde zusammen, womit die Optimierung der Platzierung von der Slackness getrieben wird, die die Latenzzeit der Pfade übrig lässt. Neben der für Applikationen in der hier betrachteten Anwendungsdomäne nicht adequaten Vorgehensweise des sukzessiven Clusterns von Tasks mit Nachrichten werden weder exakte Antwortzeiten noch Latenzzeiten von Nachrichtenübertragungen akkurat behandelt. Zudem ist die Architektur auf ein regelmäßiges Grid-Feld homogener Prozessorknoten beschränkt. Ebenfalls auf die Zusammenfassung von Tasks zu Clustern, die immer zusammenhängend platziert werden, setzen die Ansätze in [148], wobei allerings nur nicht-preemptives Scheduling untersucht wird und nur homogene Topologien unterstützt werden. Ähnlich Einschränkungen der Architekturtopologie finden sich auch in weiteren Arbeiten, beispielsweise Hypercubes in [187], hexagonale Architekturen in [165] oder auch zweidimensionale Mesh-basierte Topologien in [205].
Insgesamt ist es jedoch schwierig, Echtzeiteigenschaften innerhalb von Optimierungsproblemen in eine graphentheoretische Darstellung zu integrieren. Aus diesem Grund nutzt die Mehrzahl der Arbeiten auf dem Gebiet der optimierenden Task Allokation heuristische oder optimale Verfahren. In [85] wird beispielsweise ein Entscheidungsbaum für das Task Scheduling aufgebaut, der mittels eines heuristischen Suchverfahrens namens Critical Path/Most Immediate Successors First traversiert wird. Ziel der Optimierung ist hierbei die Minimierung der Länge eines Schedules. [30] hingegen optimiert die maximale Auslastung der Prozessoren, wobei die Auslastung aus der Summe der Ausführungszeiten und der Übertragungszeiten von Nachrichten berechnet wird. Die Zuordnung der Tasks zu Prozessorknoten wird hierbei von einer problemspezifischen Heuristik getrieben, die auf der Basis einer Wartezeit-Rate von Tasks eine Veränderung einer gegebenen initialen Platzierung vornimmt. In einem leicht veränderten Modell zeigt [31] die Anwendung eines optimalen Verfahrens im Sinne der Minimierung der durchschnittlichen Antwortzeit von Tasks,

allerdings ist dieser Ansatz aufgrund der Verwendung mittlerer Antwortzeiten für harte Echtzeitsysteme nicht geeignet. Branch & Bound Methoden zur Reduzierung der Ausführungszeiten und Kommunikationslatenzen werden in [109], [164] und [167] verwendet. Aber auch für diese Ansätze gilt, dass sie für harte Echtzeitanwendungen nicht geeignet sind, da die Optimierung zwar generell den Zeitverbrauch minimieren kann, dies aber keinerlei Rückschlüsse über die Einhaltung von Deadlines erlaubt. So kann es in diesem Sinne durchaus sinnvoller sein, einen größeren Zeitverbrauch zu erlauben, wenn dann alle Deadlines eingehalten werden können.
Bei der Verwendung stochastischer Optimierungsmethoden gibt es ebenfalls eine ganze Reihe von Arbeiten, vornehmlich auf Basis von Simulated Annealing und Genetischen Algorithmen. Simulated Annealing wird in [196], [13] und [183] verwendet. Insbesondere [183] ist hier zu nennen, da dort erstmals die Integration von Antwortzeiten für Nachrichten in das Optimierungsverfahren gelungen ist und nicht, wie in nahezu allen anderen Ansätzen, diese nur abstrakt oder abschätzend quantifiziert wird. Die untersuchte Topologie ist beschränkt auf ein einfaches, aber echtzeitfähiges Busprotokoll (Tokenring), an dem gleichartige Prozessoren angebunden sind, die sich lediglich bzgl. Resourcenbedingungen unterscheiden. Basierend auf dieser Arbeit ist in [178] eine Erweiterung auch für komplexere Bussysteme, wie CAN oder FlexRay, entstanden und eine Adaptierung an Localized Simulated Annealing vorgenommen worden.
Genetische Algorithmen sind in [153], [130] und [171] auf das Platzierungsproblem angewandt worden. Sowohl [130] als auch [171] kodieren dabei die Möglichkeit hierarchischer Topologien nahezu identisch in die Chromosomenstruktur und stellen damit erstmals ein Verfahren für komplexere Architekturen zur Verfügung, das unter anderem die Kosten der Nachrichtenübertragung mittels Gateway-Prozessoren von einem Kommunikationsmedium auf ein anderes Medium berücksichtigt. Es sind jedoch nur einfache Kommunikationspfade möglich, und das modellierte Bussystem beschränkt sich im wesentlichen auf die Modellierung aus [183]. Ebenfalls genetische Algorithmen werden in [67] eingesetzt, die zwar heterogene Architekturen als Zielarchietktur nutzen können, allerdings auf die Modellierung echtzeit-fähiger Kommunikationsmedien verzichten und stattdessen einfachere Bussysteme im Kontext von System-on-Chip Lösungen einsetzen.
Ansätze zur Lösung des Taskplatzierungs-Problems mittels optimaler Optimierungsverfahren sind im wesentlichen für Branch & Bound Verfahren und Integer Linear Programming zu finden. In [204] wird ein optimaler Branch & Bound Algorithmus beschrieben, der Taskallokationen für einfache Verteilte Systeme durchführt. Der Pruning-Part des Algorithmus bezieht sich hierbei speziell auf bestimmte Reihenfolge- und Ausschlusskriterien, die das betrachtete Tasksystem aufweisen muss. Allgemeinere Tasksysteme können mittels der Methodik aus [139] behandelt werden, die auf Basis einer normalisierten Antwortzeit von Tasks eine Platzierung von Tasks auf Prozessoren und ein Schedule auf den jeweiligen Prozessoren durch ein Branch & Bound Verfahren berechnen. Die Tasksysteme sind jedoch auf zeitbasierte Tasks ohne Preemptionen eingeschränkt, und die Übermittlung von Nachrichten wird nicht exakt modelliert; Dementsprechend sind auch keine komplexeren Topologien mit Hilfe dieses Verfahrens zu analysieren/synthetisieren.
Im Bereich Constraint Programming und Integer Linear Programming sind eben-

falls einige Ansätze zu finden, die sich mit dem Problem der Taskplatzierung auf verteilten Echtzeit-Systemen beschäftigen, dies jedoch nur auf eingeschränkten Problemklassen ermöglichen. [160] ermittelt zunächst durch ein stochastisches Verfahren eine Ausgangslösung bzgl. der sogenannten Total Execution Time. Dies Ausgangslösung wird anschließend als obere Grenze festgesetzt und Integer Linear Programming durchsucht den damit verbleibenden kleineren Lösungsraum bis hin zur optimalen Lösung. Leider kann dieser Ansatz nur mit sehr beschränkten Tasksystemen umgehen, nämlich solche, die aus einer einzigen Taskkette mit einer einzigen End-to-End Deadline bestehen, da ansonsten nicht notwendigerweise eine valide Platzierung errechnet wird. In [14] wird Integer Linear Programming genutzt, um statische Schedules für zeit-basierte Tasksysteme zu generieren, wobei das Tasksystem nur Tasks gleicher Periode erlaubt und das System nicht-preemptiv ist. Es werden zusätzlich nur einfache Topologien mit zeit-basierten Bussystemen erlaubt. Komplexere Tasksysteme können mit dem in [27] vorgestellten Ansatz platziert werden, wobei auf Systemklassen, wie sie in [183] vorgestellt wurden, gezielt wird. Das Verfahren ist in zwei Stufen aufgeteilt: In der ersten Stufe, das sogenannte Master-Problem, wird zunächst mittels Constraint Programming eine Zuweisung von Tasks zu Prozessorknoten durchgeführt, die nicht unbedingt auch eine gültige Platzierung im Sinne der Einhaltung aller Deadlines sein muss. Die zweite Phase überprüft diese Platzierung auf Feasibility und erzeugt dabei lokal pro Prozessor Schedules. Im Falle von nicht-validen Platzierungen erzeugt die zweite Phase zusätzliche einschränkende Constraints, sogenannte “nogoods”, für eine weitere Iteration der ersten Phase. Behandelt werden können mit diesem Ansatz statisch priorisierte, preemptive Tasksysteme und eine einfache Topologie mit einem Tokenring als zentrales Kommunikationsmedium.
Tabu Search basierende Methoden werden in [45] und [168] eingesetzt. [168] erweitert hierbei das Verfahren zum sogenannten Multiobjective Tabu Search und zeigt die Anwendbarkeit auf ratenmonotones Scheduling unter Verwendung der bekannten Kriterien hierfür aus [104], mit der Konsequenz, dass keine Antwortzeiten errechnet werden können, sondern lediglich die Feasibility dieses einfachen Tasksystems nachzuweisen ist. Kommunikationslatenzen werden nicht akkurat ermittelt, ebensowenig können komplexere Topologien mittels dieses Verfahrens betrachtet werden.
Unter dem Blickwinkel des Entwurfsprozesses sind noch eine Reihe weiterer Arbeiten im Umfeld des Task-Allokations Problemes erwähnenswert. Zunächst die oben schon beschriebene Arbeit [67], die eine Platzierung mittels genetischer Algorithmen durchführt. Gleichzeitig bietet der Ansatz eine Methodik zur Komposition von Tasks unterschiedlicher Klassen, beispielsweise Zeit- oder Ereignis-basiert. Basierend auf den Arbeiten von [65] und [126] werden für jede Task Aktivierungs-Ereignisströme betrachtet, die unter anderem durch eingehende Nachrichten verursacht werden. Verschiedenartige Kopplungen (AND- und OR-Kopplung, sowie zyklische Abhängigkeiten) können damit effizient behandelt werden. Diese Vorgehensweise ist damit feingranularer als die im Kontext dieser Arbeit in Kapitel 4.1 vorgestellte Forderung, dass Zieltasks mehrerer Nachrichten immer zeit-basiert sein müssen. [125] beschreibt hingegen einen typischen Entwurfszyklus eingebetteter Systeme in der Automobilelektronik mit ihren frühen Entwurfsraum-Explorationen und gibt dafür eine Design-Methodik an. Platzierungsverfahren für unterschiedliche Grade an Wis-

sen über das zu entwickelnde System in den verschiedenen Entwurfsstadien helfen hier mit abschätzenden Informationen und Zuordnungen, die mit einem Simulated Annealing Ansatz durchgeführt werden. Zwar werden komplexere Topologien als typische Zielarchitektur genannt, jedoch fehlt dem Ansatz jegliche Betrachtung von Echtzeit sowohl der Tasksysteme als auch der Nachrichtenübermittlung. [129] geht hier einen Schritt weiter und integriert Echtzeitaussagen, allerdings nur auf der Basis der Feasibility-Kriterien nach [104]. Bussysteme werden nicht akkurat modelliert, sondern auch hier kommt nur die Auslastung eines abstrakten Kommunikationsmediums in die Analyse hinein. Diese Arbeit ist ebenfalls im Umfeld der Automobilelektronik angesiedelt und ergänzt das Platzierungsverfahren durch eine Entwurfsmethodik. Die Bestimmung der Allokation von Tasks selbst wird durch einen heuristischen Ansatz namens Best Gain First berechnet, der sich sehr stark an dem Ansatz zur Multipartitionierung von Mengen von Kernigham und Lin [87] orientiert. In [144] werden hingegen die Kommunikationslatenzen für ein TDMA-basiertes Kommunikationsmedium akkurat modelliert, ebenso wie die Zeiten für Taskausführung. Allerdings beschränkt sich die Lösung nur auf nicht-preemptive, zeit-basierte Tasksysteme. Für die Platzierungsberechnung wird ein zweistufiges Verfahren benutzt, das in der ersten Stufe eine valide Initiallösung mittels des sogenannten Heterogeneous-Critical-Path Algorithmus[81] (eine Variante von List Scheduling-Verfahren) erzeugt. In der zweiten Stufe wird dann diese Initiallösung entweder mittels Simulated Annealing oder aber mit einer propreitären Mappingheuristik optimiert. Das Optimierungsziel ist dabei so gewählt, dass sowohl auf Prozessoren als auch auf Kommunikationsmedien möglichst viel Freiraum bleibt und dies Verfahren auf diese Weise ein inkrementelles Design-Vorgehen unterstützt.

5.3 Berechnung von Verteilungen

5.3.1 Motivation für die Wahl des Verfahrens

Zur Berechnung der Allokation von Tasks und Nachrichten unter den vielschichtigen Randbedingungen, die in Kapitel 4 diskutiert wurden, muss nun ein geeignetes Optimierungsverfahren gefunden werden. Wie bereits im letzten Abschnitt dargelegt, existiert eine Vielzahl von Ansätzen, das Verteilungsproblem zu lösen, die jedoch sämtlichst keine heterogenen, hierarchischen Systeme betrachten und oftmals nur sehr eingeschränkte Problemklassen betrachten. Es verbietet sich daher, undifferenziert eine Ähnlichkeit der Struktur des Lösungsraumes einfacher Problemklassen auf die hier betrachtete Problemklasse zu postulieren. Die allermeisten heuristischen und exakten Optimierungsverfahren profitieren von mehr oder weniger stetigen Lösungsräumen mit weichen Flanken hin zu einer optimalen Lösung oder lokalen Optima: Eine Task in einer einfachen Problemklasse ohne viele Randbedingungen und hierarchische Topologien von einen Prozessorknoten zu einem anderen zu verschieben, ändert im allgemeinen nicht abrupt den Wert der Energiefunktion (die die Güte der Lösung angibt) und produziert flache Flanken in der Nachbarschaftsbeziehung möglicher Lösungen. In komplexen Systemen, wie dem hier betrachteten, ist dies nicht so: Das Verschieben einer Task auf einen anderen Prozessorknoten kann den

Wert der Energiefunktion abrupt ändern, da im Zuge der Umordnung der Tasks auch Nachrichten auf andere Bussysteme gelegt werden müssen, was wiederum auch dort die lokale Energiefunktion beeinflusst (vgl. die Aussagen zur Struktur des Lösungsraumes in [129]). Zudem sind in hierarchischen Systemen womöglich nicht nur einzelne Bussysteme zu betrachten, sondern aufgrund komplexerer Pfade eine ganze Reihe von ihnen. Insgesamt ergeben sich damit aufgrund kleinster Änderungen einer möglichen Lösung potentiell sehr steile Flanken der Energiefunktion. Überträgt man die Energiefunktion der Optimierung unter Angabe einer Lösung auf eine zweidimenionale Darstellung, gehen wir für die hier betrachtete Problemklasse von einem eher zerklüfteten Gebirge mit tiefen Tälern und steilen Gipfeln aus. Die vielen unterschiedlichen, sich womöglich widersprechenden, Randbedingungen sorgen zudem dafür, dass die Zahl der validen Lösungen gering ist und es daher auch sehr schwer fällt, mittels einfacher Optimierungsverfahren, wie dem Greedy-Verfahren überhaupt eine valide Ausgangslösung als Startpunkt zu finden.

Nach dieser intuitiven Vorbetrachtung sollen nun die einzelnen Optimierungsverfahren, die in Kapitel 5.1 vorgestellt wurden, auf ihre Anwendbarkeit hin überprüft werden. Dabei sind natürlich aufgrund ihrer garantierten Optimalität exakte Verfahren zu bevorzugen, wenn sich denn die auftretenden Komplexitäten beherrschen lassen. Für das vollständige Aufzählen aller möglichen Lösungen als einfachstes optimales Verfahren gilt dies offensichtlich schon bei kleinen Problemgrößen nicht mehr. Das Branch & Bound-Verfahren liefert hier einen deutlich besseren Beitrag, da es frühzeitig invalide Lösungsteilräume abschneidet. Allerdings benötigt man hierfür eine sichere obere und untere Schranke der Energiefunktion. In der hier betrachteten Problemklasse kann dies aber nur berechnet werden, wenn die optimale Lösung bekannt ist, da es sehr viele sich gegenseitig beeinflussende Faktoren gibt, die eine formale oder algorithmische Beziehung nicht möglich machen[1]. Damit scheidet auch dieses exakte Verfahren aus. Integer Linear Programming ist hingegen ein sehr vielversprechendes Optimierungsverfahren, dass aufgrund seiner Lösungsstrategie nicht von unstrukturierten Lösungsräumen betroffen ist. Zudem ist es im allgemeinen ein sehr effizientes Verfahren, welches aber leider nur mit linearen Problemen umgehen kann. Wir werden im folgenden sehen, dass dies nicht immer ausreichend ist, so dass ILP nur für Teilprobleme der hier betrachteten Problemklasse eingesetzt werden kann. Verifikationsbasierte Techniken hingegen haben diesen Nachteil im Prinzip nicht und sind zudem auch unabhängig von der Struktur des Lösungsraumes. Allerdings leiden sie latent unter dem Problem der Zustandsexplosion und sind für große Systeme oftmals nicht geeignet.
Heuristische Verfahren sind deutlich abhängiger von der Struktur des Lösungsraumes, da die allermeisten Verfahren Nachbarschaftsbeziehungen innerhalb des Lösungsraumes ausnutzen, um zu einem Optimum zu gelangen. Wie oben schon dargelegt, ist es aufgrund der vielen Randbedingungen sehr schwer mittels einfacher Verfahren eine valide Lösung zu finden. Greedy-Verfahren beispielsweise nutzen lokal optimale Entscheidungen um eine Lösung zu berechnen, die aber in der Regel

[1] Wie oben motiviert, kann das Verschieben einer einzigen Task abrupte Änderungen der Energiefunktion zur Folge haben, die sich nicht in einer allgemeingültigen Beziehung formalisieren lassen.

in Lösungsräumen der oben beschriebenen zerklüfteten Struktur in die Irre führen. Tabu-Search-basierte Verfahren könnten hier eine gewisse Abhilfe schaffen, aber es ist zu erwarten, dass auch diese Ansätze massiv unter der Steilheit der Energiefunktion leiden. Gleiches gilt für Gradienten-basierte Verfahren, die sich sehr schnell in lokalen Optima festlaufen dürften.
Stochastische Verfahren sollten daher aufgrund des Einflusses zufälliger Entscheidungen eher geeignet sein, valide Lösungen für das Platzierungsproblem zu produzieren. Allerdings ist auch beispielsweise für Simulated Annealing bekannt, dass steile Flanken der Energiefunktion dem Finden der optimalen Lösung abträglich sind und für Genetische Algorithmen gilt — neben der Problematik der Informationskodierung — Ähnliches. Evolutionsstrategien scheiden aus, da die komplexe Problemstruktur in den einfachen Genen der Evolutionsstrategie nicht kodiert werden können.

Letztlich spielt auch die Größe der zu optimierenden Probleme eine wichtige Rolle in der Auswahl des Verfahrens. Wie bereits einleitend andiskutiert und in Kapitel 6 noch weiter vertieft dargestellt, gehen wir im Kontext dieser Arbeit von einer inkrementellen Integration von Teilsystemen zu einem Gesamtsystem aus. D.h. Platzierungen mit allen Freiheiten werden nur für Subsysteme berechnet, deren Größe aufgrund des Funktionsumfangs typischer Features in den betrachteten Anwendungsdomänen definiert werden kann. In der Automobilindustrie beispielsweise gehen wir von maximal 50 Tasks pro Subfunktion aus, im allgemeinen eher weniger. Für diese Systeme gelten domänenspezifische Einschränkungen, so dass zudem nicht die volle Breite der Architektur für eine Platzierungsberechnung herangezogen werden muss, sondern nur Teile davon. Insgesamt entstehen damit Problemgrößen, die einen Einsatz exakter Verfahren zumindest möglich machen. Im folgenden wird daher ein exaktes Optimierungskriterium vorgeschlagen, dass auf der einen Seite mit den skizzierten Problemgrößen umgehen kann und auf der anderen Seite unabhängig von der Struktur des Lösungsraumes ist. Gleichzeitig werden wir am Ende des Kapitels angeben, wie sich die Modellierung des Optimierungsproblemes auf andere Optimierungsverfahren, insbesondere stochastische Verfahren, übertragen lassen, falls aufgrund von Komplexität eine Anwendung des gewählten optimalen Verfahrens nicht gelingt.

Da, wie eingangs dargestellt, Branch & Bound-Verfahren und vollständige Aufzählungen nicht in Frage kommen und die Modellierung des Platzierungsproblems, wie später noch zu sehen sein wird, nicht-lineare Anteile enthält und somit ILP als Optimierungsverfahren ebenfalls ausscheidet, verbleibt als einziges Verfahren ein auf Verifikationstechniken basierender Ansatz. Im Kontext dieser Arbeit wurde hierfür das SAT-Checkingverfahren gewählt, weil zum Einen effiziente Algorithmen existieren, die auch mit beachtlichen Problemgrößen umgehen können und zum anderen als Model-Checking basierende Verfahren für die hier betrachteten Problemgrößen nur symbolische Techniken verwendet werden können, von denen aber bekannt ist, dass sie latent unter der Verwendung von Multiplikationen leiden. Diese sind aber für die Modellierung, wie noch zu sehen sein wird, unabdingbar. Folglich muss nun zunächst die Modellierung des Platzierungsproblems und des Optimierungsprozesses

als ein Erfüllbarkeitsproblem angegeben werden.

Das Problem der Platzierungswahl kann auch als ein System von Integer-Gleichungen angesehen werden, in denen die Randbedingungen, wie das Einhalten der Deadlines, Duplikate, Speicherbegrenzung und ähnliches, die Belegung der Variablen restringieren. Eine hinzugefügte Optimierungsfunktion, beispielsweise eine möglichst gleichmäßige Knoten- und Busauslastungen kann dann benutzt werden, das Optimum der Task- und Kommunikationszuweisung für eine gegebene Architektur und ein Tasknetzwerk zu bestimmen. Eine Methode, Lösungen für derartige Gleichungssysteme zu finden, ist die sogenannte SAT-Technik[122], die ausgehend von einem Gleichungssystem über booleschen Variablen versucht, eine gültige Belegung für alle Variablen zu finden. Da SAT-Checker jeweils nur *eine* unter den Randbedingungen mögliche Belegung liefern, ist bei einer Optimierung ein iteratives Verfahren notwendig. Die wesentliche Idee hierbei ist, die Optimierungsfunktion zunächst nur für einen groben Wertebereich zu definieren, das daraus entstehende Gleichungssystem auf Erfüllbarkeit zu testen und anschließend iterativ die Optimierungsfunktion zu verfeinern. Die Verfeinerung und der anschließende Erfüllbarkeitstest wird solange durchgeführt, bis eine (unendlich) kleine weitere Verfeinerung ein unlösbares Gleichungssystem erzeugt. Theoretisch läßt sich damit ein optimales Verfahren für das Problem der Task- und Nachrichtenplatzierung angeben. Praktisch wird es eine Ungenauigkeit zwischen dem Ergebnis und dem theoretisch möglichen Ergebnis geben, der von der Auflösung der Optimierungsfunktion abhängt. Trotzdem unterscheidet sich dieses Verfahren damit von den heuristisch getriebenen Verfahren, wie sie beispielsweise in Kapitel 5.1 oder aber in [171] vorgestellt wurden.

Die Umsetzung der in Kapitel 3.3 dargestellten Randbedingungen in Gleichungssysteme macht die massive Verwendung von Integer-Variablen inkl. komplexer Rechenoperationen, wie Addition und Multiplikation, unumgänglich. Da herkömmliche SAT-Checker, wie beispielsweise BerkMin[64] oder CHAFF[122] auf booleschen Gleichungen arbeiten, macht die Verwendung komplexer, arithmetischer Integer-Ausdrücke große Schwierigkeiten. Abhilfe schafft hier die Verwendung von HySAT [58], einem SAT-Checker, der um effiziente Techniken zur Behandlung von Integer-Arithmetik angereichert wurde.
Wir werden im Zuge der Generierung der Ungleichungssysteme für die hier betrachtete Problemklasse in den folgenden Abschnitten sehen, dass vielfach disjunktive und nicht-lineare Ungleichungen notwendig sind, um das Platzierungsproblem exakt zu beschreiben. Dies macht noch einmal deutlich, dass der Einsatz anderer numerischer Optimierungsverfahren, wie beispielsweise Integer Linear Programming[21], nur noch sehr begrenzt anwendbar ist (in Kapitel 5.5.3 wird dies detaillierter untersucht). Die Anwendung von SAT-basierten Techniken ist hingegen ein probates Mittel, derartige Systeme zu lösen, wobei wir massiv von den Fortschritten in der SAT Technologie[122, 206, 64] der letzten Jahre profitieren können. In Kapitel 5.6 wird deutlich werden, welche Größenordnung von Systemen mit dieser Technologie lösbar sind und wo die Grenzen dieser Technik liegen.

5.3.2 Einführung in Techniken des Erfüllbarkeitsnachweises

Bevor mit der eigentlichen Modellierung des Allokationsproblems begonnen wird, soll hier zunächst grob die Idee der als Optimierungsverfahren genutzten Technik der Erfüllbarkeitstest dargelegt werden. Die generelle Fragestellung, die ein SAT-Checking genanntes Werkzeug zu beantworten hat, ist die nach einer Belegung der Variablen einer Booleschen Funktion, die in konjunktiver Normalform gegeben ist, so dass die Funktion zu `true` evaluiert. Gibt es keine solche Belegung, so ist die Boolesche Funktion nicht erfüllbar, andernfalls liefert das Werkzeug eine entsprechende erfüllende Belegung. Die meisten modernen Implementierungen dieses Problems sind Erweiterungen auf Basis eines sehr grundlegenden rekursiven Algorithmus nach [41] (aufgrund der Kürzel der Autoren auch DPLL-Prozedur genannt): Gegeben sei eine Funktion ϕ in konjunktiver Normalform und eine partielle Belegung der in ϕ vorkommenden Booleschen Variablen ρ, die zu Beginn leer sein kann. Die DPLL-Prozedur erweitert ρ schrittweise solange, bis entweder $\rho \models \phi$ gilt oder ρ inkonsistent in Bezug zu ϕ wird. In letzterem Fall wird ein Backtracking durchgeführt, um für ρ andere Erweiterungen zu versuchen. Diese Erweiterungen werden zum einen durch logische Ableitungen aus der aktuellen Belegung ρ und der Formel ϕ erzeugt (die sogenannte *Deduce*-Phase), zum anderen werden in der sogenannten *Decision*-Phase nach bestimmten Heuristiken bisher unbelegte Boolesche Variablen ausgewählt und für diese ein belegender Wert geraten. Die DPLL-Prozedur wechselt zwischen diesen beiden Erweiterungsstrategien, wobei die Deduce-Phase durchlaufen wird, wenn immer dies möglich ist. Die Grundstruktur der DPLL-Prozedur wird anhand des Pseudo-Code in Algorithmus 1 ersichtlich und stellt im wesentlichen die Decision-Phase dar.

Algorithmus 1: Rekursive Suchprozedur nach [41]

```
DPLL(φ, ρ)
    DEDUCE(φ, ρ)
    if φ contains conflicting clause then return
    if no free variables left then exit_with(ρ)
    b := SELECT_UNASSIGNED_VARIABLE(φ)
    v := ONE_OF {true,false}
    DPLL(φ, ρ ∪ {b ↦ v})
    DPLL(φ, ρ ∪ {b ↦ ¬v})
```

Die Deduce-Phase (abgebildet in Algorithmus 2) führt eine Auflösung der sogenannten *Unit Clauses* durch. Unit Clauses sind hierbei Klauseln in ϕ, in denen alle Literale bis auf eines durch ρ mit dem Wert `false` belegt sind. Für eine solche Klausel gilt, dass die verbleibende unbelegte Boolesche Variable den Wert `true` (bzw. `false` bei negiertem Vorkommen des Literals) haben muss, wenn ϕ insgesamt erfüllbar sein soll. In diesem Sinne wird die nun neue Belegung in die gesamte Funktion übertragen und erzeugt ggf. neue Unit Clauses, aus denen ebenfalls weitere Belegungen von Booleschen Variablen abgeleitet werden können. Die Deduce-Phase wird solange durchgeführt, bis keine weiteren Belegungen abgeleitet werden können (also keine Unit Clauses mehr in ϕ unter der Belegung von ρ vorkommen).

Algorithmus 2: Deduce-Prozedur nach [41]

```
DEDUCE(φ, ρ)
    while unit clause (x) exists in φ do
        if x ≡ b for some b ∈ V then ρ := ρ ∪ {b ↦ true}
        if x ≡ b̄ for some b ∈ V then ρ := ρ ∪ {b ↦ false}
    od
```

Die Deduce-Phase kann in eine Konfliktsituation resultieren, in der eine oder mehrere Klauseln insgesamt zu `false` evaluieren. Dies bedeutet, dass die bisher gefundene Belegung ρ der Booleschen Variablen nicht zum Nachweis der Erfüllbarkeit von ϕ führt. In diesem Fall muss ein Backtracking durchgeführt werden, um eine andere Belegung zu finden. Hierzu wird die zuletzt belegte Variable aus der Decision-Phase (die Variablenbelegungen während der Deduce-Phase sind ja nur abgeleitet aus eben jener Belegung der Variablen in der Decision-Phase), die noch nicht für beide möglichen Wahrheitswerte probiert wurde, mit einem alternativen Wert versehen. Existiert keine solche Variable mehr, dann ist ϕ nicht erfüllbar. Werden andererseits alle Variablen, ohne einen Konflikt zu produzieren, belegt, so ist eine erfüllende Belegung gefunden und ϕ insgesamt erfüllbar.
Moderne Varianten dieses Basisalgorithmus verwenden zusätzliche Strategien, beispielsweise zur Auswahl der Variablen und der ihnen zugewiesenen Werte innerhalb der Decision-Phase. Siehe [110] für einen detaillierten Überblick über die Auswirkungen unterschiedlicher Heuristiken im Auswahlprozess der Variablen und Werte auf die Performanz von DPLL-basierten Entscheidungsprozeduren.

Wie nahezu alle auf Backtracking basierenden Algorithmen leidet auch das DPLL-Verfahren an dem Problem, Konflikte mehrfach zu erreichen, die aus immer dem gleichen Grund auftreten. Um dieses Problem zu vermeiden, ist man bemüht, den Grund dieser Konflikte zu speichern und damit dem Algorithmus zu erlauben, in zukünftigen Belegungen eben jene Konflikt-induzierenden Belegungen zu vermeiden. Typischerweise wird hierzu eine möglichst kleine Anzahl an Belegungen der Decision-Phase, die jeweils den Konflikt herbeiführen, in sogenannten Konfliktklauseln der Funktion ϕ hinzugefügt. Konfliktklauseln werden dabei aus der Negation der Belegung der den Konflikt auslösenden Booleschen Variablen gebildet.
Die Menge der den Konflikt auslösenden Booleschen Variablen wird mithilfe eines sogenannten Propagation Graphs ermittelt, der die kausale Beziehung von Belegungen der Variablen darstellt. Ein Cut in diesem Graph ergibt eine Begründung für alle hinter diesem Cut liegenden Belegungen und stellt somit ein Mittel zur Verfügung, aus einem Konflikt die daran beteiligten Variablen zu identifizieren. Allerdings muss diese Konfliktbegründung nicht eindeutig sein, so dass hier eine Heuristik ansetzen muss, die einen geeigneten Cut aus der Menge der Konflikt-induzierenden Cuts herausnimmt. [206] gibt hier beispielsweise einen Überblick über unterschiedliche Techniken zur Selektion geeigneter Cuts aus einem Propagation Graph.

Neben dieser Verbesserung des Basis-Algorithmus existieren noch eine Reihe weite-

rer Techniken, beispielsweise kann die Strategie des Backtracking um andere Heuristiken erweitert werden, die es ermöglichen an differenzierteren, aus dem Konflikt abgeleiteten Decision-Phasen neu anzusetzen (sogenanntes Backjumping). Andere Ansätze verwenden zufällig getriebene komplette Neustarts, die aber die gelernten Konfliktklauseln in der Funktion erhalten. Diese Ansätze sollen allerdings hier nicht weiter behandelt werden; dem interessierten Leser sei die umfangreiche Literatur zu diesem Thema empfohlen.

Mithilfe dieser Technik steht ein leistungsfähiges Instrumentarium zum Nachweis von Erfüllbarkeitseigenschaften über Booleschen Funktionen zur Verfügung, dass in der Praxis auch mit sehr großen Systemen gute Ergebnisse erzielen kann. Gleichzeitig wurde ja bereits im vorletzten Kapitel angedeutet, und dies wird in den folgenden Abschnitten zur Modellierung des Platzierungsproblems noch deutlich werden, dass eben jene Modellierung massiven Gebrauch von Integer-Variablen und Integer-Arithmetik erfordert. Dies ist a priori nicht mit einem SAT-Algorithmus gemäß DPLL-Prozedur lösbar, da dieser nur über Boolesche Funktionen Aussagen treffen kann. Hier muss also eine Transformation von Gleichungen über Integer-Variablen inklusive deren Arithmetik in einen Erfüllbarkeitstest für Boolesche Funktionen erfolgen. Das Erfüllbarkeitsnachweis-Werkzeug HySAT, welches im Kontext dieser Arbeit benutzt wird, bietet eben diese Umwandlung an und wir werden daher im folgenden Abschnitt das generelle Vorgehen innerhalb von HySAT für diese Transformation beschreiben, bevor die eigentliche Modellierung des Platzierungsproblems angegeben wird.

5.3.3 HySAT — Arithmetische Ausdrücke in SAT-Checkern

Die Behandlung der Erfüllbarkeit einer Formel ϕ über Integer-Arithmetik wird in HySAT in einem zweistufigen Verfahren in ein Entscheidungsverfahren für Boolesche Formeln überführt: Zunächst werden Hilfsvariablen eingeführt, die komplexe Strukturen arithmetischer Ausdrücke in eine sogenannte Triplet-Form transformieren. Triplet-Formen sind boolesche Kombinationen arithmetischer Ungleichungen, die höchsten 3 Variaben, höchstens einen binären Operator und genau einen relationalen Operator besitzen. Anschließend werden die arithmetischen Triplets in eine propositionale Logik übersetzt, wobei für die arithmetischen Variablen eine Zweierkomplement-Darstellung verwendet wird (dies führt nur zu einer logarithmisch wachsenden Komplexität). Angereichert um eine, entsprechend der Kodierung zu wählende, propositionale Axiomatisierung für arithmetische Operatoren, entsteht hieraus letztlich ein zu der ursprünglichen Form äquivalentes Boolesches Erfüllbarkeitsproblem.

Triplet-Transformation

Die Transformation der arithtmetischen Ungleichungen in eine Triplet-Form ist dem Einfügen von Hilfsvariablen in der Compilation von arithmetischen Ausdrücken sehr ähnlich: Im Detail gleicht diese Transformation der von Tseitin[188] vorgeschlagenen Linear-Time Transformation propositionaler Formeln in eine konjunktive Normalform. Die Idee dabei ist, eine komplette Formel ϕ in eine Formel $[\phi] \wedge T(\phi)$ zu

übersetzen, wobei $[\phi]$ eine neu eingefügte, propositionale Variable ist, die den Wahrheitswert von ϕ repräsentiert. $T(\cdot)$ ist dabei folgendermaßen definiert:

$$T(\phi \odot \psi) \quad = \quad ([\phi \odot \psi] \Leftrightarrow ([\psi] \odot [\psi])) \wedge T(\phi) \wedge T(\psi),$$

wobei $\odot$ beliebige Boolesche Junktoren und $[\phi]$ bzw. $[\psi]$ wiederum neu eingefügte propositionale Variablen sind. Gleichzeitig gilt:

$$T(e_1 \sim e_2) \quad = \quad ([e_1 \sim e_2] \Leftrightarrow [e_1] \sim [e_2]) \wedge T(e_1) \wedge T(e_2) \tag{5.1}$$

$$T(e_1 \otimes e_2) \quad = \quad ([e_1 \otimes e_2] = [e_1] \otimes [e_2]) \wedge T(e_1) \wedge T(e_2) \tag{5.2}$$

$$T(v) \quad = \quad v \qquad \text{wenn } v \text{ eine Variable ist} \tag{5.3}$$

wobei $\sim$ ein beliebiger relationaler Operator, $\otimes$ ein beliebiger arithmetischer Operator, $[e_1 \sim e_2]$ eine neu eingefügte propositionale Variable, sowie $[e_i]$ und $[e_1 \otimes e_2]$ eine neu eingefügte arithmetische Variable mit aus den Ranges der Sub-Ausdrücke abgeleiteten Ranges sind. Diese Umformung führt zu einer äquivalent erfüllbaren Formel: Die erfüllbare Belegung der ursprünglichen Formel kann mittels einer Projektion der Variablen der Triplet-Form aus der erfüllbaren Belegung der Variablen der Triplet-Form abgeleitet werden.

Propositionale Kodierung

Nach der Transformation in eine Triplet-Form müssen nun die arithmetischen Triplets, also diejenigen, die durch (5.1) und (5.2) entstanden sind, in propositionaler Logik kodiert werden. Dies geschieht — gemäß der gewählten Zweierkomplement-Darstellung — durch die Ersetzung der arithmetischen Variablen durch eine entsprechende Anzahl an propositionalen Variablen. Die Operatoren werden mittels einer Axiomatisierung dargestellt, die ihr Verhalten auf Bitvektoren widerspiegelt. Beispielsweise kann ein Triplet, dass eine Addition mit zwei Variablen durchführt, auf die folgende Art und Weise transformiert werden:

$$\mathcal{P}(x + y = z) \quad = \quad \bigwedge_{i=0}^{k} \text{fulladd}(z_i, c_{i+1}, x_i, y_i, c_i) \wedge \neg c_0 \wedge (z_{k+1} \Leftrightarrow c_{k+1}) \ ,$$

wobei k die Anzahl der Bits ist, die für eine Darstellung der Werte von x und y benötigt werden und c_0 bis c_{k+1} neu eingeführte propositionale Variablen sind. Die Funktion `fulladd` ist definiert durch:

$$\text{fulladd}(s, c_{\text{out}}, x, y, c_{\text{in}}) \quad = \quad (s \Leftrightarrow (x \dot{\vee} y \dot{\vee} c_{\text{in}})) \wedge (c_{\text{out}} \Leftrightarrow (x + y + c_{\text{in}} \geq 2)) \tag{5.4}$$

Die Vorgehensweise entspricht also der aus der Hardware-Synthese bekannten Synthese von Operationen auf Bitvektoren aus Grundobjekten, wie beispielsweise Volladdier, Halbaddierer, etc. Abbildung 5.2 verdeutlicht dieses Vorgehen nochmals bildlich.

Mithilfe dieser grundlegenden Technik kann nun neben Addierern eine Vielzahl weiterer arithmetischer Funktionen in eine Boolesche Darstellung abgebildet werden,

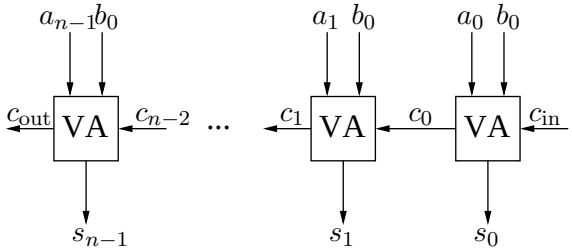


Abbildung 5.2: Transformationen einer Additions-Operation mit aus der Hardware-Synthese bekannten Verfahren unter Verwendung der Grundelemente Volladdierer.

beispielsweise die Multiplikation oder auch die Division. Für eine detaillierte Beschreibung der Transformation arithmetischer Operationen sei der interessierte Leser auf [111] verwiesen.

Abschließend müssen die propositionalen Formeln, die durch eine Transformation von $\mathcal{P}$ entstanden sind, in die Eingabesprache eines geeigneten SAT-Solvers übersetzt werden. Um eine möglichst kompakte Kodierung zu erreichen wird in HySAT, im Gegensatz zu herkömmlichen Solvern, weder eine Kodierung in konjunktiver Normalform verwendet, noch erfolgt ein Aufbau mittels binärer Boolescher Operatoren. Vielmehr versucht HySAT die bekannten Vorteile von pseudo-Booleschen Formeln[11] auszunutzen. Eine pseudo-Boolesche Formel ist eine Konjunktion linearer Constraints über Booleschen Literalen, die es beispielsweise erlaubt, die rechteste Konjunktion der Formel (5.4) durch $(2\overline{c_{\text{out}}} + x + y + c_{\text{in}} \geq 2) \wedge (2c_{\text{out}} + \overline{x} + \overline{y} + \overline{c_{\text{in}}} \geq 2)$ darzustellen, wobei $\overline{v}$ das Komplement von v ist. Die hierdurch entstehenden pseudo-Booleschen Formeln werden mittels des pseudo-Booleschen Constraint Solvers GOBLIN[57] gelöst (GOBLIN selbst ist der Kern des HySAT[58] Solvers).
Die entstandene Formel is erfüllbar, genau dann wenn das kodierte Integer-arithmetische Constraint-System ϕ erfüllbar ist. Jede erfüllbare Belegung ist außerdem eine Repräsentation der Zweierkomplement-Darstellung von ϕ's Triplet-Form. D.h. eine erfüllbare Belegung von ϕ kann mittels rückgängigmachen der Zweierkomplement-Bildung und anschließender Projektion auf ϕ's ursprüngliche Variablen extrahiert werden.
Damit steht insgesamt ein leistungsstarkes Instrumentarium zur Verfügung, große Systeme von integer-arithmetischen Ungleichungen auf Erfüllbarkeit zu untersuchen, die auch durchaus nicht-lineare Anteile aufweisen können. Für weitergehende Informationen über die internen Abläufe innerhalb des verwendeten Werkzeuges HySAT sei hier auf [73] verwiesen. Die Art und Weise, wie wir diese Technik zur Optimierung und Bestimmung von Platzierungen einsetzen, wird ausführlich in Kapitel 5.4 beschrieben. Hier soll zunächst die Erzeugung einer Integer-arithmetischen Darstellung des Platzierungsproblems betrachtet werden.

5.3.4 Platzierungsprobleme als arithmetische Ungleichungen

Im folgenden soll die Platzierungsabbildung Π in Form von Integer-Ungleichungen modelliert werden. Dazu wird eine Menge von Integer-Variablen definiert, die Platzierung und Zeitverhalten charakterisieren und aufgrund von restringierenden Un-

gleichungen den SAT-Checker so steuern, dass nur eine gültige Lösung erzeugt wird. Für jeden Task wird eine Platzierungsvariable $\texttt{place}_i \in P$ eingeführt, deren Belegung festlegt, auf welchen Knoten der Architektur $p \in P$ die Task platziert wird. Damit wird eine logarithmische Kodierung der Platzierungsinformation erreicht, mit der auf einfache Art und Weise Duplikate und verbotene Knoten modelliert werden können. Hierfür werden die folgende Konjunktionen zu dem zu lösenden Gleichungssystem hinzugefügt:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P \setminus \pi_i} \texttt{place}_i \neq p \tag{5.5}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T} : \delta_i \neq \emptyset} \bigwedge_{\forall \tau_j \in \delta_i} \texttt{place}_i \neq \texttt{place}_j \tag{5.6}$$

Grundlegende Basis für die Überprüfung der Gültigkeit einer Platzierung Π ist die Einhaltung der Deadlines, die mittels des in Gleichung 4.3 angegebenen Verfahrens analysiert werden. Wie schon in Kapitel 3.3 ausgeführt, sind die dafür notwendigen WCETs der Tasks abhängig von der Platzierung. Die jeweilige WCET eines platzierten Tasks wird über die folgenden Konjunktionen bestimmt und in der Integer-Variable $\texttt{wcet}_i$ vermerkt.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\texttt{place}_i = p) \rightarrow \texttt{wcet}_i = c_i(p) \tag{5.7}$$

Die in Abschnitt 4.2 eingeführten synthetischen Deadlines der Tasks sind gemäß Gleichung 4.4 um die Deadline der Kommunikationen zu erweitern, wenn alle Kommunikationspartner auf dem selben Knoten platziert wurden. Diese, abhängig von der Platzierung, justierte Deadline ist nicht nur für den Verteilungsspielraum notwendig, sondern dient auch der Prioritätenbestimmung der Tasks untereinander, wenn Prioritätenschemata wie beispielsweise Deadline Monotonic[100] verwendet werden. Gleichung 5.8 führt die Justierung bei in sich zusammenfallenden Kommunikationen durch, während Gleichung 5.9 die für den Task synthetisierte Deadline erhält.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \Big(\bigwedge_{\forall g_j = (\tau_j, \cdot, \cdot) \in \gamma_i} \texttt{place}_j = \texttt{place}_i \Big) \rightarrow \texttt{d}_i = \Delta_\tau(\tau_i) + \min_{\forall g_l \in \gamma_i} (\Delta_\gamma(g_l)) \tag{5.8}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \neg \Big(\bigwedge_{\forall g_j = (\tau_j, \cdot, \cdot) \in \gamma_i} \texttt{place}_j = \texttt{place}_i \Big) \rightarrow \texttt{d}_i = \Delta_\tau(\tau_i) \tag{5.9}$$

Mithilfe der bisher formulierten Gleichungen kann nun die Antwortzeitberechnung nach Gleichung 4.3 durchgeführt werden. Die Bestimmung des jeweiligen Interferenzfaktors hängt von den Prioritäten der Tasks ab, die hier nach dem Deadline Monotonic Prinzip vergeben werden[1].

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \texttt{r}_i = \texttt{wcet}_i + \sum_{\forall \tau_j \in \mathcal{T}} \texttt{interf}_i^j \tag{5.10}$$

[1] Baruah und Goossens haben in [12] gezeigt, dass für die hier betrachteten Systeme mit $d_i \leq t_i$ diese Strategie optimal ist.

Allerdings sind aufgrund der Deadlinejustierung gemäß Gleichung 5.8 die Deadlines von Tasks keine konstanten Werte, sondern können sich von betrachteter Platzierung zu Platzierung verändern. Es werden daher alle prinzipiell möglichen Interferenzen mittels der Variablen $\texttt{interf}_i^j$ addiert, die im Fall einer höheren Priorisierung von Task τ_i bzgl. Task τ_j auf 0 gezwungen werden.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \left(\mathtt{p}_i^j \wedge (\mathtt{place}_i = \mathtt{place}_j)\right) \rightarrow \mathtt{interf}_i^j = \mathtt{I}_i^j \cdot \mathtt{wcet}_j \tag{5.11}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \neg\left(\mathtt{p}_i^j \wedge (\mathtt{place}_i = \mathtt{place}_j)\right) \rightarrow \mathtt{interf}_i^j = 0 \tag{5.12}$$

Die Variablen $\mathtt{p}_i^j$ stellen die Prioritätenbeziehung der Tasks τ_i und τ_j dar und werden gemäß der Deadline Monotonic Strategie[100] vergeben. Aufgrund ihrer Dynamik werden sie durch die folgenden Gleichungen eingeschränkt:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \mathtt{p}_i^j \geq 0 \wedge \mathtt{p}_i^j \leq 1 \wedge \mathtt{p}_i^j + \mathtt{p}_j^i = 1 \tag{5.13}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{d}_i > \mathtt{d}_j) \rightarrow \mathtt{p}_i^j = 1 \tag{5.14}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{d}_i < \mathtt{d}_j) \rightarrow \mathtt{p}_i^j = 0 \tag{5.15}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{d}_i = \mathtt{d}_j) \rightarrow \mathtt{p}_i^j \leq 1 \tag{5.16}$$

Aufgrund der Möglichkeit, dass Deadlines verschiedener Tasks den gleichen Wert haben können, muss zusätzlich für diese Tasks eine Prioritätenordnung untereinander gefunden werden[1]. Dies wird durch Gleichung 5.16 erreicht, die dafür sorgt, dass der Wert von $\mathtt{p}_i^j$ bei Gleichheit der Deadlines frei wählbar ist[2] und damit eine bzgl. des Optimalitätskriteriums gute Belegung gefunden werden kann.
Sind die Deadlines zur Generierungszeit bereits fest vorgegeben und können nicht innerhalb des Optimierungsverfahrens verändert werden (beispielsweise durch die Hinzunahme von Deadlines von Nachrichten, die nur lokal auf dem Knoten versendet werden), so kann die Priorisierung bereits bei der Erstellung der Gleichungen erfolgen und muss nicht vom Optimierungsverfahren ermittelt werden. Prioritätenschemata gemäß Deadline monotoner Strategie sind bekanntlich optimal, wenn sich alle Deadlines unterscheiden und zusätzlich die Periode der Tasks immer größer oder

[1] Gleiche Deadlines bedeuten im Falle des Deadline Monotonic Ansatzes, dass die Priorität ebenfalls gleich ist. Das damit intendierte Verhalten bzgl. einer Worst-Case Analyse, wie der Antwortzeitberechnung, ist nicht klar erkennbar, da nicht vorherbestimmt werden kann, welche Task welche unterbricht. Um diesem vorzubeugen, müsste folglich die Analyse davon ausgehen, dass Tasks mit der gleichen Priorität sich wechselseitig unterbrechen. Dies entspricht jedoch keinem realen Verhalten und ist deswegen eine zu grobe Überapproximation.

[2] Streng genommen kann auf die Gleichung 5.16 verzichtet werden, da im Falle der Deadlinegleichheit nur Gleichung 5.13 greift und dieses Verhalten subsumiert und gleichzeitig die Bedingung $\mathtt{p}_i^j + \mathtt{p}_j^i = 1$ eine Eindeutigkeit der Prioritäten sicherstellt.

gleich ihrer Deadline ist. Treten gleiche Deadlines auf, so ist zunächst nicht klar, wie eine Priorisierung zu schaffen ist, die ebenfalls optimal ist. Hierzu dient das folgende Lemma.

Lemma 5.3.1 (Optimalität bei Deadline-Gleichheit). *Sei ein Tasksystem $\mathcal{T}$ mit $d_i \leq p_i$ für alle Tasks $\tau_i \in \mathcal{T}$ gegeben, für das gilt:*

- *Es existieren 2 Tasks τ_i und τ_j mit $d_i = d_j$*
- *Sei $\mathcal{T}$ priorisiert gemäß Deadline monotoner Prioritätenvergabe, wobei τ_i und τ_j eine feste aber beliebige Priorisierung zueinander haben*
- *Sei $\mathcal{T}$ feasible*

Dann ist $\mathcal{T}$ ebenfalls feasible, wenn die Prioritäten von τ_i und τ_j vertauscht werden.

Beweis: siehe Anhang C.

Insbesondere bei Deadlines von Nachrichten kann dieses Verfahren gewinnbringend eingesetzt werden, da diese nicht von der Platzierung abhängig sind. Laut Lemma 5.3.1 genügt eine einfache Sortierung der Nachrichten bzgl. ihrer Deadline, um die Priorität zu bestimmen. Für Deadlines gleicher Größe ist es mithin unerheblich, in welcher Prioritätsrelation sie zueinander stehen.

Interferenzkosten gemäß Gleichung 5.11 und 5.12 werden mittels der Anzahl der Preemptionen durch die Variable $\mathtt{I}_i^j$ und der jeweiligen WCET bestimmt.
Die durch die Deadline-Synthese geschaffene Möglichkeit der lokalen Analyse kann nun ausgenutzt werden, um die gegebenen End-to-End-Deadlines zu überprüfen.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{r}_i \leq \mathtt{d}_i \tag{5.17}$$

Interferenzen durch höher priorisierte Tasks sind jeweils paarweise zu betrachten und hängen von der Platzierung der jeweiligen Tasks ab, da nicht auf demselben Knoten platzierte Tasks keine Interferenzen auslösen können. Allgemein muss hier eine Umsetzung der Interferenzberechnung aus Gleichung 4.3 durchgeführt werden, also ein Faktor

$$\left\lceil \frac{r_i + J_j}{t_j} \right\rceil \cdot c_j := \mathtt{I}_i^j \cdot c_j$$

Dies ist auf Integer-Gleichungen nicht so ohne weiteres zu übertragen, so dass der Wert durch ein System von Ungleichungen erzwungen werden muss. Dazu wird eine Grenzwertbetrachtung unter Berücksichtigung der Definition der Ceiling-Funktion durchgeführt. Es gilt dann:

$$\frac{r_i + J_j}{t_j} \leq \left\lceil \frac{r_i + J_j}{t_j} \right\rceil < \frac{r_i + J_j}{t_j} + 1$$

Beide Teile der obigen Ungleichung können nun mithilfe der eingefügten Integer-Variablen $\mathtt{I}_i^j$ so aufgelöst werden, dass sie den Wertebereich dieser Variablen ein-

grenzen und als erlaubte Belegung nur den Wert des Ceiling-Ausdrucks erlauben.

$$\frac{r_i + J_j}{t_j} \leq \mathtt{I}_i^j \quad \Leftrightarrow \quad r_i + J_j \leq \mathtt{I}_i^j \cdot t_j \qquad \text{(a)}$$

$$\mathtt{I}_i^j < \frac{r_i + J_j}{t_j} + 1 \quad \Leftrightarrow \quad r_i + J_j > (\mathtt{I}_i^j - 1) \cdot t_j \qquad \text{(b)}$$

Die Kombination der beiden Ungleichungen (a) und (b) für $\mathtt{I}_i^j$ führt zu einer korrekten Belegung bzgl. der Ceiling-Funktion, weil die Werte der möglichen Belegungen von $\mathtt{I}_i^j$ nur ganzzahlig sein können. Da die Werte für den Jitter von Tasks und Nachrichten durch das in Kapitel 4.4 dargestellte Verfahren überapproximiert werden, handelt es sich bei ihnen um Konstanten, die zur Systemerstellung bekannt sind und nicht durch Ungleichungssysteme erzeugt werden müssen. Dies gilt insbesondere auch für ihre Unabhängigkeit von den justierten Deadlines aufgrund von in sich zusammenfallenden Kommunikationen, da die Überapproximation gemäß Definition 4.4.1 sowohl die Deadlines der Tasks als auch der Kommunikationen enthalten. Liegen die interferierenden Tasks nicht auf demselben Knoten, so erzeugt eine weitere Bedingung einen Wert von 0 für die Interferenz.
In der formalisierten Darstellung von Integer-Ungleichungen sieht die Interferenzbestimmung aller Taskpaare, die potentiell eine Interferenz aufweisen könnten[1], wie folgt aus:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{place}_i = \mathtt{place}_j) \rightarrow ((\mathtt{I}_i^j \cdot t_j \geq \mathtt{r}_i + J_j) \wedge ((\mathtt{I}_i^j - 1) \cdot t_j < \mathtt{r}_i + J_j)) \quad (5.18)$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{place}_i \neq \mathtt{place}_j) \rightarrow \mathtt{I}_i^j = 0 \quad (5.19)$$

Das Verfahren kann die $\mathtt{I}_i^j$ Variablen mit beliebigen Werten belegen, auch wenn keine Interferenz von τ_i durch τ_j auftreten kann, weil sie auf unterschiedlichen Knoten platziert werden. Die jeweiligen $\mathtt{interf}_i^j$ Variablen übernehmen in diesem Fall das Wegschalten der Interferenzkosten (vgl. Gleichung 5.12).
Bei der Berechnung einer Platzierung muss zudem noch auf den Speicherbedarf der Tasks auf dem jeweiligen Knoten geachtet werden (vgl. Definition 4.1.3(1)). Dieser wird für jedes Task-Prozessor-Paar (p, τ_i) in einer Variablen $\mathtt{m}_p^i$ berechnet und für jeden Knoten gemäß der Platzierung addiert:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\mathtt{place}_i = p) \rightarrow \mathtt{m}_p^i = \mu_i^{\mathcal{T}}(p) \quad (5.20)$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\mathtt{place}_i \neq p) \rightarrow \mathtt{m}_p^i = 0 \quad (5.21)$$

$$\bigwedge_{\forall p \in P} \mu^{\mathcal{A}}(p) - \sum_{\forall \tau_i \in \mathcal{T}} \mathtt{m}_p^i \geq 0 \quad (5.22)$$

[1] Dies ist an die Priorität der Tasks gebunden, so dass nur für solche Taskpaare diese Ungleichung eingeführt werden muss, die auch tatsächlich aufgrund der Prioritäten potentielle eine Interferenz hervorrufen. Wie bei der Antwortzeitanalyse dargestellt, sind diese Prioritäten gemäß Deadline Monotonic vergeben, wobei die Deadlines allerdings von Platzierung zu Platzierung differieren können.

Zwecks Reduktion der Anzahl der Gleichungen kann an dieser Stelle die Information über erlaubte Platzierungen ausgenutzt werden, wie sie im Attribut π_i einer Taskdefinition τ_i gegeben ist. Betrachtet werden demzufolge nur diejenigen (Task,Knoten)-Paare, die auch erlaubt sind; alle nicht erlaubten Paarungen können in Gleichung 5.22 ignoriert werden bzw. produzieren einen Wert von 0 für die spezifische $\mathtt{m}_p^i$-Variable.

5.3.5 Pfadsuche für Kommunikationen

Mit der Platzierung einer kommunizierenden Task τ_i mit $\gamma_i \neq \emptyset$ ist gleichzeitig eine Abbildung der Nachrichten auf die Kommunikationsmedien der Architektur vorzunehmen. Heterogene Systeme, wie beispielsweise in Abbildung 5.3 dargestellt, verlangen dabei ein komplexes Routing der Nachrichten vom Sendeknoten zum Empfangsknoten. Dabei kann mitunter die sequentielle Nutzung mehrerer Netzwerke notwendig sein. Die Abbildung $\Gamma(\tau_i, g_j)$ für $\tau_i \xrightarrow{g_j} \tau_j$ aus Definition **??** muss folglich gefunden werden, um einen Pfad auf Basis der Topologie der Architektur zu generieren, der gültig ist und der zudem nicht deadline-verletzend auf das System einwirkt.

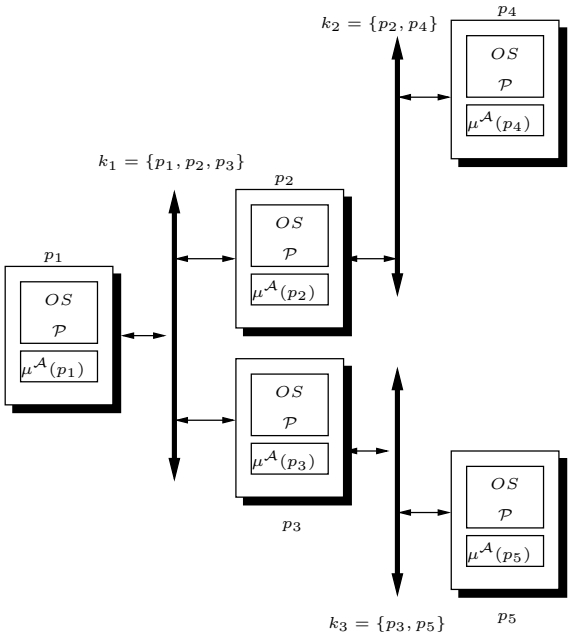


Abbildung 5.3: Beispiel einer heterogenen verteilten Steuergerätearchitektur mit komplexer Topologie.

Für jede Nachricht $g_j \in \gamma_i$ einer Task τ_i wird die Abbildung $\Gamma(\tau_i, g_j)$ festgelegt durch eine Menge boolescher Variablen $\mathtt{K}_{g_j}^f$ mit $f \in \{1, \ldots, |K|\}$. Eine Belegung von 1 auf einer Variablen $\mathtt{K}_{g_j}^f$ bedeutet, dass die Kommunikation g_j über das Kommunikationsmedium k_f übertragen werden muss.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_j \in \gamma_i} (\mathtt{place}_i \neq \mathtt{place}_j) \rightarrow \bigvee_{\forall k \in K} \mathtt{K}_{g_j}^k \tag{5.23}$$

Liegt keine Kommunikation aufgrund gleicher Platzierungen der Sender- und Empfängertasks vor, muss dafür gesorgt werden, dass $\Gamma(\tau_i, g_j) = \emptyset$ liefert.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_j \in \gamma_i} (\texttt{place}_i = \texttt{place}_j) \rightarrow \bigwedge_{\forall k \in K} \neg \texttt{K}_{g_j}^k \tag{5.24}$$

Diese Information reicht aber alleine nicht aus, da keine Ordnungen der Belegungsvariablen $\texttt{K}_{g_j}^f$ vorliegt. Sublokale Jitterberechnungen, wie sie in Kapitel 4.4.2 vorgestellt wurden, sind aber auf eine Ordnung angewiesen, da die Nachrichten die Kommunikationsmedien in einer festen, durch die Platzierung und die Topologie vorgegebenen, Reihenfolge durchlaufen und der sublokale Jitter demzufolge sukzessive aufgebaut wird. Im folgenden werden sogenannte Pfadhüllen verwendet, um die Ordnung auf dem Kommunikationsweg einer Nachricht darzustellen. Gleichzeitig sichern diese Pfadhüllen durch ihre Konstruktion die Gültigkeit des Pfades $\texttt{K}_{g_j}^f$ ab.

Ausgehend von einem Kommunikationsmedium können zunächst die maximalen zyklenfreien Pfade durch die Topologie der Architektur erzeugt werden. Dabei werden nur die Kommunikationsmedien selbst betrachtet, nicht aber die Knoten der Systemarchitektur[1]. Die maximalen Pfade werden durch die folgende Definition 5.3.1 in einer Menge aus Worten über der Sprache $K^{\leq |K|}$ abgebildet:

Definition 5.3.1 (Menge der maximalen Pfade). *Die Menge der maximalen Pfade* $MP(k_1) = \{w_{k_1} | w_{k_1} \in K^{\leq |K|}\}$ *ausgehend von einem Kommunikationsmedium* k_1 *ist gegeben durch:*

$$MP(k_1) = \Big\{ w_{k_1} | \quad w_{k_1} = \text{“}k_1 k_2 \dots k_n\text{“} \wedge \forall 1 \leq i, j \leq n : k_i \neq k_j \\ \wedge \forall 1 \leq i < n : k_i \cap k_{i+1} \neq \emptyset \\ \wedge \nexists k_{n+1} : k_n \cap k_{n+1} \neq \emptyset \wedge \forall 1 \leq i \leq n : k_{n+1} \neq k_i \Big\}$$

Mithilfe dieser Definition werden jeweils die längsten Pfade ausgehend von einem Kommunikationsmedium als Worte über der Sprache $K^{\leq |K|}$ aufgebaut, ohne das es

a) Zyklen innerhalb des Pfades gibt

b) unnötige Mehrfachbenutzung eines Kommikationsmediums gibt

Zyklen sind aufgrund von Terminierungseigenschaften zu vermeiden, da das Optimierungsverfahren in diesem Fall beliebig lange Wege durch die Systemtopologie für eine Nachricht konstruieren könnte. Zwar würde ein derartiges Vorgehen keinem Optimierungskriterium genügen und damit im Zuge der Verteilunsgberechnung wegoptimiert werden, allerdings bergen Zyklen die Gefahr des Nicht-Terminierens der Berechnungsvorschriften zu Antwortzeit, Jitter und ähnlichem.
Unnötige Mehrfachbenutzungen hingegen sind prinzipiell kein Problem, solange sie nicht zu Zyklen führen, sollten jedoch aus Komplexitätsgründen vermieden werden. Zudem gibt es kein sinnvolles Szenario, dass den Einsatz von Mehrfachnutzung notwendig erscheinen lässt. Ein Beispiel für eine Mehrfachnutzung ist in Abbildung 5.3

[1] Kommunikationsmedien stellen Mengen von durch das Medium verbundenen Knoten dar, so dass die Knoteninformation implizit berücksichtigt wird.

beispielsweise eine Kommunikationssequenz $p_5 \xrightarrow{k_3} p_3 \xrightarrow{k_1} p_1 \xrightarrow{k_1} p_2 \xrightarrow{k_2} p_4$, die zu einem maximalen Pfad von "$k_3k_1k_1k_2$" $\in MP(k_3)$ führt. Letztlich würde, abhängig vom Optimierungskriterium, diese Wegewahl aussortiert werden, so dass diese Lösungen ohne Einschränkungen der Platzierungsentscheidung ignoriert werden können.
Für das in Abbildung 5.3 gezeigte Beispiel ergeben sich demnach die folgenden maximalen Pfade:

$$MP(k_1) = \{\text{“}k_1k_2\text{“}, \text{“}k_1k_3\text{“}\}$$

$$MP(k_2) = \{\text{“}k_2k_1k_3\text{“}\}$$

$$MP(k_3) = \{\text{“}k_3k_1k_2\text{“}\}$$

Zur besseren Handhabung der Elemente der Menge der maximalen Pfade sollen nun zunächst drei Abbildungen definiert werden, die Worte über der Sprache $K^{\leq|K|}$ in kleiner Fragmente zerlegen können:

Definition 5.3.2 (Projektion, Präfix und Position). *Sei ein Wort w_k der Sprache $K^{\leq|K|}$ gegeben. Dann sei*

- *(Projektion) $pr : \mathbb{N}\times\mathrm{K}^{\leq|\mathrm{K}|} \rightarrow \mathrm{K}$ gegeben durch $pr(i, w_k) = k_i, \forall w_k =$ “$k_1 \ldots k_n$“$\wedge$ $1 \leq i \leq n$. Schreibweise für $pr(i, w_k)$ sei $pr_i(w_k)$.*
- *(Präfix) die Menge aller Präfixe eines Wortes $pre : K^{\leq|K|} \rightarrow 2^{K^{\leq|K|}}$ gegeben durch $pre($“$k_1k_2 \ldots k_n$“$) = \{$“k_1“, “k_1k_2“, $\ldots$, “$k_1k_2 \ldots k_n$“$\}$*
- *(Position) $pos : K \times K^{\leq|K|} \rightarrow \mathbb{N}$ die Position eines Kommunikationsmedium k_i im Wort w_k gegeben durch $pos(k, w_k) = j \mid pr_j(w_k) = k$*

Insbesondere durch Erzeugung von Präfixen $pre(w_k)$ kann nun ausgehend von den maximalen Pfaden $MP(k_j)$ die Menge aller möglichen Pfade in der gegebenen Systemtopologie generiert werden, die sogenannten Pfadhüllen. Elemente der Menge der Pfadhüllen selbst sind Mengen, die durch die Präfix-Bildung der maximalen Pfade gewonnen werden und geben damit kausal zusammenhängende Kommunikationsreihenfolgen, also die Ordnung der Medien auf einem möglichen Pfad, vor.

Definition 5.3.3 (Pfadhüllen). *Sei die Menge aller maximalen Pfade auf der gegebenen Systemtopologie gegeben durch*

$$\breve{MP} = \bigcup_{1\leq j\leq|K|} MP(k_j).$$

Dann ist die Menge der Pfadhüllen PH definiert durch

$$PH = \left\{ph_i \in 2^{K^{\leq|K|}} \mid \forall mp_i \in \breve{MP} \wedge 1 \leq i \leq |\breve{MP}| : ph_i = pre(mp_i)\right\} \cup \{\emptyset\}$$

Pfadhüllen fassen folglich alle von einem Knoten ausgehenden möglichen Pfade in einer Menge zusammen, womit aufgrund der topologischen Information die Ordnung der Kommunikationsmedien bzgl. eines Pfades in einer Pfadhülle erhalten bleibt und dem Gleichungssystem der Platzierung zugänglich ist. Zu diesem Zweck wird den

Belegungsvariablen $\mathtt{K}^f_{g_j}$ zusätzlich noch eine Pfadvariable für jede Nachricht $\mathtt{Pf}_{g_j}$ zugeordnet. Die leere Pfadhülle $\{\emptyset\}$ bildet die Situation ab, in der keine Nachricht über ein Kommunikationsmedium verschickt werden muss, weil sowohl Sende- als auch Empfangstask auf dem selben Knoten allokiert sind und alle $\mathtt{K}^f_{g_j}$ auf False gesetzt werden müssen. Abbildung 5.4 zeigt die Konstruktion der Pfadhüllen exemplarisch an der in Abbildung 5.3 dargestellten Architektur.

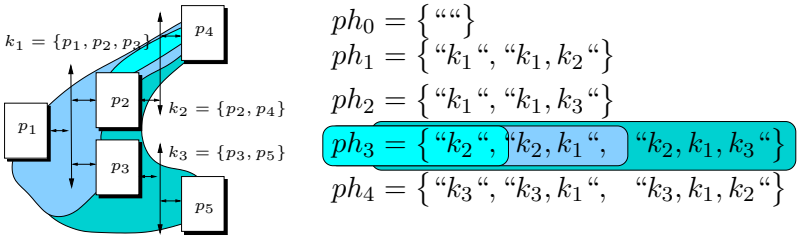


Abbildung 5.4: Pfadhüllen über einer heterogenen Topologie inklusive der totalen Ordnung einer Pfadhülle, gegeben durch die jeweiligen Wörter über der Menge der Kommunikationsmedien.

Vorteil dieser Kodierung der Platzierung von Kommunikation ist die statische Analysierbarkeit des Systems: Bei der Erstellung des Gleichungssystems für die Platzierung kann bereits das Wissen über die Topologie ausgenutzt werden und es können speziell an die gegebene Architektur angepasste Gleichungen für die Kommunikationsplatzierung erzeugt werden.
Topologische Einschränkungen bei der Wahl der Kommunikationsmedien für das Routing einer Nachricht können nun mithilfe der Pfadhüllen dargestellt werden. Die Wahl einer bestimmten Pfadhülle grenzt gleichzeitig die möglichen Belegungen der $\mathtt{K}^f_{g_j}$ Variablen auf einen für die jeweilige Pfadhülle erlaubten Bereich ein. Letztlich muss natürlich die Gültigkeit der Wahl des Kommunikationspfades gewährleistet werden, d.h. die die Nachricht sendende Task τ_i muss auf einem Knoten am ersten Kommunikationsmedium platziert sein und die die Nachricht empfangende Task τ_j muss auf einem Knoten am letzten Kommunikationsmedium angesiedelt sein. Formal bedeutet dies den in Gleichung 5.25 angegebenen Satz an Gleichungen, der sich auf die Anordnung der einzelnen Kommunikationsmedien eines Pfades der Pfadhülle abstützt. Dazu sei zunächst für jeden Pfad $p \in ph_i$ einer Pfadhülle aus der Menge der Pfadhüllen $ph_i \in PH$ die Menge der komplementären Kommunikationsmedien $\bar{p} \subseteq K$ definiert durch

$$\forall p \in ph_i : \bar{p} = K \backslash \left\{ k_j \mid \forall 1 \leq j \leq |p| : k_j = pr_j(p) \right\}$$

Die Menge der zu einem Pfad p komplementären Kommunikationsmedien wird nun benutzt, um in einem Satz von Ungleichungen die Belegung der Variablen für auf

diesem Pfad erlaubten Kommunikationsmedien einzuschränken.

$$\bigwedge_{\forall ph_l \in PH} \bigwedge_{\forall \tau_i \in \mathcal{T}:\gamma_i \neq \emptyset} \bigwedge_{\forall g_j \in \gamma_i} (\mathtt{Pf}_{g_j} = ph_l) \rightarrow \bigvee_{\forall p \in ph_l} \left(\bigwedge_{k \in \bar{p}} \neg \mathtt{K}^k_{g_j} \bigwedge_{\forall 1 \leq l \leq |p| : k_l = pr_l(p)} \mathtt{K}^{k_l}_{g_j} \wedge \ valid\_place(p) \right) \tag{5.25}$$

Anschaulich bedeutet die Formulierung in Gleichung 5.25, dass für eine Nachricht, die einer Pfadhülle ph_l zugeordnet wurde, nur diejenigen Medien benutzt werden dürfen, die in der Pfadhülle möglich sind. Alle anderen werden durch Gleichung 5.25 auf 0 gezwungen. Die disjunktive Verknüpfung liefert die Unterscheidung der verschiedenen möglichen Pfade innerhalb der Pfadhülle ph_l. Gleichzeitig wird überprüft, ob Startmedium und Zielmedium jeweils den aus der Platzierung gewonnenen Knoten beinhalten.
valid_place sorgt dafür, dass Sende- und Zieltask einer Nachricht entsprechend dem angegebenen Pfad p eine gültige Platzierung aufweisen. Dabei ist zu unterscheiden, ob $|p| = 1$ oder $|p| > 1$. Diese Unterscheidung dient der Vermeidung von Platzierung auf Gateway-Knoten zwischen zwei Kommunikationsmedien: Wird ein Pfad bestehend aus 2 Kommunikationsmedien betrachtet und ist die sendende Task auf dem Gateway-Knoten zwischen diesen Medien platziert, so würde die Nachricht an die Empfängertask zunächst vom Knoten der Sendertasks über das erste Kommunikationsmedium zum Gateway-Knoten verschickt werden. Letzterer ist aber wiederum der Knoten der Sendertask, so dass ein wenig sinnvoller Zyklus im Pfad der Nachricht entsteht. Um diese Platzierungen zu vermeiden, sind daher bei Pfaden mit mehr als einem beteiligten Kommunikationsmedium die Gateway-Knoten zwischen zwei im betrachteten Pfad benachbarten Kommunikationsmedien für eine gültige Platzierung verboten[1]. Somit ergibt sich durch diese Einschränkung in der Analyse der Pfadhüllen keine generelle Einschränkung bei der Platzierungsfreiheit, sondern nur eine Einschränkung auf sinnvolle Pfade durch die Architektur. Formal bedeutet dies für gültige Platzierungen und für eine Nachricht $\tau_i \xrightarrow{g_j} \tau_j$ die folgende Ersetzung von *valid_place* in Gleichung 5.25[2]:

$$valid\_place(p) := \begin{cases} \Pi(\tau_i) \in k_r \wedge \Pi(\tau_j) \in k_r & \text{wenn } p = \text{“}k_r\text{“} \\ \Pi(\tau_i) \in k_{r_1} \backslash (k_{r_1} \cap k_{r_2}) & \\ \wedge \ \Pi(\tau_j) \in k_{r_n} \backslash (k_{r_n} \cap k_{r_{n-1}}) & \text{wenn } p = \text{“}k_{r_1} \ldots k_{r_n}\text{“} \end{cases} \tag{5.26}$$

Betrachtet man das Beispiel aus Abbildung 5.4 für eine Nachricht $g_j \in \gamma_i$, so ergibt

[1] Diese Platzierung kann erreicht werden, wenn ein Pfad mit nur einem Kommunikationsmedium betrachtet wird.

[2] Gleichung 5.26 ist aufgrund der Komplexität der letztlich entstehenden Gleichungen bewusst etwas abstrakter und jenseits der bisherigen Integer-Ungleichungen formuliert. Die Konversion in das Format eines SAT-Checkers ersetzt beispielsweise die Formulierung $\Pi(\tau_x) \in k_{...}$ in eine Disjunktion von Vergleichen der $\mathtt{place}_x$ Variablen mit den statisch ermittelbaren Knotennummern eines Kommunikationsmediums k_s. Dies ist am folgenden Beispiel zu ersehen.

sich aus (5.25) der folgende Satz an Gleichungen:

$$
\begin{aligned}
(\mathtt{Pf}_{g_j} = ph_0) \quad &\rightarrow \quad \left(\neg \mathtt{K}^3_{g_j} \wedge \neg \mathtt{K}^2_{g_j} \wedge \neg \mathtt{K}^1_{g_j}\right) \\
(\mathtt{Pf}_{g_j} = ph_1) \quad &\rightarrow \quad \Big(\neg \mathtt{K}^3_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \neg \mathtt{K}^2_{g_j} \\
&\qquad \wedge \big(\mathtt{place}_i = p_1 \vee \mathtt{place}_i = p_2 \vee \mathtt{place}_i = p_3\big) \\
&\qquad \wedge \big(\mathtt{place}_j = p_1 \vee \mathtt{place}_j = p_2 \vee \mathtt{place}_j = p_3\big)\Big) \\
&\qquad \vee \Big(\neg \mathtt{K}^3_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^2_{g_j} \\
&\qquad \wedge \big(\mathtt{place}_i = p_1 \vee \mathtt{place}_i = p_2\big) \\
&\qquad \wedge \mathtt{place}_j = p_4\Big) \\
(\mathtt{Pf}_{g_j} = ph_2) \quad &\rightarrow \quad \Big(\neg \mathtt{K}^3_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \neg \mathtt{K}^2_{g_j} \\
&\qquad \wedge \big(\mathtt{place}_i = p_1 \vee \mathtt{place}_i = p_2 \vee \mathtt{place}_i = p_3\big) \\
&\qquad \wedge \big(\mathtt{place}_j = p_1 \vee \mathtt{place}_j = p_2 \vee \mathtt{place}_j = p_3\big)\Big) \\
&\qquad \vee \Big(\neg \mathtt{K}^2_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^3_{g_j} \\
&\qquad \wedge \big(\mathtt{place}_i = p_1 \vee \mathtt{place}_i = p_2\big) \\
&\qquad \wedge \mathtt{place}_j = p_5\Big) \\
(\mathtt{Pf}_{g_j} = ph_3) \quad &\rightarrow \quad \Big(\neg \mathtt{K}^3_{g_j} \wedge \neg \mathtt{K}^1_{g_j} \wedge \mathtt{K}^2_{g_j} \wedge \big(\mathtt{place}_i = p_4 \vee \mathtt{place}_i = p_2\big) \\
&\qquad \wedge \big(\mathtt{place}_j = p_4 \vee \mathtt{place}_j = p_2\big)\Big) \\
&\qquad \vee \Big(\neg \mathtt{K}^3_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^2_{g_j} \wedge \mathtt{place}_i = p_4 \\
&\qquad \wedge \big(\mathtt{place}_j = p_1 \vee \mathtt{place}_j = p_3\big)\Big) \\
&\qquad \vee \Big(\mathtt{K}^3_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^2_{g_j} \wedge \mathtt{place}_i = p_4 \wedge \mathtt{place}_j = p_5\Big) \\
(\mathtt{Pf}_{g_j} = ph_4) \quad &\rightarrow \quad \Big(\neg \mathtt{K}^2_{g_j} \wedge \neg \mathtt{K}^1_{g_j} \wedge \mathtt{K}^3_{g_j} \wedge \big(\mathtt{place}_i = p_3 \vee \mathtt{place}_i = p_5\big) \\
&\qquad \wedge \big(\mathtt{place}_j = p_3 \vee \mathtt{place}_j = p_5\big)\Big) \\
&\qquad \vee \Big(\neg \mathtt{K}^2_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^3_{g_j} \wedge \mathtt{place}_i = p_5 \\
&\qquad \wedge \big(\mathtt{place}_j = p_1 \vee \mathtt{place}_j = p_2\big)\Big) \\
&\qquad \vee \Big(\mathtt{K}^2_{g_j} \wedge \mathtt{K}^1_{g_j} \wedge \mathtt{K}^3_{g_j} \wedge \mathtt{place}_i = p_5 \wedge \mathtt{place}_j = p_4\Big)
\end{aligned}
$$

Da die Topologie des Systems statisch gegeben ist, können diese Gleichungen leicht erzeugt werden. Gemäß Gleichung 5.25 ist dies für jede Kommunikation zwischen zwei Tasks zu erzeugen, was leicht erkennbar zu Komplexitätsproblemen führen kann, wenn es sich um viele Nachrichten in einer komplexen, über mehrere Hierarchie-Ebenen aufgebauten, Topologie handelt. Aus diesem Grund ist es unbedingt erforderlich, die Anzahl der durch die Kommunikationswegewahl induzierten Gleichungen zu reduzieren. Reduzierungen der Anzahl der Gleichungen können

erreicht werden, wenn die Informationen des Tasknetzwerkes und der Architektur genutzt werden, um beispielsweise Kommunikationspfade einzuschränken, wenn die beteiligten Tasks strikt auf eine Untermenge der Prozessorknoten (unter Nutzung der Menge der erlaubten Knoten π_i) festgelegt sind.
Tasks, die nur auf einer Teilmenge der Knoten platziert werden dürfen, können potentiell nicht alle Kommunikationsmedien des Systems für Ihre Nachrichten erreichen. Optimierungspotential bezüglich der Menge der Gleichungen für die Wegewahl ergibt sich durch eine Beschränkung der Menge der maximalen Pfade (vgl. Definition 5.3.1) durch eine eine Nachricht $\tau_s \xrightarrow{g_t} \tau_t$, wie sie in der folgenden Definition vorgenommen wurde.

Definition 5.3.4 (Menge der eingeschränkten maximalen Pfade). *Die Menge der eingeschränkten maximalen Pfade $MP(k_1) \downarrow_{g_t} \subseteq K^{\leq |K|}$ unter einer Kommunikation $\tau_s \xrightarrow{g_t} \tau_t$ ausgehend von einem Kommunikationsmedium k_1 ist gegeben durch:*

$$MP(k_1)\downarrow_{g_t} = \begin{cases} \emptyset & \text{wenn } \pi_s \cap k_1 = \emptyset \\ \Big\{ w_{k_1} \mid \begin{array}{l} w_{k_1} = \text{“}k_1 k_2 \ldots k_n\text{“} \\ \wedge \forall 1 \leq i,j \leq n : k_i \neq k_j \\ \wedge \forall 1 \leq i < n : k_i \cap k_{i+1} \neq \emptyset \\ \wedge \nexists k_{n+1} : k_n \cap k_{n+1} \neq \emptyset \\ \wedge \forall 1 \leq l \leq n : k_{n+1} \neq k_l \\ \wedge \pi_t \cap k_n \neq \emptyset \Big\} \end{array} & \begin{array}{l} \text{sonst} \\ (*) \end{array} \end{cases}$$

wobei π_s bzw. π_t die Menge der erlaubten Knoten für Task τ_s bzw. τ_t ist und $\gamma_s \ni g_t = (\tau_t, \cdot)$.

Es gilt, sowohl die mögliche Platzierung der Ziel-Task τ_t als auch der Sende-Task τ_s zu berücksichtigen. Dementsprechend werden nur für diejenigen Kommunikationsmedien längste Pfade in $MP(k_i) \downarrow_{g_t}$ aufgenommen, zu deren erstem Kommunikationsmedium k_i die Sende-Task τ_s eine Verbindung hat. Andernfalls existiert unter der Beschränkung g_t kein Pfad, der in diesem Medium startet.
Existiert diese Verbindung jedoch, können analog zu Definition 5.3.1 alle maximalen Pfade aufgenommen werden, für die nun zusätzlich noch gelten muss, dass die Ziel-Task τ_t Verbindung zum letzten Kommunikationsmedium der sequentiellen Abfolge von Medien des Pfades hat. Diese neue zusätzliche Bedingung ist in Definition 5.3.4 durch (*) gekennzeichnet.
Betrachtet man dies am Beispielsystem aus Abbildung 5.3 unter der Annahme der folgenden Platzierungsmöglichkeiten einer Taskkette $\tau_s \xrightarrow{g_t} \tau_t$

$$\pi_s = \{p_1, p_2, p_3, p_5\} \text{ und}$$

$$\pi_t = \{p_1, p_2, p_3, p_4\},$$

so ergeben sich die folgenden durch die Nachricht g_t eingeschränkten Pfade:

$$MP(k_1)\downarrow_{g_t} = \{\text{“}k_1 k_3\text{“}\}$$

$$MP(k_2) \downarrow_{g_t} = \{\text{“}k_2\text{“}\}$$

$$MP(k_3) \downarrow_{g_t} = \{\text{“}k_3k_1k_2\text{“}\}$$

Diese Beschränkung bewirkt eine differenziertere Betrachtung der Mengen der maximalen Pfade, so dass diese Menge zur Bestimmung einer ebenfalls bezüglich der Nachricht g_t eingeschränkten Pfadhülle eingesetzt werden muss.

Definition 5.3.5 (Eingeschränkte Pfadhüllen). *Sei die Menge aller maximalen durch eine Nachricht $\tau_s \xrightarrow{g_t} \tau_t$ eingeschränkten Pfade auf der gegebenen Systemtopologie gegeben durch*

$$\breve{MP} \downarrow_{g_t} = \bigcup_{1 \leq j \leq |K|} MP(k_j) \downarrow_{g_t} .$$

Dann ist die Menge der durch g_t eingeschränkten Pfadhüllen $PH \downarrow_{g_t}$ definiert durch

$$PH \downarrow_{g_t} = \left\{ ph_i \in 2^{K^{\leq |K|}} \mid \forall mp_i \in \breve{MP} \downarrow_{g_t} : ph_i = pre(mp_i) \right\} \cup \{\emptyset\}$$

Für das oben angegebene Beispiel kann nun der folgende Satz an Pfadhüllen $ph_i \in PH \downarrow_{g_t}$ konstruiert werden:

$$ph_1 = \{\text{“}k_1\text{“}, \text{“}k_1k_3\text{“}\}$$

$$ph_2 = \{\text{“}k_2\text{“}\}$$

$$ph_3 = \{\text{“}k_3\text{“}, \text{“}k_3k_1\text{“}, \text{“}k_3k_1k_2\text{“}\}$$

Dies wiederum führt zu der deutlich kompakteren Menge an Gleichungen für die Wegewahl der Kommunikation g_t:

$$\begin{aligned}
(\mathtt{Pf}_{g_t} = ph_1) \quad \rightarrow \quad & \Big(\neg \mathtt{K}^3_{g_t} \wedge \mathtt{K}^1_{g_t} \wedge \neg \mathtt{K}^2_{g_t} \\
& \quad \wedge \big(\mathtt{place}_s = p_1 \vee \mathtt{place}_s = p_2 \vee \mathtt{place}_s = p_3\big) \\
& \quad \wedge \big(\mathtt{place}_t = p_1 \vee \mathtt{place}_t = p_2 \vee \mathtt{place}_t = p_3\big)\Big) \\
& \vee \Big(\neg \mathtt{K}^2_{g_t} \wedge \mathtt{K}^1_{g_t} \wedge \mathtt{K}^3_{g_t} \\
& \quad \wedge \big(\mathtt{place}_s = p_1 \vee \mathtt{place}_s = p_2\big) \\
& \quad \wedge \mathtt{place}_t = p_5\Big)
\end{aligned}$$

$$\begin{aligned}
(\mathtt{Pf}_{g_t} = ph_2) \quad \rightarrow \quad & \Big(\neg \mathtt{K}^3_{g_t} \wedge \neg \mathtt{K}^1_{g_t} \wedge \mathtt{K}^2_{g_t} \wedge \big(\mathtt{place}_s = p_4 \vee \mathtt{place}_s = p_2\big) \\
& \quad \wedge \big(\mathtt{place}_t = p_4 \vee \mathtt{place}_t = p_2\big)\Big)
\end{aligned}$$

$$\begin{aligned}
(\mathtt{Pf}_{g_t} = ph_3) \quad \rightarrow \quad & \Big(\neg \mathtt{K}^2_{g_t} \wedge \neg \mathtt{K}^1_{g_t} \wedge \mathtt{K}^3_{g_t} \wedge \big(\mathtt{place}_s = p_3 \vee \mathtt{place}_s = p_5\big) \\
& \quad \wedge \big(\mathtt{place}_t = p_3 \vee \mathtt{place}_t = p_5\big)\Big) \\
& \vee \Big(\neg \mathtt{K}^2_{g_t} \wedge \mathtt{K}^1_{g_t} \wedge \mathtt{K}^3_{g_t} \wedge \mathtt{place}_s = p_5 \\
& \quad \wedge \big(\mathtt{place}_t = p_1 \vee \mathtt{place}_t = p_2\big)\Big) \\
& \vee \Big(\mathtt{K}^2_{g_t} \wedge \mathtt{K}^1_{g_t} \wedge \mathtt{K}^3_{g_t} \wedge \mathtt{place}_s = p_5 \wedge \mathtt{place}_t = p_4\Big)
\end{aligned}$$

Letztlich verlangt die Einsparung von Gleichungen bei der Wegewahl einen erhöhten Aufwand für eine statische Voranalyse des Systems, da die eingeschränkten Pfadhüllen gemäß Definition 5.3.5 pro Nachricht zu generieren sind. Dieser zusätzliche Aufwand stellt aber üblicherweise nur einen geringen Overhead bei der Laufzeit des Verfahrens dar, insbesondere bei großen Architekturen und Tasksystem (vgl. hierzu die Evaluationen in Kapitel 5.6).

Weitergehende Einschränkungen, wie z.B. verbotene Kommunikationsmedien für eine Nachricht, sind auf diesem Wege einfach integrierbar, werden aber im Rahmen dieser Arbeit nicht betrachtet.

5.3.6 Dynamische Erzeugung sub-lokaler Jitter- und Deadlinewerte

Wie bereits in Kapitel 4.4.2 dargestellt, ist bei der Benutzung mehrerer Kommunikationsmedien auf einem Pfad, den eine Nachricht vom Sende-Knoten bis zum Empfangsknoten passieren muss, die Synthese sublokaler Deadlines und Jitter-Werte notwendig. Hierzu wird für jede Nachricht und jedes Kommunikationsmedium eine Variable $\texttt{locD}^k_{g_t}$ eingeführt, die der SAT-Solver unter bestimmten Randbedingungen nahezu beliebig belegen und damit eine Optimierung durchführen kann. Die Randbedingungen leiten sich im wesentlichen aus der minimalen Größe der Übertragungszeit der Nachricht und der gegebenen Deadline $\Delta_\gamma(g_t)$ eine Nachricht ab. Sei zunächst dafür gesorgt, dass die sublokale Deadline auf 0 gezwungen wird, wenn das korrespondierende Kommunikationsmedium nicht benutzt wird:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \bigwedge_{\forall k \in K} \left(\neg \texttt{K}^k_{g_t} \rightarrow \texttt{locD}^k_{g_t} = 0 \right) \tag{5.27}$$

Sollte $\texttt{locD}^k_{g_t}$ benutzt werden weil das zugehörige Kommunikationsmedium auf dem Pfad der Nachricht liegt, so muss zusätzlich sichergestellt sein, dass die Deadline, die durch $\texttt{locD}^k_{g_t}$ aufgespannt wird, eine Übertragung ermöglicht, also in jedem Fall größer als die worst-case Approximation der Nachricht $\omega_k(g_t)$ auf dem angegebenen Medium ist.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \bigwedge_{\forall k \in K} \left(\texttt{K}^k_{g_t} \rightarrow \texttt{locD}^k_{g_t} \geq \omega_k(g_t) \right) \tag{5.28}$$

Nun kann das Optimierungsverfahren nahezu frei die jeweiligen sublokalen Deadlines generieren. Allerdings ist dabei sicherzustellen, dass die Summe der Deadlines und etwaiger Verzögerungen durch den Transfer der Nachricht über Gateway-Knoten nicht die Deadline der Nachricht $\Delta_\gamma(g_t)$ überschreitet. Daher wird die Wahl der sublokalen Deadline durch die folgenden Ungleichungen weiter eingeschränkt:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \left(\sum_{k \in K} \texttt{locD}^k_{g_t} + \texttt{serv}_{g_t} \leq \Delta_\gamma(g_t) \right) \tag{5.29}$$

serv quantifiziert dabei die zeitliche Größe, die durch die Kommunikationsserver auf den Gateways verbraucht wird. Zwar ist diese Größe direkt abhängig vom jeweiligen Knoten, es kann aber davon ausgegangen werden, dass die synthetischen

Tasks τ_{IN}^{OS} und τ_{OUT}^{OS} über eine Prozessor-unabhängige Deadline verfügen. Daher sei im folgenden die Deadline für derartige Tasks überapproximiert durch:

$$\Delta_{IN} = \max_{\forall k_i,k_j \in K : k_i \cap k_j \neq \emptyset} \Delta_{IN}(k_i \cap k_j)$$

$$\Delta_{OUT} = \max_{\forall k_i,k_j \in K : k_i \cap k_j \neq \emptyset} \Delta_{OUT}(k_i \cap k_j)$$

Damit lassen sich nun auf einfache Weise die durch Gateway-Übertragung entstehenden Kosten einer Kommunikation in einem verteilten System berechnen, in dem die Kommunikationsplatzierungs-Variable $\mathtt{K}_{g_t}^k$ benutzt wird, um die Kosten pro benutztem Medium zu addieren. Die Anzahl an benutzten Medien enthält auch den Fall von nur einem benutzten Kommunikationsmedium, welches jedoch keine Gateway-Kosten erzeugt. Folglich ist ein Korrekturfaktor $\mathtt{Corr}_{g_t}$ notwendig, der diese überzählige Benutzung eines Gateway-Knotens wieder herausfiltert.

$$\bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \left(\sum_{k \in K} \mathtt{K}_{g_t}^k \neq 0 \right) \rightarrow \mathtt{serv}_{g_t} = \sum_{k \in K} \mathtt{K}_{g_t}^k \cdot (\Delta_{IN} + \Delta_{OUT}) - \mathtt{Corr}_{g_t} \quad (5.30)$$

$\mathtt{Corr}_{g_t}$ muss dafür sorgen, dass im Falle einer Nutzung der Kommunikationsmedien genau ein Paar von Gateway-Tasks aus der Berechnung der sublokalen Deadline herausgefiltert wird. Alternativ ist keines der Kommunikationsmedien benutzt worden (alle $\mathtt{K}_{g_t}^k$ haben der Wert 0). Dieser Fall wird zwar bereits durch Gleichung 5.27 abgedeckt, jedoch muss beachtet werden, dass der Wert für `serv` nicht seinen Wertebereich verlässt und negativ wird. Dies wird insgesamt durch die folgenden Implikationen bewirkt:

$$\bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \left(\sum_{k \in K} \mathtt{K}_{g_t}^k > 0 \right) \rightarrow \mathtt{Corr}_{g_t} = 1 \cdot (\Delta_{IN} + \Delta_{OUT}) \quad (5.31)$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \left(\sum_{k \in K} \mathtt{K}_{g_t}^k = 0 \right) \rightarrow \mathtt{Corr}_{g_t} = 0 \quad (5.32)$$

Mithilfe der sublokalen Deadlines und der durch die entsprechend gewählte Pfadhülle gegebene Reihenfolge der Kommunikationsmedien auf einem Pfad können nun gemäß Gleichung 4.37 aus Kapitel 4.4.2 die sublokalen Überapproximationen des Jitters $\mathtt{J}_{g_t}^k$ berechnet werden. Sei zunächst $\max(ph_l)$ der längste Pfad einer Pfadhülle definiert durch

$$\forall ph_l \in PH : \max(ph_l) = p_i \mid \forall p_j \in ph_l, p_j \neq p_i : |p_j| < |p_i|.$$

Alle folgenden Argumentationen stützen sich auf den maximalen Pfad einer Pfadhülle ab. Dies ist erlaubt, da aufgrund der Definition der Pfadhüllen alle Pfade innerhalb der Hülle die gleiche Reihenfolge der Kommunikationsmedien besitzen. Somit kann der vollständige sublokale Jitter für jede Pfadhülle in der Reihenfolge ihrer Kommunikationsmedien aufgebaut werden. Dies ist in der folgenden Gleichung zu sehen, wobei jeweils $\tilde{p} = \max(ph_l)$ sei.

$$\bigwedge_{\forall ph_l \in PH} \bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall g_j \in \gamma_i} (\mathtt{Pf}_{g_j} = ph_l) \rightarrow \bigwedge_{k \in \tilde{p}} \left(\mathtt{J}_{g_t}^k = J_{g_t} + \sum_{j=1}^{pos(k,\tilde{p})-1} \left(\mathtt{locD}_{g_t}^{pr_j(\tilde{p})} - \beta_{pr_j(\tilde{p})}(g_t) \right) \right) \quad (5.33)$$

Jitter-Werte für nicht belegte Kommunikationsmedien können beliebig gewählt werden: Da die Nachricht g_t diese Medien nicht benutzt, wird — basierend auf der Belegung der Kommunikations-Platzierungs-Variablen $\mathtt{K}^k_{g_t}$ — diese Belegung in der Antwortzeitberechnung der Nachrichten auf diesen Medien keine Berücksichtigung finden.

5.3.7 Modellierung der Kommunikationslatenzen

Die Belegungsvariablen für Kommunikationsmedien werden nun genutzt, um die Latenzzeit von Nachrichten auf den jeweiligen Kommunikationssystemen zu berechnen. Eine Nachricht kann aufgrund der Eigenschaften der Topologie des Netzwerkes mehrere Kommunikationssysteme benutzen, so dass für jedes Bussystem eine Latenzzeit $\mathtt{lat}^k_{g_t}$ berechnet wird. Eine Nichtbelegung des Systems (angezeigt durch $\neg\mathtt{K}^k_{g_t}$) führt zu einem Wert von 0. Der Wert der Latenz selbst wird direkt aus der Antwortzeit der Nachricht auf dem angegebenen Kommunikationssystem abgeleitet, die weiter unten erläutert wird.

$$\bigwedge_{\forall\tau_i\in\mathcal{T}:\gamma_i\neq\emptyset}\ \bigwedge_{\forall g_t\in\gamma_i}\ \bigwedge_{\forall k\in K}(\mathtt{K}^k_{g_t})\rightarrow\mathtt{lat}^k_{g_t}=\mathtt{rm}^k_{g_t} \tag{5.34}$$

$$\bigwedge_{\forall\tau_i\in\mathcal{T}:\gamma_i\neq\emptyset}\ \bigwedge_{\forall g_t\in\gamma_i}\ \bigwedge_{\forall k\in K}\neg(\mathtt{K}^k_{g_t})\rightarrow\mathtt{lat}^k_{g_t}=0 \tag{5.35}$$

Gültige Nachrichtenplatzierungen müssen garantieren, dass die Summe der Latenzen der Nachricht auf ihrem gewählten Pfad durch die Topologie des Systems immer kleiner oder gleich der gegebenen Deadline der Nachricht ist. Dabei ist es, anders als bei Tasksystemen, unerheblich, ob die Nachricht auf einem Zeit-basierten oder einem Ereignis-basiertem Medium übertragen werden, da die Latenzzeiten etwaige Blockierungen enthalten. Diese Blockierungen sind inhärente Eigenschaften einer Übertragung auf dem jeweiligen Medium, so dass auf einen Korrekturfaktor, wie er beispielsweise bei Tasksystemen durch σ eingeführt wurde, verzichtet werden kann. Folglich muss gelten:

$$\bigwedge_{\forall\tau_i\in\mathcal{T}:\gamma_i\neq\emptyset}\ \bigwedge_{\forall g_t\in\gamma_i}\ \bigwedge_{k\in K}\mathtt{lat}^k_{g_t}\leq\mathtt{locD}^k_{g_t} \tag{5.36}$$

Die Antwortzeitberechnungen $\mathtt{rm}^k_{g_t}$ unterscheiden sich teilweise sehr stark abhängig davon, welches Kommunikationsmedium benutzt wird (Zeit-basiert, Ereignis-basiert oder Tokenring). Im folgenden werden daher alle 3 Varianten diskutiert. Die letztendliche Erstellung der Ungleichungssysteme für die jeweiligen Medien muss aus einer Architekturbeschreibung heraus getrieben werden. Da die Topologie eines Systems innerhalb eines inkrementellen Platzierungsschrittes unveränderlich ist, stellt dies kein Problem dar.
Bereits bei der Formalisierung der Antwortzeit-Berechnungen für Nachrichten in Kapitel 4.3 wurde ersichtlich, dass auch für Nachrichten Prioritäten zu vergeben sind. Dies kann vollkommen analog zu der Prioritätenbildung bei Tasks erfolgen (vgl. Gleichung 5.13 und folgende). Eine Prioritätsbeziehung wird wiederum über eine boolesche Variable $\mathtt{pm}^{g_j}_{g_i}$ gebildet, die nach der Deadline Monotonic Strategie

vergeben wird. Im Unterschied zu Tasks kann eine Nachricht aufgrund einer Pfadeigenschaft in der Systemtopologie mehreren Kommunikationsmedien mit jeweils unterschiedlichen Deadlines $\texttt{locD}^k_{g_i}$ zugeordnet werden. Demnach gibt es zwei unterschiedliche Verfahren, die Prioritäten zwischen zwei Nachrichten zu ermitteln: Auf der einen Seite können sie lokal pro Kommunikationsmedium gemäß $\texttt{locD}^k_{g_i}$ vergeben werden, auf der anderen Seite global gemäß ihrer jeweiligen Deadline $\Delta_\gamma(g_i)$. Da die sublokalen Deadlines frei vom Optimierungsverfahren vergeben werden können, ist nahezu jede mögliche Prioritätenrelation erzeugbar. Dies und nicht zuletzt auch die Tatsache, dass für den ersten Fall Prioritätenvariablen für jedes Nachrichtenpaar auf jedem Kommunikationsmedium vergeben werden müssen, steigert die Komplexität des Platzierungsproblems erheblich. Aus diesem Grund wird im Rahmen dieser Arbeit nur von der globalen Prioritätenbestimmung Gebrauch gemacht. Demzufolge können die Ungleichungen für Prioritäten von Nachrichten analog zu Gleichung 5.13 und folgende erstellt werden.

$$\bigwedge_{\forall g_i \in G(\mathcal{T})} \bigwedge_{\forall g_j \in G(\mathcal{T}) \setminus \{g_i\}} \texttt{pm}^{g_j}_{g_i} \geq 0 \wedge \texttt{pm}^{g_j}_{g_i} \leq 1 \wedge \texttt{pm}^{g_j}_{g_i} + \texttt{pm}^{g_i}_{g_j} = 1 \tag{5.37}$$

$$\bigwedge_{\forall g_i \in G(\mathcal{T})} \bigwedge_{\forall g_j \in G(\mathcal{T}) \setminus \{g_i\}} (\Delta_\gamma(g_i) > \Delta_\gamma(g_j) \rightarrow \texttt{pm}^{g_j}_{g_i} = 1 \tag{5.38}$$

$$\bigwedge_{\forall g_i \in G(\mathcal{T})} \bigwedge_{\forall g_j \in G(\mathcal{T}) \setminus \{g_i\}} (\Delta_\gamma(g_i) < \Delta_\gamma(g_j) \rightarrow \texttt{pm}^{g_j}_{g_i} = 0 \tag{5.39}$$

$$\bigwedge_{\forall g_i \in G(\mathcal{T})} \bigwedge_{\forall g_j \in G(\mathcal{T}) \setminus \{g_i\}} (\Delta_\gamma(g_i) = \Delta_\gamma(g_j) \rightarrow \texttt{pm}^{g_j}_{g_i} \leq 1 \tag{5.40}$$

Wiederum bedarf der Fall der Deadline-Gleichheit einer besonderen Behandlung, in der das Optimierungsverfahren die Prioritätsbeziehung frei belegen darf. Im Unterschied zu Tasksystemen sind die Deadlines von Nachrichten jedoch keinen Schwankungen während des Platzierungsprozesses unterworfen[1]. Dies kann die Erzeugung obiger Ungleichungen deutlich vereinfachen, da die Werte für die jeweilige Deadline $\Delta_\gamma(g_i)$ schon zur Systemerstellungszeit bekannt sind. Somit können viele Prioritätsvariablen entfallen, wenn in den noch folgenden Antwortzeitberechnungen für Nachrichten im Zuge der Interferenzberechnung nur diejenigen Nachrichten in die Ungleichungen aufgenommen werden, die tatsächlich aufgrund der Priorisierung Unterbrechungen verursachen können.
Damit sind nun die Grundlagen gelegt, die Antwortzeitberechnungen von Nachrichten auf den unterschiedlichen Typen von Kommunikationsmedien abzuleiten.

5.3.8 Modellierung für den Tokenring

Zunächst soll mit dem Tokenring-Protokoll die einfachste Variante vorgestellt werden. Sie dient gleichzeitig auch der Absicherung der Konzepte gegenüber anderen Arbeiten und ist daher notwendigerweise auch in dieser Einfachheit zu betrachten.

[1] Deadline von Tasks konnten aufgrund von lokaler Kommunikation um eine Nachrichtendeadline erweitert werden (vgl. Kapitel 4.2). Für Nachrichten gilt dies nicht gleichermaßen, auch nicht für sublokale Nachrichtenverschickung, wie in Kapitel 4.4.2 bereits dargelegt

Sei dazu $K_{TOK} = \{k \mid k \in K \wedge type(k) = \text{Tokenring}\}$ die Menge der Kommunikationsmedien vom Typ Tokenring.
Wie in Abschnitt 27 dargestellt ist die Antwortzeit einer Nachricht auf einem Tokenring gleich der Token Rotation Time $\mathtt{TRT}_k$ des Mediums, folglich gilt für ein Tokenring-Medium k:

$$\bigwedge_{k \in K_{TOK}} \bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \mathtt{rm}_{g_t}^k = \mathtt{TRT}_k \tag{5.41}$$

Die TRT_k selbst ist gemäß Gleichung 4.23 und 4.24 die Summe der Kommunikationsübertragungszeiten aller über dieses Medium versendeten Nachrichten $g_t = (\tau_t, s_t)$ (plus der Übertragungszeit für das Token). Sei dafür die Menge aller Nachrichten $\mathcal{G}(\mathcal{T})$ eines Tasksystems T durch

$$\mathcal{G}(\mathcal{T}) = \bigcup_{\forall \tau_i \in \mathcal{T}} \gamma_i$$

definiert. Dann ist die $\mathtt{TRT}_k$ unter Verwendung der Platzierungsvariablen für Kommunikationen $\mathtt{K}_{g_t}^k$ berechenbar durch

$$\bigwedge_{\forall k \in K_{TOK}} \mathtt{TRT}_k = \sum_{g_t \in \mathcal{G}(\mathcal{T})} \left(\mathtt{K}_{g_t}^k \cdot \frac{s_t \cdot 8}{\upsilon_k} \right) + |k| \cdot \xi. \tag{5.42}$$

Da im schlimmsten anzunehmenden Fall das Token jedem am Tokenring angeschlossenen Knoten übermittelt werden muss, auch wenn er keine Nachrichten zu senden hat, kann in Gleichung 5.42 auf die explizite Bildung der Tokenhaltezeit THT verzichtet werden. Stattdessen können die Latenzzeiten aller Nachrichten aufsummiert werden, und die Übertragungszeit des Tokens ξ ist nur abhängig von der Anzahl der Knoten.
Die gewählte Modellierung der Kommunikationslatenz nur aufgrund des TRT umfasst im übrigen auch alle Nachrichten, die über Gateway-Knoten auf den Tokenring gelangen. Gleichung 5.42 betrachtet alle Nachrichten des Tasksystems und entscheidet anhand der $\mathtt{K}_{g_t}^k$ Variablen der jeweiligen Nachricht, ob diese den betrachteten Bus benutzt. Nachrichten, die das betrachtete Kommunikationsmedium als Durchgangskommunikationsmedium nutzen, werden damit ebenfalls erfasst.
Damit ist die Modellierung für Kommunikationsmedien vom Typ Tokenring vollständig abgeschlossen.

5.3.9 Modellierung für Ereignis-basierte Bus-Systeme

Ereignis-basierte Kommunikationsmedien, wie beispielsweise der CAN-Bus, gleichen in den Analyse-Verfahren sehr stark der Scheduling-Analyse von Tasksystemen (vgl. die Formalisierung für den CAN-Bus in Kapitel 17). Folgerichtig sind die Ungleichungen analog zu erzeugen. Sei $K_{CAN} = \{k \mid k \in K \wedge type(k) = \text{CAN}\}$ die Menge der Kommunikationsmedien vom Typ CAN. Gemäß Gleichung 4.9 gilt für die Antwortzeit $\mathtt{rm}_{g_t}^k$ einer Nachricht g_t auf einem Ereignis-basierten Kommunikationsmedium k:

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{CAN}} \mathtt{rm}^k_{g_t} = c^k_{g_t} \cdot \rho_k + \tilde{c}^k_{g_t} \cdot \rho_k + \sum_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \left(\mathtt{interfm}^k_{g_t || g_i} \cdot \left((c^k_{g_i} - 1) + \tilde{c}^k_{g_i} \right) \right) \cdot \rho_k \tag{5.43}$$

Die Interferenzen $\mathtt{interfm}^k_{g_t||g_i}$ sind im Unterschied zu Task-Systemen ebenfalls bezogen auf das aktuell betrachtete Kommunikationsmedium zu ermitteln, da die Anzahl der Interferenzen nach Gleichung 4.10 abhängig von der Antwortzeit auf eben diesem Medium ist. Entscheidend ist außerdem zum einen die Priorisierung der Nachrichten, wie sie in Gleichung 5.37 und folgende erzeugt wurde, und zum anderen die Platzierung der zu vergleichenden Nachrichten. Letzteres wird durch die Platzierungsvariable für Nachrichten auf Kommunikationsmedien $\mathtt{K}^k_{g_i}$ festgelegt. Sind beide zu vergleichenden Nachrichten auf dem betrachteten Medium k platziert und erlaubt die Priorisierung Unterbrechungen, so sind die Interferenzkosten durch die Interferenzkosten-Variable $\mathtt{Im}^k_{g_i||g_t}$ festgelegt, andernfalls sind sie mit 0 anzugeben.

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{CAN}} \left(\mathtt{pm}^{g_i}_{g_t} \wedge \mathtt{K}^k_{g_i} \wedge \mathtt{K}^k_{g_t} \right) \rightarrow \mathtt{interfm}^k_{g_t||g_i} = \mathtt{Im}^k_{g_i||g_t} \tag{5.44}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{CAN}} \neg \left(\mathtt{pm}^{g_i}_{g_t} \wedge \mathtt{K}^k_{g_i} \wedge \mathtt{K}^k_{g_t} \right) \rightarrow \mathtt{interfm}^k_{g_t||g_i} = 0 \tag{5.45}$$

Interferenzkosten werden wiederum durch die Fixpunkt-Eigenschaft der Formalisierung in Kapitel 17 determiniert. Dies ist analog dem Vorgehen bei Tasksystemen in Gleichung 5.18 zu erreichen. Ist das betrachtete Nachrichtenpaar nicht auf dem Medium k platziert worden, so beträgt die Interferenz 0.

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{CAN}} (\mathtt{K}^k_{g_i} \wedge \mathtt{K}^k_{g_t}) \rightarrow \left(\mathtt{Im}^k_{g_i||g_t} \cdot t_{g_i} \geq \mathtt{rm}^k_{g_t} + \mathtt{J}^k_{g_i} \right) \wedge \left((\mathtt{Im}^k_{g_i||g_t} - 1) \cdot t_{g_i} < \mathtt{rm}^k_{g_t} + \mathtt{J}^k_{g_i} \right) \tag{5.46}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{CAN}} \neg (\mathtt{K}^k_{g_i} \wedge \mathtt{K}^k_{g_t}) \rightarrow \mathtt{Im}^k_{g_i||g_t} = 0 \tag{5.47}$$

Damit liegt für Ereignis-basierte Systeme eine vollständige Modellierung in Integer-Ungleichungen vor, auf dessen Basis das Platzierungsverfahren Nachrichtenübermittlungen berechnen kann.
Wird das Ereignis-basierte Medium in einem gemischten Bussystem benutzt, so handelt es sich in der Modellierung zwar um einen eigenständigen Bus, jedoch müssen die Antwortzeit-Berechnungen um den Blockierungsfaktor des sequentiell angeschlossenen Zeit-basierten Systems angepasst werden. Das grundsätzliche Vorgehen hierzu ist in Kapitel 4.3.3 beschrieben und wird in der Modellierung integriert, indem Glei-

chung 5.43 erweitert wird:

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{CAN}} \mathtt{rm}^k_{g_t} = c^k_{g_t} \cdot \rho_k + \tilde{c}^k_{g_t} \cdot \rho_k + \mathtt{tdmainterf}^k_{g_t} \cdot (\mathtt{tdmalen}_k - \mathtt{slot}^k_{dyn} \cdot \rho_k) + \sum_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \left(\mathtt{interfm}^k_{g_t || g_i} \cdot \left((c^k_{g_i} - 1) + \tilde{c}^k_{g_i}\right)\right) \cdot \rho_k \tag{5.48}$$

Die Größe der durch den Zeit-basierten Anteil verursachten Blockierung hängt von der Anzahl der Blockierungen und der Länge des blockierenden Zeit-basierten Anteils ab. Letzterer lässt sich aus der Größe des dynamischen Slots $\mathtt{slot}^k_{dyn}$ auf dem Medium k und Länge der TDMA-Round $\mathtt{tdmalen}_k$ auf Medium k ableiten. Beide Variablen werden weiter unten in der Betrachtung von Zeit-basierten Systemen aufgebaut (Gleichungen 5.56 und 5.57).
Entsprechend der Vorschrift für die Analyse der Antwortzeit aus Gleichung 4.26 aus Kapitel 4.3.3 ist der Faktor $\mathtt{tdmainterf}^k_{g_t}$ wiederum eine obere Abschätzung, die mit einer Fixpunkteigenschaft ausgestattet ist, so dass dies einfach mit Hilfe der bereits bekannten Integer-Ungleichungen über Gauß-Klammern berechnet werden kann:

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{CAN}} (\mathtt{tdmainterf}^k_{g_t} \cdot \mathtt{tdmalen}_k \geq \mathtt{rm}^k_{g_t}) \wedge ((\mathtt{tdmainterf}^k_{g_t} - 1) \cdot \mathtt{tdmalen}_k < \mathtt{rm}^k_{g_t}) \tag{5.49}$$

5.3.10 Modellierung für Zeit-basierte Bus-Systeme

Als Vertreter der Zeit-basierten Kommunikationsmedien soll hier zunächst die einfachere Variante des TTP vorgestellt werden. Anschließend wird eine Modellierung für die komplexe Slotverteilung am Beispiel des FlexRay-Bussystems aufgebaut werden. Sei $K_{TTP} = \{k \mid k \in K \wedge type(k) = \text{TTP}\}$ die Menge der Zeit-basierten Kommunikationsmedien mit einfacher Slotzuweisung pro TDMA-Round.

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \mathtt{rm}^k_{g_t} = (c^k_{g_t} + 1) \cdot \rho_k + \sum_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \mathtt{interfm}^k_{g_t || g_i} \cdot c^k_{g_i} \cdot \rho_k + \mathtt{block}^k_{g_t} \tag{5.50}$$

Zur Bestimmung der Interferenzen durch andere Nachrichten können die Gleichungen 5.44 bis 5.47 leicht verändert übernommen werden: Während Interferenzen auf Ereignis-basierten Bussystemen eine globale Eigenschaft des Mediums sind, ist dies bei Zeit-basierten nur lokal zu betrachten. Die feste Zuordnung von Prozessoren zu den Slots im TDMA-Verfahren kann nur bei Nachrichten, die auf dem selben Prozessorknoten platziert sind, Blockierungen hervorrufen. Informationen, welcher Knoten dies für jede Nachricht ist, sind in den Pfadhüllen nur implizit enthalten, so dass zunächst diese Information extrahiert werden muss. Dazu wird eine Variable $\mathtt{in}^k_{g_t}$ für jede Nachricht und jedes Kommunikationsmedium belegt, die angibt, welchen Gateway-Knoten die Nachricht beim Betreten des Mediums k nimmt. Dies ist statisch anhand der verschiedenen Pfadhüllen ermittelbar und bildet sich aus dem Schnitt des Vorgängermediums k' und k selbst[1]. Im Falle des ersten Mediums auf einem Pfad durch die Systemtopologie wird stattdessen der Knoten angegeben,

[1] Hier ist eine Vorbedingung, dass $k \cap k'$ tatsächlich nur einen Knoten enthält, was gemäß der eingangs beschriebenen Limitationen des Modells stets gewährleistet ist.

auf dem die Sender-Task der Nachricht $g_t \ni \gamma_s$ platziert ist (referenziert über die Platzierungsvariable $\texttt{place}_{\tau_s}$).

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \bigwedge_{\forall ph_l \in PH} (\texttt{Pf}_{g_t} = ph_l) \rightarrow \texttt{in}^k_{g_t} = \begin{cases} k \cap k' & \text{wenn } \max(ph_l) = \text{“}\ldots k' k \ldots\text{“} \\ \texttt{place}_{\tau_s} & \text{wenn } \max(ph_l) = \text{“}k \ldots\text{“} \\ \bot & \text{sonst} \end{cases} \tag{5.51}$$

Globale Blockierungen, hervorgerufen durch Slots anderer Prozessorknoten, werden hingegen im Blockierungsfaktor $\texttt{block}^k_{g_t}$ kompensiert.
Sei durch $src(g_j)$ die Sender-Task der Nachricht g_j bestimmbar. In die Liste der potentiellen Preemptionsverursacher müssen alle Nachrichten aufgenommen werden, deren Sender-Task auf dem Knoten allokiert sind, von dem aus die Nachricht g_t auf das Medium k gesendet wird. Dieser Knoten ist entweder der zur Sender-Task von g_t zugehörige Knoten oder aber ein Gateway-Knoten für die Verschickung von g_t. Zusätzlich müssen alle Nachrichten aufgenommen werden, die den besagten Knoten ebenfalls als Gateway-Knoten benutzen. Diese Informationen können aus den `in`-Variablen extrahiert werden, die entsprechend definiert wurden. Es gilt folglich für die Interferenz durch andere Nachrichten auf einem Knoten p, der dem zu g_t gehörenden Slot zugeordnet ist (analog zu Gleichung 5.44 bis 5.47):

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{TTP}} \left(\texttt{pm}^{g_i}_{g_t} \wedge \texttt{K}^k_{g_i} \wedge \texttt{K}^k_{g_t} \wedge (\texttt{in}^k_{g_t} = \texttt{in}^k_{g_i})\right) \rightarrow \texttt{interfm}^k_{g_t||g_i} = \texttt{Im}^k_{g_i||g_t} \tag{5.52}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{TTP}} \neg \left(\texttt{pm}^{g_i}_{g_t} \wedge \texttt{K}^k_{g_i} \wedge \texttt{K}^k_{g_t} \wedge (\texttt{in}^k_{g_t} = \texttt{in}^k_{g_i})\right) \rightarrow \texttt{interfm}^k_{g_t||g_i} = 0 \tag{5.53}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{TTP}} \left(\texttt{K}^k_{g_i} \wedge \texttt{K}^k_{g_t} \wedge (\texttt{in}^k_{g_t} = \texttt{in}^k_{g_i})\right) \rightarrow (\texttt{Im}^k_{g_i||g_t} \cdot t_{g_i} \geq \texttt{rm}^k_{g_t} + \texttt{J}^k_{g_i}) \wedge \left((\texttt{Im}^k_{g_i||g_t} - 1) \cdot t_{g_i} < \texttt{rm}^k_{g_t} + \texttt{J}^k_{g_i}\right) \tag{5.54}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \bigwedge_{\forall k \in K_{TTP}} \neg \left(\texttt{K}^k_{g_i} \wedge \texttt{K}^k_{g_t} \wedge (\texttt{in}^k_{g_t} = \texttt{in}^k_{g_i})\right) \rightarrow \texttt{Im}^k_{g_i||g_t} = 0 \tag{5.55}$$

In strikten TDMA-Verfahren, wie es beispielsweise TTP anbietet, ist in jeder TDMA-Round jedem Prozessorknoten nur genau ein Slot zugeordnet. Die Länge dieser Slots hängt nicht zuletzt von der Größe der zu verschickenden Frames ab, kann aber natürlich auch größer als ein maximaler Frame gewählt werden. Im folgenden wird davon ausgegangen, dass die zeitliche Länge immer mindestens ein ganzzahliges Vielfaches der Übertragungsdauer eines maximalen Frames ist. Die Größe selbst unterliegt dem Optimierungsverfahren, ist jedoch nach oben durch eine statische Grenze N^k_{Slot} begrenzt. Aufgrund der statischen Anzahl an Knoten, die an ein Kommunikationsmedium angeschlossen sind, kann für jedes Medium und jeden Knoten die Variable

$\texttt{slot}_p^k$ mit $p \in k$ definiert werden.

$$\bigwedge_{\forall k \in K_{TTP}} \bigwedge_{\forall p \in k} (\texttt{slot}_p^k > 0) \wedge (\texttt{slot}_p^k \leq N_{\text{Slot}}^k) \tag{5.56}$$

Damit ist nun auch die Länge der TDMA-Round $\Lambda(TDMA)$ definiert:

$$\bigwedge_{\forall k \in K_{TTP}} \texttt{tdmalen}_k = \sum_{p \in k} \texttt{slot}_p^k \cdot \rho_k \tag{5.57}$$

Ein weiterer Blockierungsfaktor in der Antwortzeitberechnung auf Zeit-basierten Kommunikationsmedien ist laut Gleichung 5.50 die Blockierung durch andere Slots $\texttt{block}_{g_t}^k$. Gemäß Gleichung 4.12 aus Kapitel 4.3.2 ist dies die Länge der TDMA-Round abzüglich der Länge des eigenen Slots, der wiederum von dem benutzten Prozessorknoten abhängt. Letzterer wird mittels der $\texttt{in}_{g_t}^k$-Variable referenziert, so dass sich in $\texttt{ownslotlen}_{g_t}^k$ die Länge des benutzten Slots berechnen lässt:

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \bigwedge_{\forall p \in k} (\texttt{in}_{g_t}^k = p) \rightarrow \texttt{ownslotlen}_{g_t}^k = \texttt{slot}_p^k \tag{5.58}$$

Dieser Wert wiederum ist für alle TDMA-Runden, die für die Versendung der Nachrichten von Nöten sind, aufzusummieren. Entsprechend der Vorschrift in Gleichung 4.12 ist dies erneut eine Fixpunkt-Berechnung der Antwortzeit der Nachricht mittels der oberen Abschätzung. Demzufolge wird dies in der bereits bekannten Art und Weise determiniert, indem die Anzahl der TDMA-Rounds, die innerhalb der Antwortzeit der Nachricht $\texttt{rm}_{g_t}^k$ blockieren können, in Gleichung 5.60 und 5.61 erzeugt und anschließend in Gleichung 5.59 mit der Länge der TDMA-Blockierung multipliziert werden.

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \texttt{block}_{g_t}^k = \texttt{Imb}_{g_t}^k \cdot \left(\texttt{tdmalen}_k - \texttt{ownslotlen}_{g_t}^k \cdot \rho_k\right) \tag{5.59}$$

$$\begin{aligned} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \left(\texttt{K}_{g_t}^k\right) \rightarrow \; & ((\texttt{Imb}_{g_t}^k \cdot \texttt{tdmalen}_k \geq \texttt{rm}_{g_t}^k) \\ & \wedge ((\texttt{Imb}_{g_t}^k - 1) \cdot \texttt{tdmalen}_k < \texttt{rm}_{g_t}^k)) \end{aligned} \tag{5.60}$$

$$\bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{TTP}} \left(\neg \texttt{K}_{g_t}^k\right) \rightarrow \texttt{Imb}_{g_t}^k = 0 \tag{5.61}$$

Damit liegt für Zeit-basierte Kommunikationsmedien eine vollständige Modellierung vor, die es einem auf SAT-Checking basierenden Optimierungsverfaren ermöglichen, Nachrichtenübermittlungen und deren Antwortzeiten und Einhaltung der zugehörigen Deadlines zu ermitteln.

Zur Unterstützung gemischter Zeit- und Ereignis-basierter Kommunikationssysteme ist für den Zeit-getriebenen Anteil lediglich in Gleichung 5.56 ein zusätzlicher dynamischer Slot $\texttt{slot}_{dyn}^k$ hinzuzufügen, sowie die 5.56 nachfolgenden Gleichungen um diesen Slot zu erweitern.

5.3.11 Modellierung für komplexe Slotverteilung am Beispiel des FlexRay 2.0

Die Modellierung für die komplexe Slotverteilung, wie sie in Kapitel 25 dargestellt wurde, soll hier am Beispiel der FlexRay-Spezifikation in der Version 2.0[56] dargestellt werden. Im Unterschied zu der Version 0.2[55] des FlexRay-Protokolls gibt es rigide Einschränkungen bzgl. der Länge der Slots und der zu verwendenden Frames innerhalb eines Slots: Alle Slots müssen exakt gleich lang sein und es müssen in jedem Slot Frames der gleichen Größe versendet werden. Hinsichtlich der multiplen Slotbelegung eines Knotens innerhalb einer TDMA-Round hat sich allerdings nichts geändert, so dass das in Kapitel 25 nachgewiesene Verfahren zur Bestimmung der Antwortzeit auf die Erstellung von Gleichungssystemen übertragen werden muss. Bei der Betrachtung der möglichen Verschiebungen der TDMA-Round hilft hierbei die strikte Festlegung auf Slots gleicher Länge. Trotzdem ist die Anzahl der zu analysierenden Verschiebungen abhängig von der Anzahl der Slots, die ein Prozessorknoten belegt. Diese wiederum unterliegt in einem Optimierungsschritt der Kontrolle des Platzierungsverfahren. Da eine dynamische Anzahl an Gleichungen nicht ausdrückbar ist, soll im folgenden von einer absoluten Obergrenze an möglichen Slots ausgegangen werden[1].
Sei $K_{MSA} = \{k \mid k \in K \wedge type(k) = \text{FlexRay}\}$ die Menge der Kommunikationsmedien vom Typ FlexRay, die die multiple Slotallokation durch Prozessorknoten erlauben. Sei die maximale nutzbare Anzahl an Slots auf einem entsprechenden Medium k mit S^k gegeben. Für jeden dieser Slots wird nun eine Belegungsvariable $\mathtt{s}_i^k$ mit $i \in \{0, \ldots, |S^k| - 1\}$ eingeführt, die den diesem Slot zugeordneten Prozessorknoten $p \in k$ aufnehmen kann. Aufgabe des Optimierungsverfahrens ist es nun, eine unter der Evaluierungsfunktion günstige Belegung der Slots zu finden, wobei auch Slots ungenutzt ($\mathtt{s}_i^k = \perp$) bleiben können. Letztere sind dementsprechend auch mit einer Länge von 0 ausgestattet, wohingegen alle belegten Slots eine identische Länge $\mathtt{slotlength}^k$ aufweisen. Auch diese Länge ist vom Optimierungsverfahren zu bestimmen und muss ein ganzzahliges Vielfaches des längsten möglichen Frames ω_k sein. Abhängig von der Allokation der Slots zu Prozessorknoten wird folglich der dynamische Anteil der Slotlänge $\mathtt{slen}_i^k$ eines spezifischen Slots Nummer i aus der TDMA-Round bestimmt:

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{i=0}^{S^k-1} \left(\mathtt{s}_i^k = \perp\right) \rightarrow \mathtt{slen}_i^k = 0 \tag{5.62}$$

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{i=0}^{S^k-1} \neg \left(\mathtt{s}_i^k = \perp\right) \rightarrow \mathtt{slen}_i^k = \mathtt{slotlength}^k \cdot \omega_k \tag{5.63}$$

[1] Dieses Vorgehen ist absolut üblich in Zeit-basierten Bussystemen und beispielsweise konform zur TTP-Spezifikation, die 64 Slots als Obergrenze festlegt. FlexRay spezifiziert in der Version 2.0 eine solche Beschränkung von 1028 Slots

Die Länge der TDMA-Round kann dann einfach aus der Summe der Längen der einzelnen Slots gebildet werden.

$$\bigwedge_{\forall k \in K_{MSA}} \texttt{tdmalen}_k = \sum_{i=0}^{S^k-1} \texttt{slen}_i^k \tag{5.64}$$

Gleichung 4.19 legt den Grundstein für die Antwortzeitberechnung einer Nachricht bezüglich einer bestimmten Verschiebung z der TDMA-Round auf dem Medium k. Übertragen auf das übliche Vorgehen bei der Generierung von Ungleichungen für den SAT-Checker ergibt sich die folgende Berechnungsvorschrift:

$$\begin{aligned}\bigwedge_{z=0}^{S^k-1} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{\forall k \in K_{MSA}} \texttt{rm}_{g_t}^{k,z} = \; & (c_{g_t}^k + 1) \cdot \rho_k + \sum_{\forall g_i \in \mathcal{G}(\mathcal{T}) \setminus \{g_t\}} \texttt{interfm}_{g_t||g_i}^{k,z} \cdot c_{g_i}^k \cdot \rho_k \\ & + \sum_{j=0}^{S^k-1} \texttt{interfb}_{g_t,j}^{k,z} \cdot \texttt{slen}_j^k \cdot \texttt{y}_{g_t,j}^k\end{aligned} \tag{5.65}$$

wobei die Summe über alle $\texttt{interfm}_{g_t||g_i}^{k,z}$ blockierende Nachrichten mit einer höheren Priorität berücksichtigt, die vom gleichen Knoten auf das Medium k gesendet werden müssen. Für die Berechnung von `interfm` sei auf das Vorgehen im letzten Abschnitt verwiesen, welches hier identisch eingesetzt werden kann.
$\texttt{interfb}_{g_t,j}^{k,z}$ hingegen stellt die Blockierung durch andere, nicht dem betrachteten Prozessorknoten zugeordnete, Slots der TDMA-Round zur Verfügung. Da mehrere Slots dem betrachteten Prozessorknoten zugeordnet sein können, sind diese natürlich aus der Blockierung durch die TDMA-Round herauszurechnen. Dafür notwendig ist eine Variable $\texttt{y}_{g_t,i}^k$, die dafür sorgt, dass bei einem dem betrachteten Prozessor zugeordneten Slot die Blockierungszeit auf 0 gezwungen wird:

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{i=0}^{S^k-1} \left(\texttt{s}_i^k = \texttt{in}_{g_t}^k\right) \rightarrow \texttt{y}_{g_t,i}^k = 0 \tag{5.66}$$

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{i=0}^{S^k-1} \neg\left(\texttt{s}_i^k = \texttt{in}_{g_t}^k\right) \rightarrow \texttt{y}_{g_t,i}^k = 1 \tag{5.67}$$

Für die Interferenz der blockierenden übrigen Slots $\texttt{interfb}_{g_t,j}^{k,z}$ der betrachteten Verschiebung der TDMA-Round ist zu beachten, dass die in Kapitel 25 dargestellte Betrachtung der Blockierungen (vgl. zum Beispiel Abbildung 4.11) aus mehreren, sequentiell aufeinanderfolgenden Slots zusammengesetzt ist, die nicht dem Prozessorknoten $\texttt{in}_{g_t}^k$ zugeordnet sind. Da dies eine dynamische Eigenschaft ist, kann die Berechnung der Antwortzeit für eine TDMA-Verschiebung gemäß Gleichung 4.19 nicht auf dieser Basis erzeugt werden. Stattdessen wird hier jeder nicht $\texttt{in}_{g_t}^k$ zugeordnete Slot als eigenständige Blockierung behandelt. Da die maximale Anzahl an Slots durch S^k beschränkt ist, ist dies wiederum statisch ermittelbar, und entsprechende Blockierungsanteile können in Gleichung 5.65 erzeugt werden. Diese Herangehensweise ist formal abgesichert durch Lemma 4.3.1: Jede Verschiebung innerhalb

eines blockierenden, zusammengesetzten Slots führt zu einer gleichen oder geringeren Antwortzeit. Mithin kann diese Verschiebung auch auf den Slotgrenzen erfolgen. Gleichzeitig wird durch die Offsets der Slots und deren sequentielles Zusammenspiel in der Blockierungsberechnung sichergestellt, dass aufeinanderfolgende blockierende Slots sich exakt so auf die Antwortzeit auswirken, als würde der zusammengesetzte Block betrachtet werden. Die Offsets der als Blockierung wirkenden Slots können nun aus den Informationen über die (feste) Anordnung der Slots in der Zeit und deren Längen gewonnen werden.
Sei die Position des Slots pos_h^z in der ursprünglichen TDMA-Round bezüglich eines Slots der Nummer h in einer Verschiebung der TDMA-Round um z Slots folgendermaßen definiert[1]:

$$pos_h^z = \big(\big(S^k - z\big) + h\big) \bmod S^k$$

Der Faktor pos_h^z ist statisch zur Generierungszeit aufgrund der Struktur des Kommunikationsmediums bestimmbar, so dass eine vom Problem abhängige, statische Anzahl an Gleichungen erzeugt werden kann. Die Modulo-Operation ist in Zusammenhang mit der maximalen Slot-Anzahl S^k und der Verschiebung notwendig, um den Index des Slots in der verschobenen TDMA-Round abzubilden auf den Index eben dieses Slots in der original TDMA-Round. Diese Information wiederum ist notwendig, um über die `slen`-Variablen Zugriff auf die Länge des Slots zu bekommen. Mit pos_h^z kann folglich der Startoffset eines Slots h innerhalb einer Verschiebung z der TDMA-Round des Mediums k bestimmt werden und damit folgt für die Differenz der Antwortzeit und dem Offset des betrachteten Slots:

$$\mathtt{w}_{g_t,j}^{k,z} = \begin{cases} 0 & \text{wenn } \mathtt{rm}_{g_t}^{k,z} - \sum_{h=0}^{j-1} \mathtt{slen}_{pos_h^z}^k < 0 \\ \mathtt{rm}_{g_t}^{k,z} - \sum_{h=0}^{j-1} \mathtt{slen}_{pos_h^z}^k & \text{sonst} \end{cases}$$

Ein Wert kleiner oder gleich 0 in der Differenz der Antwortzeit $\mathtt{rm}_{g_t}^{k,z}$ und dem Offset besagt, dass die zum Offset gehörende Blockierung ihren Startzeitpunkt jenseits des Intervalls $[0, \mathtt{rm}_{g_t}^{k,z})$ hat und damit keine Blockierung zur Antwortzeit beiträgt. Für die erste Blockierung wird hier hingegen ein Offset von 0 angenommen. Dies entspricht der Formalisierung in Kapitel 25 und stellt im übrigen die Interpretation einer kritischen Instanz dar.
Dies kann für die Interferenz der Blockierungen durch andere Slots benutzt werden, indem die Gauß-Klammer $\left\lceil \frac{r_{g_t} - O_j}{p_{g_j}} \right\rceil$ für den Blockierungsfaktor durch andere TDMA-Slots gemäß Gleichung 4.19 in der bekannten Art und Weise abgebildet wird:

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{z=0}^{S^k-1} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{j=0}^{S^k-1} (\mathtt{K}_{g_t}^k) \rightarrow \; ((\mathtt{interfb}_{g_t,j}^{k,z} \cdot \mathtt{tdmalen}_k \geq \mathtt{w}_{g_t,j}^{k,z}) \wedge ((\mathtt{interfb}_{g_t,j}^{k,z} - 1) \cdot \mathtt{tdmalen}_k < \mathtt{w}_{g_t,j}^{k,z})) \tag{5.68}$$

[1] Die Berechnung entspricht einer komplementären (um $S^k - z$ Slots) Rotation der ursprünglichen, nicht verschobenen TDMA nach rechts. Dies entspricht einer Links-Rotation um z Slots.

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{z=0}^{S^k-1} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{j=0}^{S^k-1} \neg(\mathtt{K}_{g_t}^k) \rightarrow \mathtt{interfb}_{g_t,j}^{k,z} = 0 \tag{5.69}$$

Letzlich kann hieraus eine Antwortzeitberechnung erzeugt werden, die Gleichung 4.22 genügt, wobei allerdings die maximale Anzahl der Slots beschränkt ist auf S^k[1].

$$\bigwedge_{\forall k \in K_{MSA}} \bigwedge_{\forall g_t \in \mathcal{G}(\mathcal{T})} \bigwedge_{i=0}^{S^k-1} \left(\bigwedge_{\forall j \in \{1,\ldots,S^k\}, i \neq j} \mathtt{rm}_{g_t}^{k,i} \geq \mathtt{rm}_{g_t}^{k,j} \right) \rightarrow \mathtt{rm}_{g_t}^{k} = \mathtt{rm}_{g_t}^{k,i} \tag{5.70}$$

Damit liegt für Zeit-basierte Kommunikationsmedien mit multipler Prozessorknoten-Zuordnung innerhalb einer TDMA-Round eine vollständige Modellierung vor, die es einem auf SAT-Checking basierenden Optimierungsverfaren ermöglichen soll, Nachrichtenübermittlungen und deren Antwortzeiten und Einhaltung der zugehörigen Deadlines zu ermitteln.

Zur Unterstützung gemischter Zeit- und Ereignis-basierter Kommunikationssysteme ist für den Zeit-getriebenen Anteil lediglich ein zusätzlicher dynamischer Slot $\mathtt{s}_{dyn}^k$ hinzuzufügen, sowie die Anzahl der Slots S^k um eins zu erhöhen.

Weitergehende Konzepte, wie beispielsweise die in Kapitel 4.3.5 dargestellten gemischten Ereignis- und Zeit-getriebenen Echtzeit-Betriebssysteme, können mit dem gleichen Verfahren integriert werden, wie es in den vorangegangen Abschnitten skizziert wurde. Aufgrund der Ähnlichkeit der Prinzipien kann im folgenden aber darauf verzichtet werden, und wir werden uns stattdessen auf die Anwendbarkeit des Verfahrens sowohl im Hinblick auf eine gute Integration in einen Entwicklungsprozess als auch bezüglich der Skalierbarkeit und Aussagekraft für industriell relevante Problemgrößen konzentrieren. Zunächst soll jedoch noch eine etwas genauere Betrachtung der algorithmischen Seite der Optimierungsmethodik folgen, sowie anschließend eine kurze Motivation, wie die in den vorangegangenen Abschnitten dargestellte Modellierung des Platzierungsproblems auch auf andere Optimierungsverfahren übertragen werden kann.

5.4 Verwendung von HySAT

Viele der oben angegebenen Gleichungen können in der Schnittstelle zum verwendeten SAT-Checker HySAT stark vereinfacht werden, da HySAT ein komfortables Interface mit z.B. `if-then-else`-Konstrukten zur Verfügung stellt, das letztlich eine automatische Abbildung der hier vorgestellten Modellierung durchführt (vgl. Dokumentation zu HySAT[72]). In Kapitel 5.6 wird detaillierter beschrieben, wie diese Konstrukte helfen können, die Komplexität des Modells zu reduzieren. Um aber kompatibel zu anderen SAT-Checkern zu bleiben, wurde die entsprechende allgemeine Formulierung in Gleichung 5.5 bis 5.70 gewählt.

[1] Die Beschränkung der Slotlänge auf einen festen Wert im FlexRay-Protokoll ist für die hier vorgestellte Modellierung nicht notwendig. Es ist durchaus möglich, auch unterschiedliche Slotlängen über die jeweilige `slen`-Variable zu definieren, ohne dass die Erfüllbarkeit dieser Modellierung in Frage gestellt wird.

Der Kern des HySAT-Werkzeuges ist eingebunden in ein Framework namens TATONO, welches Analysen auf den Tasksystemen und der Architektur durchführt und daraus den eigentlichen Eingabecode für HySAT ableitet. Eine beispielhafte Darstellung eines kleinen Systems ist in Anhang D.2 beschrieben. Zudem wird innerhalb von TATONO die Unterstützung des inkrementellen Integrationsprozesses inklusive der Generierung der lokalen Deadlines und Jitter-Werte durchgeführt, wie es im Kapitel 6.2 beschrieben wird. Dazu gehört ebenfalls das Vorhalten eines entsprechenden System-Repositories, in dem die inkrementelle platzierten Teilsysteme eingecheckt werden können und das damit in jedem Schritt der inkrementellen Integration eine aktuelle Konfiguration enthält. Einen Überblick über die Software-Struktur des Frameworks ist in Anhang D.1 beschrieben.
Dieser kleine Überblick über die auf HySAT basierende Platzierungs-Software soll hier genügen. Stattdessen werden wir uns im folgenden der Optimierungsmethodik an sich widmen, also welche grundlegenden Algorithmen kommen zum Einsatz, um aus einer Entscheidungsprozedur für Erfüllbarkeitstest tatsächlich ein optimales Optimierungsverfahren zu machen. Weiterhin wird an einem Beispiel dargelegt, wie verschiedene Optimierungskriterium in dieses Verfahren integriert werden können.

5.4.1 Optimierungsmethodik

Die mittels der Gleichungen 5.5 bis 5.70 erzwungenen möglichen Belegungen und die damit korrespondierenden Platzierungen sollen nun gemäß einer Bewertungsfunktion charakterisiert und optimiert werden. Da SAT-basierte Methoden für jeden Lauf exakt eine erfüllende Belegung produzieren, muss die Optimierungsfunktion so gewählt werden, dass sie bzgl. des Bewertungskriteriums nicht optimale Lösungen verbietet. Üblicherweise kann eine solche Optimierungsfunktion nicht auf einem so feingranularem Kriterium a priori gefunden werden, da dies die Kenntnis der optimalen Platzierung vorraussetzt. Vielmehr soll mithilfe eines iterativen Verfahrens das Kriterium schrittweise verfeinert werden, um sich der optimalen Lösung anzunähern. Dies bedeutet, dass es eine Vielzahl von SAT-Läufen gibt, die durch immer stringentere Gleichungen Lösungen in der Nähe des Optimums finden, bis schließlich das Optimum selbst gefunden ist. Anhand eines typischen Vertreters aus der Menge der Optimierungsfunktionen soll dieses Verfahren im folgenden demonstriert werden.
Beispielhaft für typische Optimierungsfunktionen soll hier das Ziel betrachtet werden, die Auslastung aller Prozessorknoten möglichst gleich zu gestalten. Die Auslastung eines Knotens (im englischen Utilisation genannt) ist ein wohlbekannter Faktor (vergleiche [104]), der aus der Summe der Verhältnisse von Laufzeit zu Periode der Tasks berechnet wird. Eine gleichmäßige Auslastung der Knoten wird erreicht, wenn die jeweilige Utilisation der Knoten nahe an der mittleren Utilisation des Systems liegt. Diese wird durch das folgende Schema berechnet.

$$\bar{U} = \frac{1}{|P|} \cdot \sum_{\forall \tau_i \in \mathcal{T}} \bar{U}_i$$

Da die mittlere Auslastung voraussetzt, dass die mittlere WCET eines Tasks verwendet wird, die WCETs der Tasks jedoch abhängig vom Prozessorknoten sein können,

ist zunächst der mittlere Utilisationsanteil jedes Task zu berechnen.

$$\forall \tau_i \in T : \bar{U}_i = \frac{\frac{1}{|P|} \cdot \sum_{\forall p \in P} c_i(p)}{t_i}$$

Bei der Betrachtung gleichmäßiger Verteilungen wird üblicherweise die Varianz verwendet (vergleiche z.B. [23]), die für das Platzierungsproblem formuliert das folgende Aussehen hat:

$$\sigma^2 = \frac{1}{|P|} \cdot \sum_{\forall p \in P} \left(U_p - \bar{U}\right)^2,$$

wobei die Utilisation U_p eines Knoten p unter einer Platzierung Π wiefolgt berechnet wird:

$$\forall p \in P : U_p = \sum_{\forall \tau_i \in \mathcal{T} : \Pi(\tau_i) = p} \frac{c_i(\Pi(\tau_i))}{t_i}$$

Die Varianz kann als Optimierungskriterium behandelt werden: Bei einer vollständig gleichmäßigen Verteilung wird die Varianz $\sigma^2 = 0$, d.h. in einem Optimierungsverfahren ist der Faktor σ^2 zu minimieren.

Bei der Umsetzung der Varianz-Gleichungen in eine Konjunktion von Integer-Gleichungen müssen zunächst die reell-wertigen Utilisationsfaktoren in einen Integerbereich umgewandelt werden. Dies geschieht durch die Multiplikation aller Werte um eine genügend große Integerkonstante und anschließende Rundung. Auf diese Weise ist die Modellierung der Varianz möglich, leidet allerdings unter einer ungenaueren Auflösung, so dass das Optimierungsverfahren nur eine angenäherte optimale Lösung liefern kann. Für die iterative Anwendung von SAT-Lösungen wird nun die Gleichung

$$\sigma^2 \leq C \tag{5.71}$$

dem Gleichungssystem des Platzierungsproblems hinzugefügt und bei mehreren Aufrufen von HySAT jeweils die Konstante C variiert[1]. Die Variationen von C werden durch ein an binäre Suche[32] angelehntes Verfahren bestimmt. Dieses Vorgehen wird so lange wiederholt, bis ein Schwellwert ε_O in der Veränderung zum vorherigen Wert von C erreicht ist. Algorithmus 3 stellt das Verfahren dar. Das Ergebnis eines HySAT-Laufes wird in einer Ergebnisvariablen s gespeichert, die zum einen den Zustand der Lösung (also infeasable oder feasable in $s.state$) angibt, zum anderen das Ergebnis inklusive dem Wert für das Optimierungskriterium C enthält ($s.solution$ bzw. $s.C$). Wird eine gültige Lösung gefunden, so wird, basierend auf dem Ergebnis für das Optimierungskriterium $s.C$, ein neuer Wert für die binäre Suche nach dem Optimum festgelegt. Andernfalls erhöht der Algorithmus das Optimierungskriterium durch die Hälfte der Schrittweite. Das Finden einer ersten Lösung wird durch ein stetiges Verdoppeln des Optimierungskriteriums erreicht. Ist keine Lösung zu finden, terminiert der Algorithmus theoretisch nicht[2]. In der Praxis hingegen wird in realen Systemen der Nachweis der Erfüllbarkeit an den minimalen Constraints eines

[1] Bei einem maximierenden Optimierungskriterium ist entsprechend ein $\geq$ in Gleichung 5.71 einzufügen.

[2] Hier ist ein entsprechend zu parametrisierender Timer eingebaut.

Systems scheitern, über die hinaus keine weitere Suche notwendig wird (beispielsweise macht es keinen Sinn Abweichungen von der durchschnittlichen Auslastung über einen bestimmten Faktor hinaus zu suchen, da für derartige Systeme a priori der Nachweis der Feasibility nicht gelingen kann).

Algorithmus 3: Binäre Suche basierendes Optimierungsverfahren für das Platzierungsverfahren mittels HySAT

Voraussetzung: Optimierungskriterium: Minimierend, Schwellwert ε_O, Gleichungssystem für Platzierung inkl. Varianzberechnung G, $init_C$ gegeben

$C_{up} := init_C; C_{lo} := 0$
$step := C_{up}$
$best := \emptyset$
while$(step \geq \varepsilon_O)$ **do**
  $G' := G \cup \{\sigma^2 < C_{up}\} \cup \{\sigma^2 \geq C_{lo}\}$
  $G'.s :=$ HySAT(G')
  if $(best = \emptyset \wedge G'.s.state = infeasible)$ **then**
    $step := 2 \cdot step$
  else
    $step := \frac{1}{2} step$
  end if
  if$(G'.s.state = infeasible)$ **then**
    $C_{lo} := C_{up}$
    $C_{up} := C_{up} + step$
  else
    $C_{up} := \frac{1}{2} G'.s.solution$
    $step = C$
    $best := G'.s.solution$
  end if
od
return *best*

Die initiale Belegung der Konstanten C ist zur Verbesserung des Laufzeitverhaltens dabei so zu wählen, dass möglichst eine gültige Lösung durch HySAT erzeugt wird. Dies ist im hier betrachteten Anwendungsszenario ein hoher Wert für die Utilisationsabweichung, der einfach z.B. mittels statischer Analysen a priori bestimmt werden kann oder aber durch eine Schätzung ermittelt wird.

5.4.2 Parallelisierung der Optimierungsmethodik

SAT-Checking mittels der in den vorangegangenen Abschnitten dargestellten prinzipiellen Vorgehensweise ist ein heuristischer Ansatz, der prinzipiell unter Komplexitätsproblemen leiden kann. Insbesondere können Erfüllbarkeitsanfragen auf nahezu identischen Formeln zu extremen Unterschieden in der Laufzeit führen. Effekte dieser Art sind in Abbildung 5.5 zu sehen.

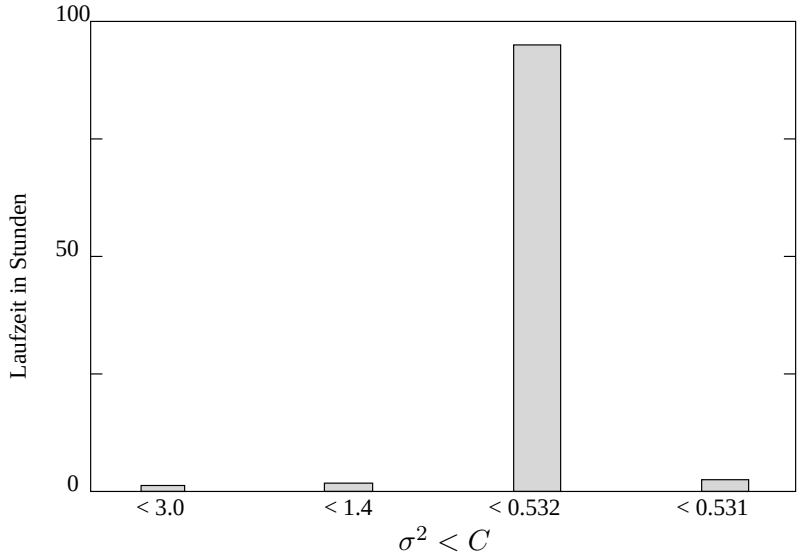


Abbildung 5.5: Stark differierende Laufzeiten von HySAT für die einzelnen Probleminstanzen (Optimierungskriterium gleichmäßige Verteilung), die jedoch im Ergebnis alle erfüllbar waren.

Das hier betrachtete Beispielsystem stellt eine Platzierungsaufgabe von 30 Tasks auf eine Architektur mit 8 Prozessorknoten, die durch einen Tokenring-Bus verbunden sind, dar. Das Optimierungskriterium wurde so gewählt, dass die Knoten möglichst gleichmäßig belastet sind. Die Verteilung wurde gemäß Algorithmus 3 durchgeführt und führte nach insgesamt etwa 106 Stunden zu einem Ergebnis. Abbildung 5.5 zeigt einen Ausschnitt aus den iterativen Aufrufen des SAT-Solvers mit einer jeweilig anders besetzten Optimierungsbedingung. Obwohl die Formeln des Problems ansonsten vollkommen identisch waren, ergeben sich doch erhebliche Unterschiede in den Laufzeiten. Besonders anschaulich ist dieses Phänomen bei den letzten beiden dargestellten Läufen: Obwohl sich die zu vergleichenden Konstanten C nur um eine Stelle unterscheiden, differieren die Laufzeiten ganz erheblich (90 Stunden im Vergleich zu 6 Stunden). Generell ist dies ein inheräntes Problem bei SAT-Solving Techniken, dessen Auswirkung sich auf der Basis der zu lösenden Modelle nicht vorhersagen lässt.
Es ist aber festzustellen, dass aus Sicht des Ergebnisses der längste Lauf für $C < 0.532$ nicht notwendig gewesen wäre, da auch $C < 0.531$ zu einem Ergebnis führte, nur mit einer deutlich geringeren Laufzeit. Da die einzelnen Läufe des iterativen Vorgehens gemäß Algorithmus 3 vollkommen unabhängig voneinander sind, bietet sich eine Parallelisierung des Verfahrens an: Ausgehend von einem Startwert für die Optimierungsfunktion kann parallel ein Bündel von Läufen ein Intervall nach einem Optimum durchsuchen. Findet eines dieser Läufe eine gültige Belegung, so können andere parallel laufende Prozesse des Platzierungsverfahren abgebrochen werden. Anschließend wird durch die neu gefundene obere oder untere Grenze ein weiteres Bündel an parallelen Entscheidungsprozeduren in dem dann neu definierten Intervall des Lösungsraumes gestartet. Ein derartiges Vorgehen könnte z.B. den sehr teuren Lauf im oben dargestellten Beispiel vermeiden, indem die Prozedur für $C < 0.531$ bereits nach einer deutlich geringeren Zeit eine erfüllende Belegung findet und da-

mit der Lauf für $C < 0.532$ abgebrochen werden kann. Abhängig von der Anzahl an parallelen Prozessen und der prinzipiellen Komplexität des Platzierungsproblems besteht somit die Hoffnung, extreme Ausreißer beim Durchsuchen des Lösungsraumes zu umgehen.

Algorithmus 4: Binäre Suche basierendes paralleles Optimierungsverfahren

$C_{up} := init_C; C_{lo} = 0; step := C_{up}; best := \emptyset$
$P_R := \bigcup_{i=1}^{N} \{G'_i\}$
for each $G'_i \in P_R$ **do**
  $G'_i.term :=$ **true**
od
while$(step \geq \varepsilon_O)$ **do**
  $P_T := \{G'_i \mid \forall G'_i \in P_R : G'_i.term =$ **true**$\}$
  $t := |P_T|$
  if $t > 1$ **then**
    $minstep := (C_{up} - C_{lo})/t$
  else
    $minstep := 0$
  end if
  for $i = 1$ **to** t **do**
    $C'_i := C_{up} - (i-1) \cdot minstep$
    $G'_i = G \cup \{\sigma^2 < C'_i\} \cup \{\sigma^2 \geq C_{lo}\}$
    fork(HySAT(G'_i))
    $G'_i.term :=$ **false**
    $P_R := P_R \cup \{G'_i\}; P_T := P_T \backslash \{G'_i\}$
  od
  wait on termination signal ts_i /* Signale sind sequentialisiert → nur ein Signal pro `wait` */
  $G'_i.term := TRUE$
  if $(best = \emptyset \wedge G'.s.state = infeasible)$ **then**
    $step := 2 \cdot step$
  else
    $step := \frac{1}{2} step$
  end if
  if$(G'.s.state = infeasible)$ **then**
    for each $G'_j \in P_R$ mit $j \neq i$ **do**
      if $C'_j \leq C'_i$ **then**
        kill(HySAT(G'_j)); $G'_j.term :=$ **true**
      end if
    od
    $C_{lo} := C'_i$; $C_{up} := C'_i + step$
  else
    for each $G'_j \in P_R$ mit $j \neq i$ **do**
      if $C'_j > G'_i.s.solution$ **then**
        kill(HySAT(G'_j)); $G'_j.term :=$ **true**
      end if
    od
    $C_{up} := \frac{1}{2} G'.s.solution$; $step := C_{up}$
    $best := G'.s.solution$
  end if
od

Algorithmus 4 zeigt das prinzipielle Vorgehen bei der parallelen Optimierung in Form eines Pseudo-Codes. Zunächst wird in C_{up} und C_{lo} die obere bzw. untere Grenze des Lösungsraumes gesetzt[1]. Dann wird durch P_R die Menge der laufenden HySAT-Prozesse definiert. Dabei wird von einer festen Anzahl an Prozessen N ausgegangen. Die einzelnen Prozesse und deren Ergebnis werden als ein Gleichungssystem-Objekt G'_i interpretiert, wobei G'_i die Gleichungen enthält, wie in Algorithmus 3, aber noch zusätzlich um weitere Attribute angereichert wird (etwa die Lösung s oder die Terminierungseigenschaft $term$). Initial werden alle Prozesse aus P_R zunächst auf "terminiert" gesetzt. Mit der `while`-Schleife wird das iterative, nach wie vor auf binärer Suche basierende, Optimierungsverfahren eingeleitet: Zunächst nimmt die Menge P_T alle Prozesse aus P_R auf, die als terminiert markiert sind. Diese sollen nun wieder gestartet werden, sich aber bzgl. der Optimierungsfunktion gleichmäßig über das Intervall des noch gültigen Lösungsraumes verteilen. Die gleichmäßige Verteilung stellt die *minstep*-Variable zur Verfügung, die das Intervall, das durch die obere und untere Grenze C_{up} und C_{lo} aufgespannt wird, in gleich große Subintervalle aufteilt. Die einzelnen Optimierungskonstanten C'_i werden anschließend in der folgenden `for`-Schleife berechnet, das Gleichungssystem um diese Vorschriften erweitert und letztlich wird ein HySAT-Prozess für jeden in der Menge der terminierten Prozess-Objekte P_T vorkommenden Objekte gestartet. `fork` übernimmt hierbei die Aufgabe, den HySAT-Prozess auf einem freien Prozessor zu dispatchen. Nach Markierung der Objekte wird auf die Terminierung eines HySAT-Prozesses gewartet (über das `wait on`-Kommando, welches ein entsprechendes Terminierungssignal empfangen kann). Da Terminierungssignale grundsätzlich sequentialisiert sind, genügt es im weiteren Verlauf eben jenes empfangene Terminierungssignal ts_i für das Prozessobjekt G'_i zu bearbeiten. Das Vorgehen ist hierbei bzgl. der neuen Grenzwerte für den Lösungsraum identisch zu Algorithmus 3. Unterschiede hingegen existieren bei der Verwaltung der Prozesse: Liefert die terminierte Anfrage keine erfüllbare Belegung des Gleichungssystems, ist eine neue untere Grenze gefunden. Folglich werden all jene HySAT-Prozesse beendet, deren Optimierungsaufgabe unterhalb dieser neuen Grenze angesiedelt war. Die Beendigung erfolgt über das `kill`-Kommando. Wird umgekehrt eine Erfüllbarkeit nachgewiesen und ein Resultat geliefert, wird dieser Wert als neuer minimaler Wert vermerkt und alle Prozesse, die oberhalb dieses Wertes nach einer Lösung suchen, werden beendet.
Alle durch diese beiden Aktionen terminierten Prozess-Objekte werden in P_R als terminierend markiert, so dass sie in der nun folgenden weiteren Iteration der `while`-Schleife erneut mit neuen Werten dispatched werden können.

Angewendet auf das in Abbildung 5.5 dargestellte Beispielsystem ergibt sich bei Verwendung von 5 parallelen Prozessen[2] eine Laufzeit von 14 Stunden (gegenüber 106 Stunden ohne Parallelisierung) für das gleiche Ergebnis der Platzierung. Insgesamt zeigen Evaluationen, dass sich im Mittel die Laufzeiten durch die angegebene Parallelisierung um etwa 50% senken lassen, bei extremen Ausreißern wie dem oben angegebenen Beispiel jedoch auch deutlich mehr.

[1] Wobei die obere Grenze in beide Richtungen im Laufe des Verfahrens variieren kann, während die untere Grenze als Startpunkt, und damit optimalsten Punkt, zunächst mit 0 angegeben wird.
[2] Zur Evaluation wurde hier eine SUN E10K mit 8 Prozessoren verwendet.

5.4.3 Mögliche Erweiterungen

Um die Geschwindigkeit der Optimierungsphase mittels der SAT-basierten Technik zu erhöhen, gibt es neben der Verbesserung der Grundstrategie des Solvers einige potentiell vielversprechende Ansätze, die im folgenden kurz skizziert werden sollen. Innerhalb des Erfüllbarkeitstests einer Instanz des Optimierungsproblems ist unter anderem die Reihenfolge der Belegung der einzelnen Variablen ausschlaggebend. Führen diese Belegungen frühzeitig zu Widersprüchen, können auch frühzeitig große Teilbäume des Belegungsbaumes der Booleschen Variablen abgeschnitten werden. Es wäre daher denkbar, die Bits gewisser Variablen, z.B. die der Allokationsvariablen `place`, früher zu belegen. Beispielsweise macht es wenig Sinn, die Variablen für die Anzahl an Preemptionen einer Task durch eine andere belegen zu wollen, wenn noch gar nicht klar ist, wohin beide platziert werden. Dies könnte potentiell die Effizienz des Verfahrens erhöhen, bedeutet aber eine problemspezifische Adaptierung der SAT-Engine.
Ein weiterer vielversprechender Ansatz zielt darauf ab, die Laufzeiten des Erfüllbarkeitstest nachfolgender Probleminstanzen im iterativen Optimierungsprozess zu beschleunigen, indem Fakten aus früheren Läufen gelernt werden und in die Gleichungsmenge der nachfolgenden Instanzen integriert werden. Dies würde dazu führen, dass invalide Belegungen in den nachfolgenden Instanzen von vorne herein ausgeschlossen werden und würde die Effizienz der binären Suche erhöhen. Dafür sind diejenigen Belegungen zu protokollieren, die unabhängig von dem konkreten Wert der Optimierungsfunktion invariant zu einer invaliden Belegung der Variablen des Gleichungssytems führen. Ferner ist hierbei zu beachten, dass nicht zu viel gelernt wird, da ansonsten die Anzahl der dadurch zusätzlich entstehenden Gleichungen wiederum effizienz-hemmend wirken. Es könnte folglich sinnvoll sein, gezielt Klauseln wieder zu vergessen (dies wird beispielsweise beim Bounded Model Checking mittels SAT-Verfahren eingesetzt, vgl. [3]).
Eine dritte Möglichkeit wäre, Teilprobleme a priori symbolisch zu lösen und dies entweder zusätzlich oder alternativ in das Gleichungssystem aufzunehmen. Eine solche Strategie würde die einzelnen Läufe der binären Suche insgesamt in ihrer Laufzeit reduzieren, bedeutet aber, dass a) das Gesamtproblem in sinnvolle Teilprobleme zerlegt werden kann und das b) eine mächtige symbolische Auswertung des Teilproblems erfolgen kann. Diese mächtige symbolische Auswertung könnte mittels der Verwendung einer BDD-Darstellung[113] des Teilproblemes erreicht werden. Die BDD-Darstellung verfügt über die hier benötigte Aussagekraft, nicht spezielle Belegungen der Variablen zu besitzen, sondern sie stellt die Gesamtheit aller Lösungen eines Booleschen Problems an sich dar. Eine Beobachtung, von der wir im nächsten Kapitel noch Gebrauch machen werden, ist, dass eine Vielzahl der Variablen einer Probleminstanz eindeutig durch eine kleine Anzahl an wesentlichen Variablen (beispielsweise die Platzierungsvariablen) in ihrer Belegung determiniert sind. Auf diese Weise könnte sich eine BDD-Darstellung eines Satzes an Gleichungen dieser Hilfsvariablen entledigen und stattdessen eine Funktion liefern, die die Feasibility nur anhand der wesentlichen Variablen darstellen kann. Hieraus könnte dann ein neuer Satz an Klauseln abgeleitet werden, der dem SAT-Verfahren hilft, entweder bestimmte Belegungen von vorne herein auszuschließen oder aber die Anzahl der Va-

riablen zu reduzieren. Ein Beispiel für ein dafür notwendiges Teilproblem könnte die Einschränkung der Feasibility durch Speicherkapazitäten auf den Knoten, oder aber sogar die Feasibility des Tasksystems durch Antwortzeiten sein. Problematisch bleibt der Aufbau der BDD-Struktur bzgl. der Komplexität; eine derartige Anwendung ist bisher nirgendwo evaluiert worden und es ist nicht klar, ob BDD-Darstellungen tatsächlich mit den hier notwendigen komplexen Formeln effizient aufgebaut werden können oder aber ob es Teilprobleme gibt, die klein genug sind, um eine Anwendung eines solchen Verfahrens zu ermöglichen.

Ein weiterer vielversprechender Ansatz ist es, den SAT-Solver nur für einen Teil der Belegungsvalidation einzusetzen und stattdessen diejenigen Anteile der Problemformulierung, die einen hohen Anteil an arithmetischen Operationen verlangen (beispielsweise die Berechnungen der Antwortzeiten), getrennt durchzuführen. Dies entspricht einem hybriden SAT-Ansatz, wie er z.B. im Bereich der Verifikation von Hybriden Automaten eingesetzt wird[58]: SAT-Solving für hybride Automaten[71] verwendet neben booleschen Constraints zusätzlich Sätze von, üblicherweise linearen, Constraints über unendlichen Datentypen. Dabei wird eine Kopplung von sogenannten Entscheidungsvariablen auf Seiten des SAT-Solvers mit den kontinuierlichen Constraints vorgenommen, die besagt, dass eine Entscheidungsvariable zu TRUE evaluiert, wenn auch das ihr zugeordnete lineare Constraint gilt. Ein Beispiel für eine derartige Kopplung könnte etwa foldendermaßen aussehen:

$$v_d \wedge \{3.5x + 17.2y - 25.0z \leq 37.8\}$$

Konjunktiv verknüpfte Sätze dieser Entscheidungsvariablen-Constraint-Paare ergeben letztlich ein Entscheidungsproblem für einen Linear-Program-Solver, wie etwa `CPLEX` oder `glpk`, der jeweils versucht, eine erfüllende Belegung für denjenigen Satz an linearen Constraints zu finden, die jeweils durch die Belegung der Entscheidungsvariablen freigeschaltet werden. Der im Rahmen dieser Arbeit verwendete Solver HySAT verfügt beispielsweise über eine derartige Kopplung an einen LP-Solver.
Wir können nun dieses Prinzip der Kopplung andersartiger Entscheidungsverfahren auch im Bereich der Platzierungsberechnung einsetzen, indem anstelle eines LP-Solvers ein spezifisches Entscheidungsverfahren für Echtzeitanalysen eingesetzt wird. Dabei können die Entscheidungsvariablen direkt mit den Platzierungsvariablen $\texttt{place}_i$ verknüpft werden, und anstelle linearer Constraints werden Informationen über die aktuelle Platzierung weitergeleitet. Das Entscheidungsverfahren für Echtzeitsysteme führt dann beispielsweise eine Antwortzeitberechnung durch und liefert die Feasibility der bis dahin erzeugten Systemkonfiguration zurück. Letztere schließlich führt dazu, bestimmte Belegungen der Entscheidungsvariablen — und damit die Belegungen der assoziierten Platzierungsvariablen — zu validieren oder zu widerlegen. Dies könnte beispielsweise in der folgenden Form in den Gleichungssystemen abgebildet werden:

$$\texttt{place}_i^p \rightarrow \{P_i = p\},$$

wobei die Platzierungsinformation hier nun explizit in Form von (Task,Prozessor)-Paaren ($\texttt{place}_i^p$) erfolgt[1] und die ehemals linearen Constraints der Informations-

[1]Zur Generierung der notwendigen booleschen Entscheidungsvariablen muss man sich hier von der logarithmischen Kodierung der Platzierungsinformation trennen.

übergabe der aktuellen Konfiguration an das Entscheidungsverfahren für Echtzeitsysteme dienen. Da SAT-Checker sukzessive Boolesche Variablen belegen, muss das proprietäre Entscheidungsverfahren sicher mit unvollständigen Belegungen umgehen können.
Auf der einen Seite lässt sich mit Hilfe eines derartigen hybriden Ansatzes ein gewaltiges Potential an Komplexität einsparen, da im Prinzip alle von der Platzierungskonfiguration direkt abhängigen Anteile der Gleichungen in das sekundäre Entscheidungsverfahren verschoben werden können. Andererseits können jedoch Widersprüche, die im kompletten SAT-Verfahren zum frühzeitigen Abschneiden eines Teilbaumes des Suchraumes führen, in der deutlich reduzierten, aber immer noch sehr komplexen, Probleminstanz für das hybride Verfahren nicht auftreten. Die Folge wäre eine Steigerung der zu untersuchenden Möglichkeiten, was naturgemäß zu einer vermehrten Komplexität bzgl. der Laufzeit des Verfahrens führt. Hier müssten folglich entsprechend sorgfältige Evaluationen zeigen, ob und für welche Teile sich ein Verschieben in die sekundäre Entscheidungsprozedur lohnt. Erste manuelle Versuche zeigen aber sehr vielversprechende Ergebnisse, die sicher weiter optimiert werden können, wenn beispielsweise zusätzliche Informationen, wie etwa das Ausnutzen von Symmetrie in Teil-Homogenen Systemen, aus dem sekundären Entscheidungsverfahren in das primäre SAT-Verfahren propagiert werden könnten.

5.5 Übertragung der Modellierung auf andere Optimierungsverfahren

Optimierungen unter Verwendung von SAT-Checkern, wie sie im letzten Kapitel beschrieben wurden, können mitunter daran leiden, dass die Optimierungsmethodik auf einer Heuristik basiert. Diese Heuristik sucht, wie eingangs dargestellt, gezielt Belegungen für die Variablen eines Gleichungssystems, wobei die Hoffnung besteht, dass die frühe Belegung bestimmter Variablen durch Widersprüche in anderen Gleichungen des Systems frühzeitig große Bereiche des Suchraumes abschneidet. Wie in Kapitel 5.6 zu sehen sein wird, funktioniert dies bis zu einer bestimmten Größe der Probleme ganz hervorragend. Nichtsdestotrotz kann es aufgrund spezifischer Eigenschaften der Problemstellungen dazu führen, dass die Optimierungsmethode unter Verwendung von SAT-Checking in akzeptabler Zeit keine Lösung finden kann. Ein gutes Beispiel für ein derartiges Ausreißer-Verhalten ist in Kapitel 5.4.2 in Abbildung 5.5 bereits dargestellt worden und lieferte den Grund für eine Parallelisierung des Optimierungsverfahrens. Zeigt sich ein derartiges Verhalten für alle Läufe des Optimierungsansatzes auf einer Problemstellung, kann die SAT-basierte Methode keine oder zumindest keine optimale Lösung finden. In solchen Fällen müsste auf andere, bekannte Verfahren für derartige Problemstellungen zurückgegriffen werden. Wie dies unter Verwendung der Modellierung aus Abschnitt 5.3.4 bis 5.3.11 geschehen kann, soll im folgenden anhand zweier weit verbreiteter stochastischer Optimierungsverfahren, nämlich Genetische Algorithmen[161] und Simulated Annealing[1], skizziert werden. Die grundlegenden Mechanismen lassen sich natürlich auch leicht auf andere, z.B. heuristische oder Branch&Bound Verfahren anwenden. Im Anschluss daran werden wir noch kurz erläutern, wie die Modellierung auf ein weiteres opti-

males Verfahren übertragbar ist, nämlich Integer Linear Programming[21].

Aufgabe eines Optimierungsverfahrens ist das Finden einer möglichst optimalen Belegung der Variablen für die in der Modellierung oben eingeführten Ungleichungssysteme. Hierzu wird die Menge der Variablen zunächst in *primäre* und *sekundäre* Variablen klassifiziert. Primäre Variablen legen die Allokation von Tasks auf Knoten und von Nachrichten auf Kommunikationsmedien fest. Eine Belegung der primären Variablen legt eindeutig eine Lösung des Optimierungsproblems fest. Sekundäre Variablen hingegen leiten sich unter Berechnungsvorschriften direkt aus den primären Variablen ab. SAT-basierte Optimierungstechniken müssen auch für sekundäre Variablen eine gültige Belegung finden, wohingegen in stochastischen oder heuristischen Verfahren die Belegung dieser Variablen aus den für sie geltenden Berechnungsvorschriften auf der Basis der Belegung der primären Variablen errechnet werden können. Sie werden also vollständig von den primären Variablen determiniert. Eine Klassifizierung der Variablen aus den Gleichungssystemen 5.5 bis 5.70 ist in Tabelle 5.1 zu ersehen.

Primäre Variablen	teilw. primäre Variablen
$\texttt{place}_i$	$\texttt{p}_i^j$
$\texttt{K}_{g_t}^k$	$\texttt{locD}_{g_t}^k$
$\texttt{Pf}_{g_t}$	
$\texttt{slot}_{dyn}^k$	
$\texttt{slot}_p^k$	
$\texttt{slotlength}^k$	
$\texttt{s}_i^k$	

Tabelle 5.1: Klassifizierung der Variablen der Gleichungssysteme des letzten Kapitels in primäre und teilweise primäre Variablen.

Alle übrigen, nicht in Tabelle 5.1 aufgeführten Variablen sind der Klasse der sekundären Variablen zuzuordnen. Teilweise primär bedeutet, dass für diese Variablen nur in bestimmten Fällen eine Belegung gesucht werden muss, je nach Kontext, der durch die primären Variablen festgelegt wird. Beispielsweise sind Belegungen für die sublokalen Deadlines von Nachrichten auf Kommunikationsmedien $\texttt{locD}_{g_t}^k$ nur dann von Bedeutung, wenn die Allokation tatsächlich die Benutzung mehrerer Kommunikationsmedien verlangt. Ähnliches gilt auch für die Priorität der Tasks, die nur im Falle von Deadlinegleichheit nicht durch die primäre Belegung determiniert wird.
$\texttt{pm}_{g_j}^{g_i}$ ist explizit auch als sekundäre Variable zu behandeln, da Nachrichtendeadlines — im Unterschied zu Taskdeadlines — sich aufgrund einer Platzierung nicht verändern können. Es kann daher eine Priorisierung der Nachrichten gemäß der Deadlinemonotonen Strategie vergeben werden. Wie in Anhang B nachgewiesen, ist bei Gleichheit der Deadlines die Priorisierung der Nachrichten mit gleichen Deadlines unerheblich, so dass auch dieses vorab bei der Erstellung des Problems determiniert werden kann.
$\texttt{K}_{g_t}^k$ und $\texttt{Pf}_{g_t}$ sind natürlich nicht so unabhängig zu betrachten, wie es die Tabelle suggeriert. Entsprechend den Vorschriften in Abschnitt 5.3.5 sind sie wechselseitig abhängig und diese Abhängigkeit muss bei der Belegung berücksichtigt werden.

$\texttt{locD}^k_{g_t}$ ist in den obigen Gleichungen frei innerhalb eines vorgegebenen und durch das Problem beschriebenen Wertebereiches wählbar, wobei sehr kleine Änderungen nahezu keinen Effekt auf die Güte des Ergebnisses der Platzierungsberechnung haben. Im Zuge der Benutzung durch heuristische oder stochastische Verfahren sollte deshalb für die Synthese der sublokalen Deadlines eine Quantisierung des möglichen Bereiches erfolgen, so dass das Optimierungsverfahren den Wertebereich der Variablen in diskreten Intervallen durchsuchen kann. Als Konsequenz müssen beispielsweise Gleichungen 5.28 und 5.29 um einen Quantisierungsfaktor $\hbar$ erweitert werden:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \bigwedge_{\forall k \in K} (\texttt{K}^k_{g_t} \rightarrow \texttt{locD}^k_{g_t} \cdot \hbar \geq \omega_k(g_t)) \tag{5.72}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall g_t \in \gamma_i} \left(\sum_{k \in K} \texttt{locD}^k_{g_t} \cdot \hbar + \texttt{serv}_{g_t} \leq \Delta_\gamma(g_t) \right) \tag{5.73}$$

Das jeweilige Optimierungsverfahren muss folglich nur noch einen deutlich reduzierten Wertebereich abdecken. Die Wahl der Größe des Quantisierungsfaktors $\hbar$ ist in entsprechenden Evaluationen auf einem der Probleminstanz angemessenen Wert zu bestimmen.

In den folgenden beiden Abschnitten soll nun anhand zweier stochastischer Optimierungsverfahren demonstriert werden, wie die Bestimmung einer Belegung der primären Variablen prinzipiell erfolgt. Alle übrigen sekundären Variablen sind gemäß ihren Berechnungsvorschriften aus den Gleichungen 5.5 bis 5.70 zu errechnen und werden daher im folgenden nicht weiter betrachtet.

5.5.1 Simulated Annealing

Die prinzipielle Idee hinter dem Simulated Annealing-Verfahren wurde bereits in Kapitel 5.1 erläutert. Eine Übertragung der Modellierung des Verteilungsproblems auf dieses Verfahren beschränkt sich darin, die primären Variablen, wie sie in Tabelle 5.1 aufgelistet sind, bestimmen zu lassen. Wir haben dies in [178] bereits für einfache, nicht hierarchische Systeme durchgeführt und dort auch die Variante Localized Simulated Annealing behandelt, wollen das hier jedoch allgemeiner ohne Angabe einer speziellen Algorithmenvariante motivieren. Dabei wird zunächst mit einer beliebigen Startkonfiguration begonnen und anschließend wird entsprechend der Annealing-Strategie eine zufällige Nachbarschaftskonfigurationen erstellt. Nachbarkonfigurationen sind zunächst Veränderungen an *einer* der primären Variablen, d.h. es wird entweder eine Task auf einen anderen Knoten verschoben, eine andere Pfadhülle gewählt oder beispielsweise der Slot einer Nachricht gewechselt. Dabei ist zu beachten, dass einige primäre Variablen nicht unabhängig von anderen sind (z.B. Pfadhülle und gewähltes Kommunikationsmedium). Zudem sind gewisse Eigenschaften einzuhalten, beispielsweise muss aufgrund von Änderungen der Priorität eines einzelnen Taskpaares trotzdem die Transitivität der Prioritätsrelation erhalten bleiben. Dies hat unter Umständen zur Folge, dass eine Nachbarschaftskonfiguration sich nicht nur in einer primären Variable unterschieden kann, sondern sich Änderungen auch auf weitere primäre Variablen auswirken. Alle sekundären Variablen

hingegen ergeben sich zwangsläufig aus der Belegung der primären. Welche der primären Variablen in der Berechnung einer Nachbarkonfiguration gewählt wird, sollte zufällig bestimmt werden, wobei dies auch von der aktuellen Konfiguration abhängt, da diese gewisse primäre Variablen aufgrund der durch die Konfiguration gegebenen Platzierung unveränderlich hält (beispielsweise können lokale Kommunikationen keine andere Pfadhülle wählen).

Die Energiefunktion, die letztlich der Optimierungsgüte entspricht, ist dabei abhängig vom Optimierungskriterium zu wählen. Allerdings darf sie sich nicht ausschließlich, wie bei der oben dargestellten SAT-Methodik, auf das Optimierungskriterium an sich abstützen, sondern muss zusätzlich noch die Validität der Konfiguration quantifizieren. Ein übliches Mittel hierfür ist die Bestrafung von invaliden Konfigurationen durch eine höhere Energie. Da die Invalidität einer Konfiguration nicht nur von einem Teil, sondern durchaus von mehreren Teilen der Konfiguration abhängt, sollte hier eine gestaffelte Bestrafung pro verletzender Bedingung gewählt werden. Beispielsweise kann jede Task, deren Antwortzeit über der Deadline liegt, mit einem Straf-Energieanteil in die Gesamtenergie der Optimierungsfunktion eingehen. Auf diese Weise sinkt die Gesamtenergie einer Konfiguration, wenn weniger Teilbedingungen nicht erfüllt sind. Dies lässt sich auch auf andere Bedingungen, wie beispielsweise verbotene Platzierungen etc. übertragen. Für eine detailliertere Betrachtung sei hier jedoch auf [178] verwiesen.

5.5.2 Genetische Algorithmen

Auch dieses Verfahren wurde in Kapitel 5.1 bereits eingehend beschrieben, so dass wir uns hier im wesentlichen auf die Kodierung der Information beschränken können. Ein Individuum einer Population entspricht dabei einer Konfiguration. Die Lebensfähigkeit eines Individuums entspricht wiederum seiner Energiefunktion, die analog der im Simulated Annealing Abschnitt beschriebenen Charakteristika gebildet werden kann und ebenfalls invalide Lösungen, also nicht-lebensfähige Individuen, bestraft. Ein entscheidendes Merkmal Genetischer Algorithmen ist die sorgfältige Auswahl einer geeigneten Gen-Struktur. Da einfachere Varianten des Platzierungsproblems bereits mit Genetischen Algorithmen erfolgreich behandelt wurden[130, 171], können wir die dort gewählte Genstruktur auf das hier betrachtete Problem adaptieren: Jede primäre Variable mit Ausnahme der Pfadhüllen wird zunächst in einem einzelnen Gen kodiert. Die einzelnen Werte sollten dabei einer Gray-Kodierung folgen, um eine möglichst gleichmäßige Mutation zu erlauben. Die Wegewahl von Nachrichten zwischen zwei Tasks erfolgt hierbei ausschließlich über die Interpretation der Kommunikationsmedien-Variablen `K`. Dies ermöglicht auf der einen Seite eine gleichmäßige Mutationsrate, erzeugt aber auf der anderen Seite Konfigurationen, die nicht gültig sind. Diesem Umstand muss in der Energiefunktion durch entsprechend anzusetzende Straf-Energieanteile Rechnung getragen werden. Die Gesamtheit der primären `K`-Variabeln sollte hierbei in einem einzigen Gen kodiert werden, um unsinnige Varianten der Kommunikationspfade bei der Crossover-Operation zu verhindern und nur den Einfluß der Mutation geltend zu machen. Strafen für nichtmögliche Pfade sollten entsprechend hoch in der Energiefunktion berücksichtigt werden, so dass nur

sinnvolle Mutationen durchkommen, allerdings auch nicht zu hoch, da ansonsten der Wechsel von Tasks auf andere Knoten durch die dann möglicherweise invaliden Kommunikationspfade verhindert werden würde. Diese Diskussion zeigt schon hier, dass die Wahl der Informationskodierung im Bereich der Wegewahl sehr empfindlich ist und sorgsam in praktischen Versuchen evaluiert werden muss. Sowohl [130] als auch [171] können hier nur bedingt einen Hinweis auf die passende Genstruktur geben, da sie für komplexere Architekturen keine generellen Aussagen über die Leistungsfähigkeit ihrer Kodierung machen können.
Ähnliche Beobachtungen können hinsichtlich der Priorisierung der Tasks gemacht werden, da auch hier das Ändern eines einzelnen Elementes der Prioritätsrelation unter Umständen die transitiven Eigenschaften jener Relation zerstört und dies mit einer entsprechenden Strafenergie auszustatten ist.

5.5.3 Integer Linear Programming

Integer Linear Programming (vgl. ebenfalls Kapitel 5.1) kann aufgrund seiner bekannten Nichtanwendbarkeit bei nicht-linearen Constraints nur in solchen Probleminstanzen eingesetzt werden, in denen nur lineare Constraints entstehen. Für das Platzierungsproblem bedeutet dies, dass es eingesetzt werden kann, solange die Architektur keine Zeit-basierten Bussysteme enthält; Andernfalls muss ein alternatives Optimierungsverfahren benutzt werden.
Im Rahmen des Sonderforschungsbereiches/Transregio AVACS[132] haben wir eine Modellierung des Platzierungsproblems für Integer Linear Programming für einfache Systeme (homogene Architektur, nur ein Kommunikationsmedium) erarbeitet[166], die in ein Branch & Bound-Verfahren eingebettet eine Optimierung auf Basis von Linear Programming durchführt. Die wesentliche Idee hierbei ist, zunächst eine relaxierte Lösung (auch ganzzahlige Variablen werden als rellwertig behandelt) zu berechnen. Anschließend werden gemäß einer geschickt zu wählenden Rundungsheuristik die primären ganzzahligen Variablen (vgl. Tabelle 5.1) auf den ganzzahligen Bereich verschoben. Die Ergebnisse aller sekundären Variablen werden verworfen und stattdessen gemäß ihren Gleichungen offline, d.h. nicht im Linear Programming-Schritt, neu berechnet und die Lösung auf Feasibility getestet. Dieser Schritt stellt eine Entscheidung in einem Branch & Bound Framework dar, mit dem Teile des Lösungsraumes abgeschnitten werden können, indem die Gültigkeit oder Nichtgültigkeit der ganzzahligen Lösung bestimmte Belegungskombinationen der primären Variablen ausschließen. Gleichzeitig werden aus Symmetriegründen weitere sogenannte Cutting Planes eingefügt, die den Lösungsraum ebenfalls verkleinern können. Dieses Verfahren wird solange fortgeführt, bis das Branch & Bound Framework terminiert. Die beste bis dahin gefundene Lösung ist dann die optimale Lösung.
Die linearen Ungleichungssysteme, die das Platzierungsproblem charakterisieren, können im Prinzip vollkommen analog den Ungleichungen für das SAT-basierte Verfahren entnommen werden. In heterogenen Systemen kann allerdings nicht so häufig auf die gewinnbringenden Cutting Planes zurückgegriffen werden, da sie im wesentlichen Symmetrie ausdrücken: Ist eine bestimmte Menge von Tasks auf einem Prozessorknoten infeasible, so ist sie dies auch auf allen anderen Knoten. Dieses Beispiel nutzt aus, dass es sich bei der Architektur um eine homogene Architektur

handelt. Auf die hier betrachteten Problemklassen lässt sich dies so ohne weiteres nicht übertragen; So wäre obige Aussage aufgrund unterschiedlicher WCET-Werte einer Task auf unterschiedlichen Prozessorknoten schlicht falsch.
Ein wesentlicher Unterschied zur Modellierung in Kapitel 5.3.4 liegt in der Modellierung der primären `place`-Variablen. Die Verwendung ganzzahliger Variablen bringt es mit sich, dass diese — wie oben dargestellt — mit einer Rundungsfunktion aus einer relaxierten Lösung auf eine ganzzahlige Belegung gebracht werden müssen. Die Beschaffenheit des Platzierungsproblems induziert im Grunde eine eindeutige Aussage, ob eine Task auf welchem Knoten platziert wurde. Es ist daher aus Sicht der Rundungsfunktion leichter, diese Platzierungsinformation als 0-1-Wert zu betrachten. Dementsprechend wurde in [166] auch ein solcher Ansatz gewählt, mit dem Nachteil, dass nun die Skalierung bzgl. der Größe der Architektur schwieriger ist, weil eine quadratische Abhängigkeit der Anzahl an Variablen von der Architekturgröße und keine logarithmische, wie beim SAT-Verfahren, zu generieren ist. Auf der anderen Seite ermöglicht die 0-1-Kodierung es, deutlich einfacher Cutting Planes zu integrieren (vgl. wiederum [166]). Inwieweit dieses Verhalten auch für andere primären Variablen zutrifft, ist zum jetzigen Zeitpunkt noch nicht zu sagen und bedarf eingehender Evaluationen.

5.6 Experimentelle Ergebnisse

Nachdem nun die Tranformation des Platzierungsproblems für heterogene, hierarchische eingebettete Echtzeitsysteme auf eine Entscheidungsprozedur für Erfüllbarkeitstests abgebildet wurde und auch die Einbindung in einen Entwicklungsprozess inklusive der Prinzipien und Bewertung der dafür notwendigen Deadline-Synthese beschrieben ist, soll im folgenden der Ansatz durch experimentelle Ergebnisse und Komplexitätsanalysen auf seine Anwendbarkeit hin untersucht werden. Dazu werden wir schrittweise die Komplexitätsklassen der zu platzierenden Systeme erhöhen bzw. unterschiedliche Sichten analysieren. Die Ergebnisse dieses Abschnitts ergeben zusammen mit den Messungen aus den Kapiteln 4.4 und 4.5.3 einen Eindruck von der Performanz und Leistungsfähigkeit des Platzierungsverfahrens.
Zunächst wird der auf SAT-Checking basierende Ansatz auf seine Anwendbarkeit hin untersucht, indem ein Vergleich mit einem in der Community gut bekannten Benchmark geführt wird. Anschließend werden die dort verwendeten einfachen Architekturen auf hierarchische Topologien von Kommunikationsnetzwerken erweitert, die die Nützlichkeit der Pfadhüllen zeigen sollen. Skalierungsmessungen und eine quantitative Analyse der Komplexität des Verfahrens schließen sich an, bevor wir zunächst den Ansatz der inkrementellen Integration evaluieren und anschließend demonstrieren, dass das Verfahren auch in heterogenen Architekturen anwendbar ist und diverse Busprotokolle unterstützt.

5.6.1 Vergleich mit bekannten Verfahren

Zunächst soll die auf SAT-Checking basierende Platzierungsmethodik, wie sie in dieser Arbeit entwickelt wurde, mit einem sehr bekannten Benchmark verglichen

werden. Hieraus wird sich die Anwendbarkeit des Verfahrens ergeben, und gleichzeitig kann auch die Optimalität der erzeugten Lösungen dargestellt werden.
Der Benchmark, um den es sich bei der vergleichenden Evaluation im folgenden handelt, besteht aus einer Taskmenge und einer einfachen Architektur. Er wurde in den frühen neunziger Jahren an der Universität York entwickelt und mittels eines Simulated Annealing-Ansatzes einer Platzierungsoptimierung unterworfen (vgl. [183]). Die Taskmenge besteht aus 43 Tasks, die in unterschiedlichen Graden untereinander durch Kommunikationen verbunden sind. Tabelle 5.2 zeigt das vollständige Tasksystem mit allen aus Echtzeitsgesichtspunkten wichtigen Eckdaten.

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
0	60	4	3000	$50 \rightarrow 1, 150 \rightarrow 2$	0
1	60	4	1500	$60 \rightarrow 3, 70 \rightarrow 4, 30 \rightarrow 5$	
2	60	2	1200	$20 \rightarrow 3$	
3	60	2	1700		1
4	60	2	3000	$60 \rightarrow 6$	
5	60	4	3000	$80 \rightarrow 6$	
6	60	6	1100		2
7	35	2	500	$40 \rightarrow 8$	1
8	35	2	700		1
9	35	8	900	$90 \rightarrow 11$	0
10	35	14	2200	$250 \rightarrow 11$	
11	35	4	1000		1
12	14	2	1000	$150 \rightarrow 13, 150 \rightarrow 14$	2
13	14	2	1500	$50 \rightarrow 15$	
14	14	2	1600		
15	14	2	1300		3
16	14	2	1100	$50 \rightarrow 17$	3
17	14	2	1000		2
18	35	1	1000	$50 \rightarrow 19$	1
19	35	1	1600		1
20	14	1	1900	$40 \rightarrow 21$	
21	14	2	2000		3
22	14	1	1000	$40 \rightarrow 23$	
23	14	1	2000	$40 \rightarrow 24$	
24	14	1	1000	$20 \rightarrow 25$	
25	14	1	2000	$20 \rightarrow 26$	
26	14	2	7000	$20 \rightarrow 27, 20 \rightarrow 28$	
27	14	1	1100	$50 \rightarrow 29$	
28	14	1	900	$30 \rightarrow 29$	
29	14	1	500		6
30	14	1	600	$50 \rightarrow 31$	7
31	14	2	800	$70 \rightarrow 32$	
32	14	2	1300		7
33	20	3	1000	$50 \rightarrow 35$	2,3
34	20	2	1000	$50 \rightarrow 35$	0,1
35	20	2	1000	$60 \rightarrow 36, 60 \rightarrow 37$	
36	20	2	1000		6,7
37	20	2	1000		
38	20	3	1000	$50 \rightarrow 40$	2,3
39	20	2	1000	$50 \rightarrow 40$	0,1
40	20	2	1000	$60 \rightarrow 41, 60 \rightarrow 42$	
41	20	2	1000		6,7
42	20	2	1000		

Tabelle 5.2: Original Benchmark aus [183], wobei alle Taskketten lose gekoppelt sind, d.h. End-to-End-Deadlines sind nur jeweils über einer Task und deren Nachrichten definiert.

Taskketten, die durch Kommunikationen zwischen Tasks gebildet werden, sind hier-

bei *nicht* als Taskketten im klassischen Sinne, also mit End-to-End-Deadlines, zu interpretieren, sondern stellen vielmehr ein lose gekoppeltes System dar. Diese lose Kopplung besagt, dass alle Tasks periodisch sind und nicht durch eingehende Nachrichten angestoßen werden. Stattdessen wird nur erwartet, dass eine eingehende Nachricht nicht älter als maximal eine frühere Invokation der empfangenden Task sein darf. Demzufolge sind End-to-End-Deadlines hier so aufzubauen, dass sie gerade die sendende Task plus deren Nachricht innerhalb einer Periode (der sendenden Task) umfassen darf. Es sind also weder Jitter noch preemptionsfreie Taskfenster zu betrachten, und auch eine Deadlinesynthese ist sehr einfach möglich.
Jede Task enthält weitere Randbedingungen, die bei der Platzierung zu beachten sind: Redundanztasks und Speicherverbrauch. Der jeweilige Speicherverbrauch und Einschränkungen bei der erlaubten Platzierung sind Tabelle 5.2 zu entnehmen, die Redundanztasks sind in Tabelle 5.3 dargestellt.

Original	Duplikat
33	38
34	39
35	40
36	41
37	42

Tabelle 5.3: Redundanzen des Beispiels aus [183], die nicht zusammen auf einem Prozessorknoten allokiert werden dürfen.

Die Zielarchitektur des Benchmarks besteht aus einem homogenen Prozessorknotennetzwerk mit 8 Knoten, die gemeinsam an einem Tokenring-Bus angeschlossen sind. Letzterer ist so spezifiziert, wie die Modellierung eines solchen Kommunikationsmediums in Kapitel 27 dargestellt ist: Nachrichten werden grundsätzlich als Ganzes verschickt und dazu noch gemeinsam mit allen anderen auf einem Knoten anfallenden Nachrichten innerhalb einer Sendeberechtigung. Hierzu muss, wie in 27 bereits dargelegt, die maximale Umlaufzeit des Tokens kleiner der kleinsten Periode einer sendenden Task sein. Weitere Protokollanteile sind in dem Bussystem aus [183] nicht spezifiziert und werden auch hier nicht betrachtet.
Verschiedene Größen an Speicherkapazitäten bilden das einzige Unterscheidungsmerkmal der Prozessorknoten; sowohl Prozessortyp als auch sämtliche weitere Architekturparameter sind ansonsten identisch, so dass in Tabelle 5.2 nur eine einheitliche WCET verwendet werden muss. Die unterschiedlichen Ausstattungen an Speicher finden sich in der Tabelle 5.4.

In [183] wurden zwei Untersuchungen mit jeweils unterschiedlichen Optimierungszielen vorgenommen. Dabei kam ein auf Simulated Annealing basierendes Verfahren zum Einsatz, für das die Parameter entsprechend an das Problem angepasst waren. Eine möglichst minimale Token Rotation Time war das Optimierungsziel der ersten Untersuchung, die eine Übertragungsrate von 90 bytes/ms auf dem Bus vorraussetzt. Optimierungsziel des zweiten Versuches hingegen war eine möglichst gleichmäßige Auslastung auf allen Prozessorknoten, wobei eine Optimierungsmethodik zum Ein-

Prozessor	Speicher
0	10000
1	10000
2	10000
3	12000
4	7000
5	7000
6	12000
7	10000

Tabelle 5.4: Kenndaten für die Prozessorknoten der Architektur aus [183].

satz kommt, wie sie in Kapitel 5.4.1 dargestellt wurde. Aufgrund des sehr engen Lösungsraumes für valide Platzierungen bei einer Busübertragungsrate von 90 bytes/ms wurde hier die Geschwindigkeit des Busses erhöht und die Rate auf 250 bytes/ms eingestellt. Sowohl die Ergebnisse aus [183] als auch die Vergleichswerte derselben Experimente mittels des in dieser Arbeit vorgestellten HySAT-basierenden Ansatzes sind in Tabelle 5.5 aufgelistet.

Experiment	Ergebnis lt. [183]	Ergebnis HySAT	Laufzeit HySAT
[183] mit $90\frac{bytes}{ms}$	$TRT = 8.67ms$	$TRT = 8.55ms$	47min
[183] mit $250\frac{bytes}{ms}$	$\sigma^2 = 51.84$	$\sigma^2 = 27.66$	107min
[183] mit CAN	n.a.	$U_{CAN} = 0.371$	361min

Tabelle 5.5: Vergleich zwischen den Ergebnissen aus [183] und einer Optimierung mittels HySAT. Zusätzlich wurde im letzten Experiment statt des in [183] verwendeten Tokenringes ein CAN-Bus eingesetzt.

Anhand der Ergebnisse in Tabelle 5.5 ist zu erkennen, dass das auf Simulated Annealing basierende Verfahren in [183] trotz der relativen Einfachheit dieses speziellen Platzierungsproblemes nicht in der Lage war, die optimale Lösung zu finden (vgl. erste Zeile der Tabelle). Der SAT-basierte Ansatz hingegen findet das Minimum der Token Rotation Time für das angegebene System in unter einer Stunde. Gleiches ist beim zweiten Versuch, die Auslastung des Systems möglichst gleichmäßig über alle Prozessorknoten zu verteilen, zu beobachten[1]. Aussagen über die Laufzeiten in [183] liegen leider nicht vor, doch es ist zu erwarten, dass diese jeweils deutlich unter denen des HySAT-Verfahrens liegen dürften[2]. Zusätzlich zu den beiden ersten Experimenten wurde im letzten Versuch der Tokenring durch einen für physikalische Systeme realistischeren CAN-Bus ersetzt. Optimierungsziel in diesem Fall ist eine möglichst minimale Buslast und auch in diesem Experiment konnte nach einer etwas längeren Berechnungszeit eine optimale Lösung gefunden werden. Ähnliche Evaluationen haben die Autoren von [183] leider nicht mehr unternommen, obwohl sie die Formalisierung und Modellierung des zeitlichen Verhaltens von CAN-Bussen

[1] Bzgl. der Bedeutung der Varianz σ^2 siehe die Ausführungen in Kapitel 5.4.1 auf den Seiten 158 und folgende.

[2] Untersuchungen in [178] an einer sehr ähnlichen Probleminstanz zeigen dies.

u.ä. durchaus vorangetrieben haben (vgl. z.B. [181]).

Beispielhaft für komplexere Tasksysteme soll die folgende Taskmenge die Anwendbarkeit auch auf Applikationen mit End-to-End-Deadlines und Jitter-Effekten zeigen. Das hier betrachtete Tasksystem besteht aus:

- 30 Tasks, keine Redundanzen
- 10 Taskketten
  - maximal 3 Nachfolge-Tasks einer Task innerhalb der Taskkette
  - maximale Länge der Taskketten von 5

Eine genaue Aufstellung dieses Tasksystems ist in Anhang E.1 zu finden. Entscheidender Unterschied zu dem in [183] benutzten Beispielsystem ist die Verwendung von Taskketten mit einer strengen Kopplung, d.h. versendete Nachrichten werden als Aktivierungsereignis der Empfangstasks genutzt. Dies hat zur Folge, dass sowohl End-to-End-Deadlines als auch die damit einhergehende Deadline-Synthese sowie die Behandlung von preemptionsfreien Taskfenstern zum Einsatz kommen. Zudem führt die Triggerung von Tasks durch Nachrichten zu Jitter-Effekten, die ebenfalls berücksichtigt werden müssen.
Als Zielarchitektur soll die oben und in [183] beschreibene Topologie aus 8 Prozessorknoten an einem Tokenring betrachtet werden. Das Tasksystem enthält für diese Architektur spezifische Platzierungseinschränkungen: 70% der Tasks dürfen jeweils nur auf einem Sub-Cluster von 4 Prozessorknoten platziert werden, die übrigen 30% sind nahezu frei verteilbar. Mit einem Optimierungskriterium zur Minimierung der Token Rotation Time liefert das SAT-basierte Platzierungsverfahren auch in diesem Beispiel die optimale Lösung nach einer Berechnungszeit von 14 Minuten (mit einem Ergebnis von $TRT = 1.3ms$).

Insgesamt zeigen die oben angegebenen Messungen, dass die in dieser Arbeit vorgestellte Methodik auf der Basis von Entscheidungsprozeduren der Erfüllbarkeit hinlänglich bekannten heuristischen oder stochastischen Verfahren überlegen ist. Da weitergehende Messungen für die heuristischen/stochastischen Verfahren (hier insbesondere Simulated Annealing; aber Gleiches gilt auch beispielsweise für Genetische Algorithmen) insbesondere für komplexere Architekturen nicht vorliegen, kann hieraus keine weitreichendere Schlussfolgerung gezogen werden. Allerdings ist zu erwarten, dass die heuristischen/stochastischen Verfahren sich hierbei schwerer tun werden, da sie zum einen maßgeblich von der Wahl der Eingabeparameter, die üblicherweise sehr problemspezifisch sind, abhängen und auf der anderen Seite komplexere Architekturen beispielsweise die Bestimmung geeigneter Nachbarschaftsfunktionen[1] deutlich erschweren. Die steigende Komplexität wird sich also nicht in erster Line, wie bei dem SAT-basierten Ansatz, in der Laufzeit der Optimierungsmethodik niederschlagen, sondern auf die Güte der gefundenen Lösung. Die in dieser Arbeit vorgestellte Optimierungsmethode kann hingegen durch ein geeignetes zu wählendes

[1] Nachbarschaftsfunktionen sind in Verfahren, wie dem Simulated Annealing notwendig, um temporäre Zwischenlösungen zu verlassen. Dazu ist ein möglichst stetiger Lösungsraum vorteilhaft[1].

Unschärfe-Intervall bzgl. der optimalen Lösung zur Einschränkung der Laufzeiten die Güte der Lösung kontrolliert absenken (vgl. Algorithmus 4). Wir werden diese Eigenschaft in der Behandlung komplexer Architekturen in den folgenden Kapiteln noch nutzen, um die Berechnungszeit des Verfahrens zu senken.

5.6.2 Hierarchische Bus-Topologien

Die Untersuchungen im Abschnitt 5.6.1 zeigen zwar die grundsätzliche Anwendbarkeit und Performanz bzgl. Lauffzeiten und Güte des Ergebnisses, jedoch sind dort nur sehr einfache Architekturen betrachtet worden. Dies soll nun in diesem Abschnitt im Sinne der in Kapitel 3.1 vorgestellten typischen Architekturen komplexer eingebetteter Echtzeitsysteme erweitert werden. Wie dort schon beschrieben, sind die vorkommenden Topologien im wesentlichen auf hierarchisch angeordnete Bus-Strukturen zu reduzieren, in denen Unteräste eines Architekturbaumes über Gateway-Knoten an zentralere Backbone-Bussysteme angekoppelt sind. Zur Erfassung des Effektes von komplexeren, hierarchisch organsierten Systemen auf sowohl die Laufzeit- als auch die Ergebnis-Eigenschaften werden im folgenden die aus dem letzten Kapitel bekannten Benchmarksysteme erweitert. Hierzu werden zusätzliche Bussysteme und Knoten hinzugenommen, wie dies in Abbildung 5.6 in den Architekturen A bis C dargestellt ist.

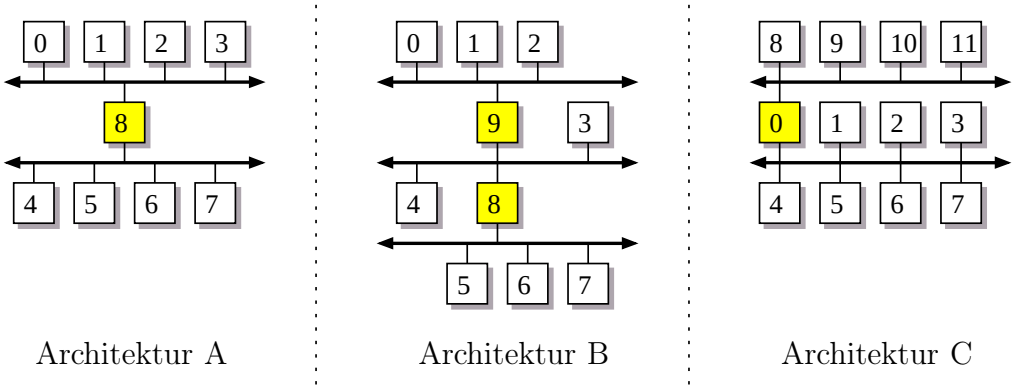


Abbildung 5.6: Verschiedene hierarchisch organisierte Steurgerätenetzwerke. In Architektur A und B verfügen die Gateway-Knoten 8 und 9 über keinen Speicher, in Architektur C hingegen ist Gateway-Knoten 0 in seinen Eigenschaften aus [183] belassen. Als Bussysteme wird durchgängig ein Tokenring gemäß [183] eingesetzt.

Architektur A wie auch Architektur B in Abbildung 5.6 erweitern die Steuergerätearchitektur aus [183], indem die einfache Struktur aufgebrochen und eine Anzahl der Prozessorknoten neu eingeführten Bussystemen zugeordnet werden. Dazu wird ein hierarchischer Baum aufgebaut, an dessen Nahtstellen neue Gateway-Knoten (Prozessorknoten 8 und 9 in Abbildung 5.6) eingesetzt werden. Um nur eine Gateway-Funktion erfüllen zu können, sind diese Knoten mit keinerlei Speicher ausgestattet, so dass das Optimierungsverfahren nicht in der Lage ist, Tasks auf diesen Knoten anzusiedeln[1]. Alle übrigen Prozessorknoten haben die gleichen Eckdaten wie in Ta-

[1]Damit bleibt das ursprüngliche System in seiner Eigenschaft, nur wenige valide Lösungen

belle 5.4 auf Seite 174 beschrieben. Zusätzlich eingeführte Bussysteme sind jeweils wiederum vom Typ Tokenring.
Architektur C in Abbildung 5.6 erweitert die in Kapitel 5.6.1 dargestellte Architektur durch Hinzunahme weiterer Knoten an einem zweiten Bussystem vom Typ Tokenring. Letzteres ist über Knoten 0 an die ursprüngliche Architektur angekoppelt und verändert keine weiteren Eigenschaften dieses Knotens.

Als Optimierungsfunktion für eine Platzierungsberechnung mittels des auf HySAT basierenden Verfahrens wurde die Minimierung der Summe der Token Rotation Times auf allen angeschlossenen Bussystemen gewählt. Dies entspricht gemäß dem Modell in Kapitel 27 der schlimmsten Latenzzeit einer Nachricht, die über alle Bussysteme versendet werden muss. Bei der Optimierung, deren Ergebnisse in Tabelle 5.6 dargestellt sind, kam das in [183] beschriebene Tasksystem zur Anwendung.

Experiment	Ergebnis	Laufzeit	Güte
Architektur A + [183]	$\sum TRT_i = 10.77ms$	490min	99%
Architektur B + [183]	$\sum TRT_i = 16.32ms$	740min	97%
Architektur C + [183]	$\sum TRT_i = 8.55ms$	790min	100%
Architektur C + [183]*	$\sum TRT_i = 8.55ms$	540min	100%

Tabelle 5.6: Ergebnis der Platzierung mittels HySAT unter einer gegebenen Güte des Ergebnisses in Prozent.

Aufgrund der Erhaltung des eigentlichen Platzierungsproblemes aus [183], jedoch um Erweiterung durch hierarchisch zu nutzende Bussysteme, erwächst die Notwendigkeit, tatsächlich mehrere Bussysteme nutzen zu müssen. Dies ist in den ersten zwei Versuchen aus Tabelle 5.6 ersichtlich, da Nachrichten, die mehrere Bussysteme nutzen, auf jedem dieser Busse eine Last erzeugen, die sich in der Summe der TRTs der Tokenringe wiederfindet. Die unterschiedlichen Güten der Ergebnisse wurden eingeführt, um die Berechnungszeit des Optimierungsverfahrens zu begrenzen und entsprechen dem Unterschied des größten gefundenen Wertes der Optimierungsfunktion, die nicht erfüllbar ist und der kleinsten erfüllbaren Belegung der Variablen der Optimierungsfunktion. Wie zu erwarten war, ist die Laufzeit stark abhängig von der Anzahl der verwendeten Kommunikationsmedien. Dies liegt an der zusätzlichen Anzahl an Pfadhüllen, die das Optimierungsverfahren pro Nachricht berücksichtigen muss (vgl. Kapitel 5.3.5 auf den Seiten 137 und folgende). Gleichwohl zeigen diese Messungen, dass auch komplexe Architekturen innerhalb einer erträglichen Zeitspanne berechnet werden können, wenn denn geringe Abstriche bei der Optimalität des Ergebnisses gemacht werden.
Optimierungen des angegebenen Tasksystems auf der Architektur C in Abbildung 5.6 sind in den letzten beiden Spalten in Tabelle 5.6 zu finden. Hier ist das Ergebnis ein anderes als auf Architektur A und B, da zusätzliche Knoten hinzugekommen sind, ansonsten aber die ursprüngliche Architektur als Teil des Systems erhalten geblieben ist. Platzierungsrestriktionen, wie sie in Tabelle 5.2 auf Seite 172 angegeben

hervorzubringen, erhalten.

wurden, bleiben auch hier gültig, so dass 20 der 43 Tasks des Systems auch auf den Knoten 8 bis 11 platziert werden können. Wiederum wurde als Optimierungsfunktion die Minimierung der Summe aller Token Rotation Times angegeben. Das Ergebnis entspricht dem des ursprünglichen Versuches aus Kapitel 5.6.1. Dies liegt auf der Hand, da ein Verschieben einer Task auf einen der Prozessorknoten 8 bis 11 auf dem zweiten Tokenring eine zusätzliche Last erzeugen würde, die die Summe der TRTs erhöhen würde. Demnach ist die Optimierungsfunktion minimal, wenn die gleiche Taskanordnung erzeugt wird, die auch ohne den zusätzlichen Tokenring und an ihm hängende Prozessorknoten erzeugt wird.
Im letzten Versuch in Tabelle 5.6 wurde zusätzlich die Hälfte der 20 frei platzierbaren Tasks dahingehend eingeschränkt, dass nur eine Platzierung auf den Knoten 0 bis 7 (also dem ursprünglichen Anteil der Architektur) erlaubt wurde. Das Ergebnis ist naheliegenderweise identisch. Allerdings wird hier nochmals die Abhängigkeit der Laufzeit von der Anzahl der potentiell benutzbaren Kommunikationsmedien pro Nachricht deutlich. Das doch recht große Einsparpotential ist auf die Verwendung der eingeschränkten Pfadhüllen, wie sie auf Seite 144 eingeführt wurden, zurückzuführen: Nachrichten, die nur auf dem unteren Teil der Architektur versendet werden dürfen, erzeugen entsprechend weniger Pfadhüllen-abhängige Ungleichungen und reduzieren die Komplexität des Erfüllbarkeitsproblems.

Alle bisherigen Experimente verwendeten nur eine Art von Kommunikationsmedien, nämlich den Tokenring. Anspruch der in dieser Arbeit vorgestellten Methodik ist aber ausgewiesenermaßen die Behandlung heterogener hierarchischer Architekturen. Die Anwendbarkeit auch solcher Systeme soll durch das nun folgende Experiment demonstriert werden, indem in Architektur C der obere Tokenring-Bus durch einen CAN-Bus ersetzt wird. Das Kommunikationsmedium im ursprünglichen Anteil der Architektur (also die Verbindung von Knoten 0 bis 7) bleibt jedoch weiterhin ein Tokenring. Tabelle 5.7 zeigt das Ergebnis mit der eingeschränkten Platzierungsrestriktion, wie sie im vorigen Versuch vorgestellt wurde.

Experiment	Ergebnis	Laufzeit	Güte
Architektur C^* + [183]*	$\sum TRT_i = 8.55ms$	185min	100%

Tabelle 5.7: Ergebnis der Platzierung auf Architektur C, jedoch mit einem CAN-Bus für Knoten 8-11, mittels HySAT unter einer gegebenen Güte des Ergebnisses in Prozent.

Das Optimierungskriterium hierbei wurde dabei lediglich auf den Tokenring beschränkt und sollte die TRT des Mediums minimieren. Wenig überraschend entsteht auch hier das identische Ergebnis, wie in allen bisherigen Versuchen. Allerdings lohnt sich ein Blick auf die entstandene Platzierung, die in diesem Fall nicht, wie bisher, nur die Knoten 0 bis 7 nutzt. Dies ist in Tabelle 5.8 dargestellt.

Es ist ersichtlich, dass drei Tasks auf den Prozessorknoten 8 bis 11 allokiert wurden, obwohl sie Kommunikationen zu Tasks auf den Knoten 0 bis 7 versenden müssen. Dementsprechend ist die Antwortzeit der Nachrichten zwischen diesen Tasks und

Prozessorknoten	Allokierte Tasks
0	0 9 34 37
1	3 7 8 10 11 18 19 39
2	6 12 13 14 17 38
3	1 4 15 16 20 21 33
4	22 23 24 25
5	
6	26 27 28 29 36
7	30 31 32 40 41 42
8	
9	2
10	
11	5 35

Tabelle 5.8: Ergebnis der Platzierung auf Architektur C mit einem CAN-Bus für Knoten 8 bis 11.

ihren Partnertasks höher als in den ursprünglich erreichten Platzierungen, die beispielsweise in den Versuchen auf Architektur C erzeugt wurden. Es wird trotzdem ein bzgl. des Optimierungskriteriums optimales Ergebnis erzeugt, weil die Optimierung auf die Minimierung des TRT des unteren Bussystems aus ist und der CAN-Bus keinen Anteil an der Optimalität einer Platzierung hat[1]. Nachrichten, die über beide Medien versendet werden, erzeugen auf dem Tokenring die gleiche Last, die sie auch erzeugen würden, wenn der CAN-Bus nicht vorhanden wäre. Letztlich ist die Platzierung der drei angesprochenen Tasks bzgl. des Optimierungskriteriums sogar nahezu beliebig; sie hätten auch auf den Knoten 8,10,5 oder den Knoten des Ergebnisses aus Tabelle 5.6 platziert werden können, ohne jeglichen Einfluss auf die TRT des Tokenring-Busses. Vielmehr zeigt sich hier die Leistungsfähigkeit des entwickelten Verfahrens, auch mit Kommunikationen über mehrere Bussysteme hinweg umgehen zu können.

Eine Anwendung des zweiten Benchmark-Systemes aus Kapitel 5.6.1 führt zu sehr ähnlichen Ergebnissen, die hier der Vollständigkeit halber nur noch tabellarisch in Tabelle 5.9 dargestellt sind. Herauszuheben ist am ehesten das Ergebnis für eine Platzierung auf einem Netzwerk, dass aus Tokenring und CAN-Bus besteht (vgl. Architektur C* in Tabelle 5.9): Das Optimierungsverfahren ist wiederum bemüht, Kommunikationen möglichst auf den CAN-Bus auszulagern und erreicht daher eine Platzierung, in der keine einzige Nachricht mehr über den Tokenring-Bus verschickt werden muss.

Wie an der Interpretation des Ergebnisses aus den Versuchen in Tabelle 5.7 und 5.9 ersichtlich wird, ist die Wahl des Optimierungskriteriums sorgfältig abzuwägen. Würde beispielsweise auf eine Kombination aus Minimierung des TRT und der Aus-

[1]Die Optimierung zielt explizit nicht auf die Minimierung der Antwortzeiten von Nachrichten, da nur ein Bussystem betrachtet wurde. Dies ist anders als in den einfachen Architekturen, in denen die TRT auch immer identisch mit der Antwortzeit von Nachrichten ist.

Experiment	Ergebnis	Laufzeit	Güte
Architektur A + $\mathcal{T}_{synth}$	$\sum TRT_i = 2.6ms$	49 min	100%
Architektur B + $\mathcal{T}_{synth}$	$\sum TRT_i = 3.2ms$	113 min	100%
Architektur C + $\mathcal{T}_{synth}$	$\sum TRT_i = 1.5ms$	14 min	100%
Architektur C* + $\mathcal{T}_{synth}$	$\sum TRT_i = 0.0ms$	113 min	100%

Tabelle 5.9: Ergebnis der Platzierung des synthetischen Benchmarksystems (siehe Anhang E.1) mittels HySAT unter einer gegebenen Güte des Ergebnisses in Prozent. Architektur C* besteht wiederum aus einem Tokenring und einem CAN-Bus.

lastung des CAN-Busses hin optimiert, ließen sich derartige Ergebnisse vermeiden. Es lassen sich hier beliebige Funktionen finden, die eine Optimierung unter gewissen Randbedingungen des Systems vornehmen und die insbesondere auch mehrere Optimierungsziele in gewichteten Mischformen abdecken können. In der aktuellen Implementierung des Platzierungsverfahrens haben wir jedoch darauf verzichtet, derartige Funktionen anzubieten, zumal die zur Auswahl stehenden Möglichkeiten einer eindringlichen Untersuchung aus Entwicklungsprozess- und Systemintegrations-Sicht bedürfen. Stattdessen werden in der momentanen Ausbaustufe die folgenden Optimierungskriterien angeboten, aus denen der Benutzer eines auswählen kann:

- Minimierung von TRT in Tokenringbussen, bei mehreren Tokenring-Bussen auch die Summe der TRTs
- Minimierung der Auslastung auf CAN-Bussen, bei mehreren CAN-Bussen auch die Summe der Auslastung
- Minimierung der Länge einer TDMA-Round in Zeit-basierten Kommunikationsmedien sowie die Summe der TDMA-Längen bei mehrfacher Verwendung
- Gleichmäßige Lastverteilung auf allen Prozessorknoten, unabhängig von den gewählten Bus-Systemen

Diese Optionen stellen letztlich womöglich kein wirklich ausreichendes Instrumentarium für eine industrielle Anwendung des Verfahrens zur Verfügung, jedoch sind sie durchaus geeignet, die generelle Anwendbarkeit des Ansatzes zu zeigen. Erweiterungen auf weitergehende, auch kombinierte, Optimierungskriterien sind jedoch einfach in die Methodik integrierbar, und wir werden in Kapitel 6.4 zeigen, wie beispielsweise mehrfach parametrisierte Optimierungen anzuwenden sind.

5.6.3 Skalierbarkeit und Komplexität der Methodik

Nachdem in den letzten beiden Abschnitten die generelle Anwendbarkeit des Platzierungs- und Optimierungsverfahrens gezeigt werden konnte, stellt sich nun die Frage nach der Komplexität. Das zugrunde liegende Entscheidungsverfahren der Erfüllbarkeit ist im wesentlichen ein auch auf Heuristiken (z.B. welche Variablen werden zu welchem Zeitpunkt mit welchem Wert belegt, vgl. [42, 122]) beruhender Ansatz. Die Laufzeit hängt dabei wesentlich von der Anzahl der Booleschen Variablen in der jeweiligen Probleminstanz ab, wobei letzteres wiederum stark durch die Größe und

Struktur des Tasksystems sowie der Architekturgröße und Komplexität der Topologie bestimmt wird. Zur Bestimmung der Skalierbarkeit des Verfahrens werden wir daher zunächst einige Messungen vornehmen und anschließend eine Abschätzung der Anzahl der Variablen auf Grundlage der Formalisierung angeben.

Messungen zur Skalierbarkeit

Die Skalierbarkeit des Ansatzes kann generell in drei Dimensionen betrachtet werden:

- Größe der Architektur, d.h. wie hängt die Komplexität des Verfahrens von der Anzahl der Prozessorknoten einer Architektur ab
- Größe des Tasksystems, d.h. wie hängt die Komplexität des Verfahrens von der Anzahl der Tasks ab
- Topologie der Architektur, d.h. was bedeuten zusätzliche hierarchische Sub-Netze für die Komplexität

Einflüsse der Topologie sind bereits im letzten Kapitel (vgl. die Ergebnisse aus den Tabellen 5.6 und 5.9) ermittelt worden und zeigen, dass das Einfügen zusätzlicher Sub-Netze die Laufzeit des Verfahrens drastisch erhöht. Allgemeinere Aussagen sind hierbei nicht möglich, weil die Komplexität bzgl. der Topologie einer Architektur auch sehr stark vom verwendeten Tasksystem abhängt: Durch die Verwendung der eingeschränkten Pfadhüllen entstehen Komplexitätsverhältnisse, die nur in einer Kombination aus Topologie und Tasksystem bewertbar wären. Man kann jedoch generell feststellen, dass in einem inkrementellen Integrationsprozess innerhalb der typischen Anwendungsdomänen (beispielsweise im Bereich der Kfz-Elektronik) nur sehr wenige Nachrichten tatsächlich über das gesamte Bordnetz versendbar sind. Üblicherweise sind die überwiegende Mehrheit der Kommunikationen nur auf Sub-Netzen angesiedelt, so dass auch bei einer komplexen Vernetzung die Komplexität des Platzierungsproblems durch die Verwendung der eingeschränkten Pfadhüllen beherrschbar bleibt (siehe die beiden Ergebnisse für Architektur C in Tabelle 5.6 bzw. 5.9).

Zur Betrachtung der Abhängigkeit der Komplexität des Verfahrens von der Anzahl der Prozessorknoten dient die folgende Messung, deren Ergebnis in Abbildung 5.7 dargestellt ist. Ausgangspunkt ist ein festes Tasksystem mit 30 Tasks (siehe Anhang E.1), dass auf eine einfache Architektur — bestehend aus einer variablen Anzahl identischer Prozessorknoten und einem zentralen Backbone vom Typ Tokenring — platziert werden soll.

Für die Messung wurde die Anzahl der Prozessorknoten an dem gemeinsamen Kommunikationsmedium schrittweise um etwa 8 Knoten angehoben. Dargestellt sind in Abbildung 5.7 die entstehende Komplexität der jeweiligen Probleminstanz in den Größen Laufzeit, Anzahl der Booleschen Variablen und Anzahl der Booleschen Literale. Deutlich zu erkennen ist das nahezu lineare Ansteigen der Anzahl der Booleschen Variablen mit der Anzahl der Prozessorknoten. Dies ist in erster Linie auf die

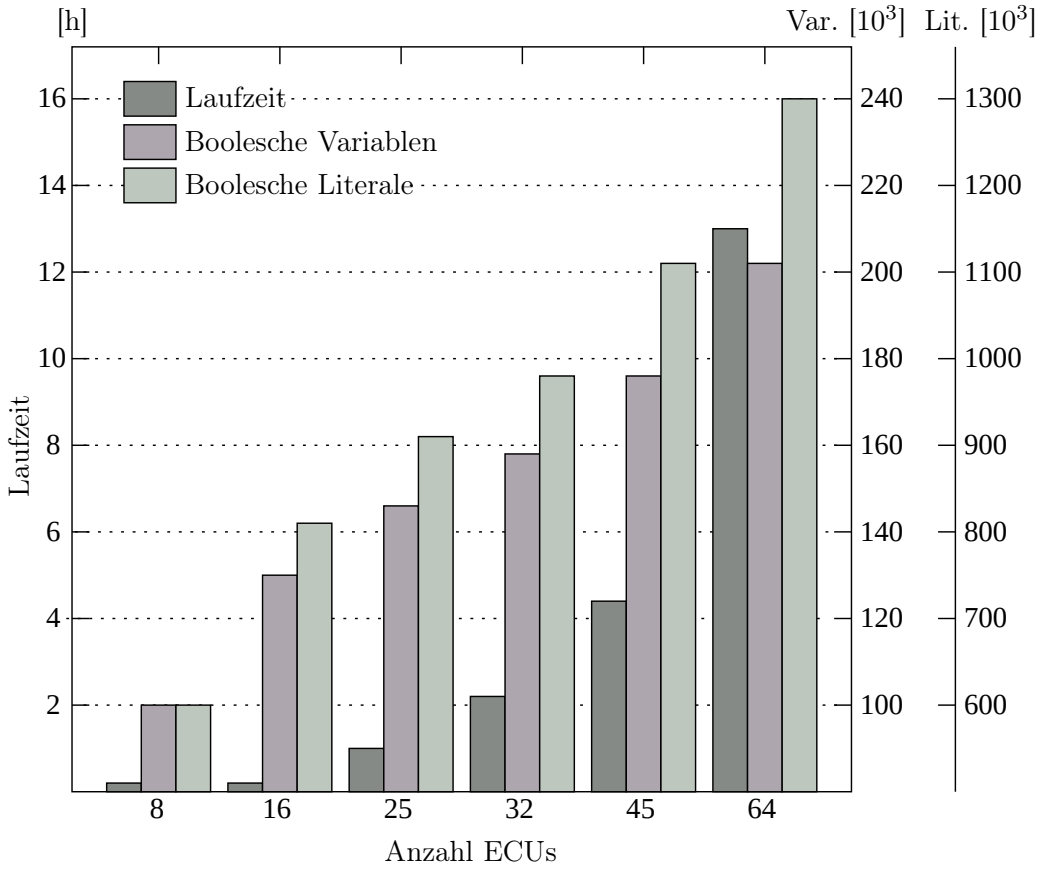


Abbildung 5.7: Skalierung der Komplexität des Verfahrens (Laufzeit, Anzahl Boolescher Variablen und Literale) in Abhängigkeit von der Anzahl der Prozessorknoten bei konstanter Taskanzahl.

logarithmische Kodierung der Platzierungsabbildung Π zurückzuführen und ermöglicht eine gute Skalierung bzgl. der Architekturgröße. Gerade im Hinblick auf eine inkrementelle Platzierung ist dies eine ganz wesentliche Eigenschaft des Ansatzes, da hier oftmals schon am Anfang recht große Architekturen betrachtet werden müssen. Die Laufzeiteigenschaften verhalten sich erwartungsgemäß mehr oder weniger exponentiell in der Anzahl der Booleschen Variablen. Das Experiment zeigt auch, dass ab Größen von 60 und mehr Prozessorknoten eine Grenze erreicht wird, ab der die Laufzeit einen intensiven Einsatz des Verfahrens stark beeinträchtigt. Andererseits weisen die typischen Task-Subnetzwerke in einem inkrementellen Platzierungsprozess üblicherweise einen relativ hohen Grad an Lokalität bzgl. eines Sub-Netzes der Architektur auf (beispielsweise wird die Mehrheit der Tasks, die der Motorsteuerung dienen, auch auf dem Sub-Netz für Powertrain angesiedelt sein). Dies führt zu einer Einschränkung der Platzierungsmöglichkeiten, was sich wiederum positiv in der Laufzeit (nicht jedoch bei der Anzahl der Variablen) niederschlägt[1].

Untersuchungen bezüglich der Abhängigkeit der Komplexität von der Anzahl der

[1]Dieses Verhalten ist übrigens in dem hier betrachteten Benchmark bewusst nicht ausgenutzt worden, sondern das Tasksystem aus Anhang E.1 ist bei jeder Erweiterung um zusätzliche Prozessorknoten so angepasst worden, dass es allen Tasks erlaubt war, auch alle neu hinzugekommenen Knoten zu benutzen.

Tasks sind in Abbildung 5.8 dargestellt. Hierzu wurde eine feste Architekturgröße gemäß [183] verwendet und auf diese ein Tasksystem mit variabler Anzahl an Tasks platziert. Das Tasksystem entspricht Fragmentierungen des aus Tabelle 5.2 auf Seite 172 dargestellten Systems. Dieses wurde schrittweise so vergrößert, dass Taskketten immer als Ganzes erhalten wurden. Wiederum wurden als Maß der Komplexität die Laufzeit, die Anzahl der Booleschen Variablen und Booleschen Literale der jeweiligen Probleminstanz verwendet.

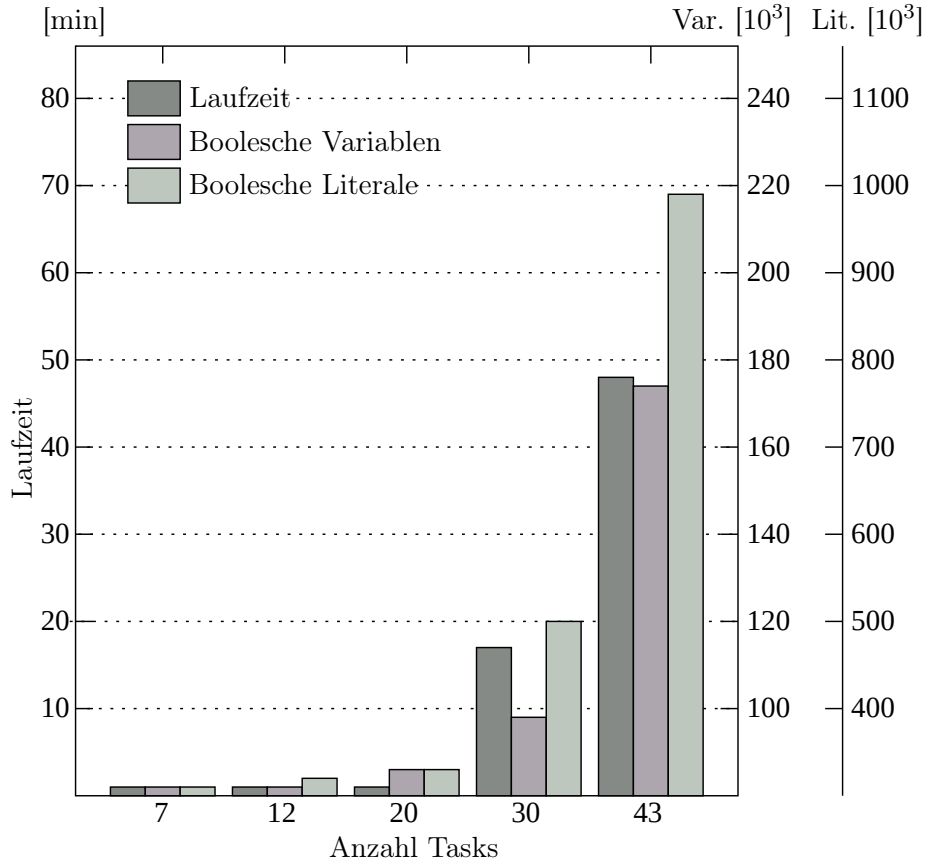


Abbildung 5.8: Skalierung der Komplexität des Verfahrens (Laufzeit, Anzahl Boolescher Variablen und Literale) in Abhängigkeit von der Anzahl der Tasks bei konstanter Architekturgröße.

Augenscheinlich besteht kein linearer Zusammenhang zwischen der Anzahl der Booleschen Variablen und der Anzahl der Tasks, sondern mindestens ein quadratischer. Dies liegt an der expliziten Kodierung der Nebenbedingungen wie auch der Platzierungsfunktion Π für Tasks (siehe auch die Ausführungen weiter unten). Insgesamt zeigt sich, dass für Systeme bis zu 50 Tasks auf einfachen Architekuren eine gute Performanz[1] erreicht wird. Diese Größenordnung ist vor dem Hintergrund einer inkrementellen Integration mehr als ausreichend, da — wie in Kapitel 6.2 dargestellt werden wird — im wesentlichen einzelnen Funktionen eines komplexen Systems integriert werden und diese Funktionen typischerweise mit deutlich weniger Tasks

[1] Man beachte hier auch nochmals, dass es sich bei diesem Verfahren um einen *optimalen* Ansatz handelt.

implementiert sind.

Nachdem die oben dargestellten Messungen an konkreten Probleminstanzen einen Einblick in die Komplexität des Verfahrens lieferten, soll die Komplexität im folgenden systematisch anhand der Ungleichungen diskutiert werden.

Quantitative Betrachtungen der Komplexität

Aussagen zur Laufzeit sind bei heuristischen Verfahren wie SAT-Checking generell nur sehr eingeschränkt zu treffen, da sie maßgeblich auch von der Struktur des Erfüllbarkeitsproblems abhängen und dies nicht unmittelbar aus den vorliegenden Gleichungssystemen abzulesen ist. Letztlich können hierzu nur Messungen angestellt werden, wie wir sie bereits oben vorgestellt haben. Neben diesen nicht weiter quantifizierbaren Effekten hängt die Laufzeit aber auch ganz wesentlich von der Anzahl der Booleschen Variablen und Constraints ab. Kann diese Anzahl reduziert werden, so ist i.a. auch davon auszugehen, dass die Laufzeit eines Erfüllbarkeitstests sinkt. Daher werden wir im folgenden aufzeigen, welche Anzahl Boolescher Variablen durch die verwendete Modellierung des Problems entstehen, wie dies kodiert wird und an welchen Stellen Reduzierungen dieser Zahl durchgeführt werden.
Zunächst bleibt festzustellen, dass die Ungleichungen von 5.5 bis 5.70 fast ausschließlich aus integer-arithmetischen Anteilen aufgebaut sind. In Kapitel 5.3.3 wurde bereits das Verfahren vorgestellt, mit dem eine Transformation integer-arithmetischer Ausdrücke in Boolesche Ungleichungen erfolgt. Wesentliches Merkmal ist hierbei die Verwendung einer Zweierkomplement-Darstellung für alle Integer-Variablen. Dies bedingt natürlich eine Endlichkeit der verwendeten Variablen, also die Kentniss des Ranges, den eine Variable haben kann. Hiermit kann nun für jede Variable die Breite der Booleschen Darstellung berechnet werden. Sei die Anzahl der Bits einer Variable in Zweierkomplementdarstellung definiert durch

$$\mathcal{Z}(x) = \lceil \log_2(x) \rceil + 1$$

Zur Reduzierung der Anzahl der Booleschen Variablen sind die Ranges der einzelnen Variablen möglichst klein zu wählen. Diese Ranges sind grundsätzlich bzgl. der Probleminstanz berechenbar, und wir werden im folgenden für alle verwendeten Variablen zeigen, auf welche Art und Weise der kleinste Wertebereich aus der Beschreibung des Platzierungsproblems determiniert wird. Gleichzeitig wird die Anzahl dieser Variablen — wiederum abhängig von der betrachteten Probleminstanz — ebenfalls quantifiziert, so dass auf diese Weise letztlich errechnet werden kann, wieviele Boolesche Variablen das Problem in Abhängigkeit von der Problemgröße erzeugt. Hierzu benötigen wir zunächst noch Informationen zu den möglichen auftretenden Nachrichten eines Systems: Sei also die Menge aller Nachrichten definiert durch

$$\mathcal{G} = \bigcup_{\forall \tau_i \in \mathcal{T}} \gamma_i$$

Zusammen mit dem sonstigen Wissen aus der Problembeschreibung können nun die Anzahlen und Bitbreiten der einzelnen Variablen berechnet werden. In Tabelle 5.10

sind zunächst alle Variablen aufgelistet, die für die Platzierung und Antwortzeitberechnung der Tasks auf die Architektur verantwortlich sind.

Variable	Anzahl	Bitbreite
$\texttt{place}_i$	$\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\lvert P\rvert\right)$
$\texttt{wcet}_i$	$\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\max_{p\in P}\left(c_i\right)\right)$
$\texttt{d}_i$	$\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\Delta_\tau(\tau_i)+\min_{g_l\in\gamma_i}(\Delta_\gamma(g_l))\right)$
$\texttt{r}_i$	$\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\Delta_\tau(\tau_i)+\min_{g_l\in\gamma_i}(\Delta_\gamma(g_l))\right)$
$\texttt{p}_i^j$	$\lvert\mathcal{T}\rvert\cdot\lvert\mathcal{T}\rvert$	1
$\texttt{I}_i^j$	$\lvert\mathcal{T}\rvert\cdot\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\left\lvert\frac{\Delta_\tau(\tau_i)+\min_{g_l\in\gamma_i}(\Delta_\gamma(g_l))+J_j}{p_j}\right\rvert\right)$
$\texttt{interf}_i^j$	$\lvert\mathcal{T}\rvert\cdot\lvert\mathcal{T}\rvert$	$\mathcal{Z}\left(\left\lvert\frac{\Delta_\tau(\tau_i)+\min_{g_l\in\gamma_i}(\Delta_\gamma(g_l))+J_j}{p_j}\right\rvert\right)$
$\texttt{m}_p^i$	$\lvert\mathcal{T}\rvert\cdot\lvert P\rvert$	$\mathcal{Z}\left(\max_{p\in P}(\mu_i^\tau(p))\right)$

Tabelle 5.10: Anzahl und Bitbreite der Variablen, die für die Platzierung und Antwortzeitberechnung der Task zuständig sind, in Abhängigkeit von der Probleminstanz.

Die angegebene Bitbreite entspricht gleichzeitig der Anzahl der Booleschen Variablen, die die jeweilige Integer-Variable erzeugt. Sie ist i.a. sehr einfach aus der Problemstruktur herleitbar (siehe bespielsweise die Ranges für die $\texttt{place}_i$, die von der Anzahl der Prozessorknoten der Architektur bestimmt wird oder die Ranges der Antwortzeit-Variablen $\texttt{r}_i$, die durch die maximal mögliche Deadline der betrachteten Task τ_i begrenzt werden kann, in Tabelle 5.10). An manchen Stellen sind etwas umfangreichere Berechnungen möglich, um die Bitbreite zu reduzieren. So ist beispielsweise die Anzahl der maximal möglichen Preemptionen $\texttt{I}_i^j$ einer Task τ_i durch eine höher priorisierte Task τ_j nach oben beschränkt durch die Gauß-Funktion über dem Quotienten aus maximal möglicher Antwortzeit von Tasks τ_i (was der maximal möglichen Deadline entspricht) plus Jitter der Task und der Periode des unterbrechenden Tasks.
Die Anzahl der Variablen ist direkt von der Problemgröße abhängig. Die in Gleichung 5.5 und folgende dargestellte Kodierung des Problems bewirkt z.B., dass die Anzahl der Platzierungsvariablen (aber auch der Deadline- und Antwortzeit-Variablen) linear mit der Anzahl der Tasks eines Systems steigt. Zusammen mit der Zweierkomplementdarstellung dieser Werte ergibt sich letztlich das Verhalten, welches in Abbildung 5.7 gemessen wurde. Variablen, die Taskrelationen, wie etwa Priorität oder Preemptionen, darstellen sind hingegen durch ein quadratisches Wachstum gekennzeichet.

Tabelle 5.11 untersucht Anzahl und Bitbreiten derjenigen Variablen, die mit der Nachrichtenallokation zu tun haben, also welche Kommunikationsmedien werden

benutzt, welche Pfadhülle sind dafür notwendig sowie der allgemeine Teil der Antwortzeitberechnung von Nachrichten.

Variable	Anzahl	Bitbreite
$\mathtt{K}_{g_t}^k$	$\lvert K\rvert \cdot \lvert\mathcal{G}\rvert$	1
$\mathtt{Pf}_{g_t}$	$\lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\lvert\text{Pfadhüllen}\rvert\right)$
$\mathtt{locD}_{g_t}^k$	$\lvert K\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{serv}_{g_t}$	$\lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\lvert K\rvert \cdot (\Delta_{IN} + \Delta_{OUT})\right)$
$\mathtt{Corr}_{g_t}$	$\lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_{IN} + \Delta_{OUT}\right)$
$\mathtt{J}_{g_t}^k$	$\lvert K\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{rm}_{g_t}^k$	$\lvert K\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{lat}_{g_t}^k$	$\lvert K\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{pm}_{g_i}^{g_j}$	$\lvert\mathcal{G}\rvert \cdot \lvert\mathcal{G}\rvert$	1

Tabelle 5.11: Anzahl und Bitbreite der Variablen, die für die Platzierung und Antwortzeitberechnung der Nachrichten zuständig sind, in Abhängigkeit von der Probleminstanz.

Die Anzahl der jeweiligen Variable hängt direkt von der Anzahl der Nachrichten des Systems und der Anzahl der Kommunikationsmedien der Architektur ab. Die Bitbreiten sind wiederum ähnlich wie oben einfach zu bestimmen. Lediglich die $\mathtt{Pf}_{g_t}$-Variablen, die zur Selektion der verwendeten Pfadhülle notwendig sind, sind bzgl. ihrer Bitbreite nicht allgemein anzugeben. Tatsächlich hängt die Breite direkt von der Menge der möglichen Pfadhüllen ab, die jedoch sehr architektur-spezifisch ist. Somit kann aufgrund der beliebigen Topologie der betrachteten Systeme keine allgemeine Berechnungsvorschrift angegeben werden. Vielmehr ist hier eine problemspezifische Analyse der Architektur erforderlich, die diese Werte erstellt (siehe auch das Beispiel 5.4 auf Seite 140).

Abhängig von den jeweils vorhandenen Typen an Kommunikationsmedien (Tokenring, CAN, TTP oder FlexRay) sind weitere Variablen einzuführen, die für die Antwortzeitberechnung von Nachrichten zuständig sind. Diese sind — getrennt nach dem jeweiligen Typ des Mediums — in den Tabellen 5.12 bis 5.15 dargestellt.

Variable	Anzahl	Bitbreite
$\mathtt{TRT}_k$	$\lvert K_{tok}\rvert$	$\mathcal{Z}\left(\sum_{g_t \in \mathcal{G}} \frac{s_t \cdot 8}{\upsilon_k} + \lvert k\rvert \cdot \xi\right)$

Tabelle 5.12: Anzahl und Bitbreite der Variablen, die für die Antwortzeitberechnung der Nachrichten auf einem Tokenring-Bus zuständig sind, in Abhängigkeit von der Probleminstanz.

Auffällig ist insbesondere die Einführung neuer Konstanten (N_{slot}^k und S^k), die nicht durch die Systemspezifikation vorgegeben sind. Sie sind jedoch erforderlich, um das Problem einerseits einer statischen Erzeugung von Ungleichungssystemen zugänglich

Variable	Anzahl	Bitbreite
$\mathtt{Im}^k_{g_i \| g_j}$	$\lvert\mathcal{G}\rvert \cdot \lvert\mathcal{G}\rvert \cdot \lvert K_{CAN}\rvert$	$\mathcal{Z}\left(\left\lceil \frac{\Delta_\gamma(g_t) + J_{g_j}}{p_j} \right\rceil\right)$
$\mathtt{interfm}^k_{g_i \| g_j}$	$\lvert\mathcal{G}\rvert \cdot \lvert\mathcal{G}\rvert \cdot \lvert K_{CAN}\rvert$	$\mathcal{Z}\left(\left\lceil \frac{\Delta_\gamma(g_t) + J_{g_j}}{p_j} \right\rceil\right)$

Tabelle 5.13: Anzahl und Bitbreite der Variablen, die für die Antwortzeitberechnung der Nachrichten auf einem CAN-Bus zuständig sind, in Abhängigkeit von der Probleminstanz.

Variable	Anzahl	Bitbreite
$\mathtt{block}^k_{g_t}$	$\lvert\mathcal{G}\rvert \cdot \lvert K_{TTP}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{slot}^k_p$	$\lvert K_{TTP}\rvert \cdot \lvert P\rvert$	$\mathcal{Z}\left(N^k_{slot}\right)$
$\mathtt{tdmalen}_k$	$\lvert K_{TTP}\rvert$	$\mathcal{Z}\left(\sum_{p\in k} N^k_{slot} \cdot \rho_k\right)$
$\mathtt{ownslot}^k_{g_t}$	$\lvert K_{TTP}\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(N^k_{slot}\right)$
$\mathtt{in}^k_{g_t}$	$\lvert K_{TTP}\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\lvert P\rvert\right)$
$\mathtt{Imb}^k_{g_t}$	$\lvert K_{TTP}\rvert \cdot \lvert\mathcal{G}\rvert$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$

Tabelle 5.14: Anzahl und Bitbreite der Variablen, die für die Antwortzeitberechnung der Nachrichten auf einem TTP-Bus zuständig sind, in Abhängigkeit von der Probleminstanz.

Variable	Anzahl	Bitbreite
$\mathtt{s}^k_i$	$\lvert K_{MSA}\rvert \cdot S^k$	$\mathcal{Z}\left(\lvert P\rvert + 1\right)$
$\mathtt{slen}^k_i$	$\lvert K_{MSA}\rvert \cdot S^k$	$\mathcal{Z}\left(N^k_{slot} \cdot \omega_k\right)$
$\mathtt{slotlength}^k$	$\lvert K_{MSA}\rvert$	$\mathcal{Z}\left(N^k_{slot}\right)$
$\mathtt{rm}^{k,z}_{g_t}$	$\lvert K_{MSA}\rvert \cdot \lvert\mathcal{G}\rvert \cdot S^k$	$\mathcal{Z}\left(\Delta_\gamma(g_t)\right)$
$\mathtt{interfm}^{k,z}_{g_i \| g_j}$	$\lvert K_{MSA}\rvert \cdot \lvert\mathcal{G}\rvert \cdot \lvert\mathcal{G}\rvert \cdot S^k$	$\mathcal{Z}\left(\left\lceil \frac{\Delta_\gamma(g_i) + J_{g_j}}{p_j} \right\rceil\right)$
$\mathtt{interfb}^{k,z}_{g_i \| g_j}$	$\lvert K_{MSA}\rvert \cdot \lvert\mathcal{G}\rvert \cdot \lvert\mathcal{G}\rvert \cdot S^k$	$\mathcal{Z}\left(\Delta_\gamma(g_i)\right)$
$\mathtt{w}^{k,z}_{g_t,i}$	$\lvert k_{MSA}\rvert \cdot \lvert\mathcal{G}\rvert \cdot S^k \cdot S^k$	$\mathcal{Z}\left(\Delta_\gamma(g_i)\right)$

Tabelle 5.15: Anzahl und Bitbreite der Variablen, die für die Antwortzeitberechnung der Nachrichten auf einem Bus mit komplexer Slotverteilung zuständig sind, in Abhängigkeit von der Probleminstanz.

zu machen (im Falle von S^k, siehe Diskussion auf Seite 154) und andererseits um für bestimmte Variablen überhaupt eine Beschränkung des Wertebereiches angeben zu können (N^k_{slot}). Zwar stellen diese Konstanten eine Einschränkung des Entwurfsraumes aus Gründen des Optimierungsverfahrens dar, allerdings existieren üblicherweise Engineering-Guidelines im Entwurf derartiger Systeme, die solche Einschränkungen ohnehin vorschreiben.

Wie bereits in Kapitel 5.3.3 dargelegt, ist die Zweierkomplement-Darstellung der Integer-Variablen nicht die einzige Quelle für Boolesche Variablen. Die Transformation der arithmetischen Ausdrücke in Boolesche Gleichungen erzeugt zusätzliche

Hilfsvariablen durch die Verwendung von Grundbausteinen auf der einen Seite (z.B. zusätzlich notwendige Carry-Variablen im Falle eines Addierers) und die notwendige Triplet-Form der Darstellung auf der anderen Seite. Im folgenden soll daher dieser Overhead ebenfalls quantifiziert werden. Dazu wird jede Ungleichung der Modellierung auf ihre problemabhängige Anzahl und die Verwendung bestimmter Grundoperationensklassen, die eine quantifizierbare Anzahl an Hilfsvariablen erzeugen, hin untersucht. Tabelle 5.16 (entnommen aus [111]) zeigt, welche Grundoperationsklassen es gibt und gibt den jeweiligen Bedarf an zusätzlich eingeführten Booleschen Hilfsvariablen an.

Operation	zusätzliche Variablen
Logische	—
Vergleiche	n
Addition/Subtraktion	$2n$
Multiplikation/Division	$2 \cdot n \cdot m$

Tabelle 5.16: Anzahl der zusätzlichen Booleschen Hilfsvariablen laut [111]

Die Klassifizierung der Operationen in Tabelle 5.16 erfolgte anhand gleicher Kosten bzgl. Hilfsvariablen. Andere als dort dargelegte Operationen sind in der Modellierung des Platzierungsproblems nicht verwendet worden. Die Anzahl der Hilfsvariablen ist jeweils gemessen an der maximalen Bitbreite der Operatoren n und beinhaltet damit eine etwaige Bereichserweiterung im Falle ungleicher Ranges zweier Operanden. Bei der Multiplikation/Division sind die Ranges beider Operanden zu beachten.

Mithilfe dieser Klassifizierung können nun die einzelnen Gleichungen der Modellierung betrachtet werden, und anhand der Variablenzahlen und Bitbreiten kann anschließend errechnet werden, welche Variablenanzahl für eine Probleminstanz zu erwarten ist. Wir werden dies hier exemplarisch anhand des Beispiels aus [183] durchführen und daher nur die dafür notwendigen Ungleichungen betrachten. Berechnungen für die Anzahl von Hilfsvariablen für alle übrigen Ungleichungen sind vollkommen analog durchzuführen.
Zunächst sind in Tabelle 5.17 die Ungleichungen betrachtet worden, die für die Platzierungsentscheidung von Tasks zuständig sind.

Basierend auf den Werten aus Tabelle 5.16 kann nun die Anzahl der Booleschen Variablen einer Ungleichung ausgerechnet werden: Sie ergibt sich aus der Anzahl der Bits der Variablendarstellung aus Tabelle 5.10 bis 5.15 und dem Anteil der Hilfsvariablen, der aus Tabelle 5.17 entsteht. Letzterer wird errechnet durch eine Multiplikation der Anzahl der Gleichungen mit der Anzahl der Hilfsvariablen pro Gleichungsinstanz. Pro Gleichungsinstanz muss die Summe aus verwendeten Operationen multipliziert mit deren Hilfsvariablenanzahl laut Tabelle 5.16 (die wiederum von der Bitbreite der Operanden abhängig ist) berücksichtigt werden. Beispielsweise ergibt sich für Gleichung 5.13 die folgende Anzahl an Variablen, wenn das System aus 43 Tasks besteht und die Bitbreiten der Additionsoperanden mit 12 Bit, sowie

Gleichung	Anzahl	Operationen			
		Logische	Vergleiche	Add.	Mul.
(5.5)	$\sum_{\tau_i \in \mathcal{T}} (\lvert P \rvert - \lvert \pi_i \rvert)$		1		
(5.6)	$\sum_{\tau_i \in \mathcal{T}} \lvert \delta_i \rvert$		1		
(5.7)	$\lvert \mathcal{T} \rvert \cdot \lvert P \rvert$	1	2		
(5.8)	$\lvert \mathcal{T} \rvert$	$1 + \lvert \gamma_i \rvert$	$1 + \lvert \gamma_i \rvert$		
(5.9)	$\lvert \mathcal{T} \rvert$	$2 + \lvert \gamma_i \rvert$	$1 + \lvert \gamma_i \rvert$		
(5.10)	$\lvert \mathcal{T} \rvert$		1	$1 + \lvert T \rvert$	
(5.11)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	2	2		1
(5.12)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	3	2		
(5.13)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	2	3	1	
(5.14)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	1	2		
(5.15)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	1	2		
(5.17)	$\lvert T \rvert$		1		
(5.18)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	2	3	3	2
(5.19)	$\lvert T \rvert \cdot (\lvert T \rvert - 1)$	1	2		
(5.20)	$\lvert \mathcal{T} \rvert \cdot \lvert P \rvert$	1	2		
(5.21)	$\lvert \mathcal{T} \rvert \cdot \lvert P \rvert$	1	2		
(5.22)	$\lvert P \rvert$		1	$1 + \lvert \mathcal{T} \rvert$	

Tabelle 5.17: Anzahl der Constraints und deren Operatoren für den Bereich der Taskplatzierung

die Operanden der Vergleichsoperation mit 4 Bit angegeben sind:

$$\begin{aligned} HV &= \lvert \mathcal{T} \rvert \cdot (\lvert \mathcal{T} \rvert - 1) \cdot (3 \cdot n_{=} + 2 \cdot n_{+}) \\ &= 43 \cdot 42 \cdot (3 \cdot 4 + 2 \cdot 12) \\ &= 65016 \end{aligned}$$

Dieses Berechnungsschema wird nun auf alle für die Probleminstanz notwendigen Gleichungen angewandt. Das hier betrachtete Beispiel aus [183] verwendet als einziges Kommunikationsmedium einen Tokenring, so dass wir uns an dieser Stelle auf die dafür notwendigen Gleichungen beschränken werden. Die dafür interessierenden Ungleichungssysteme für Pfadhüllen, Antwortzeitberechnung der Nachrichten und Latenzzeiten des Tokenring sind in Tabelle 5.18 und 5.19 dokumentiert.

Eine Anwendung auf das in [183] beschriebene und in Kapitel 5.6.1 evaluierte Platzierungsproblem ergibt eine Mächtigkeit der Taskmenge von $\lvert \mathcal{T} \rvert = 43$ und der Nachrichtenmenge von $\lvert \mathcal{G} \rvert = 36$. Weiterhin existieren aufgrund der einfachen Topologie nur zwei Pfadhüllen (eine mit dem Tokenring und die leere Pfadhülle). Zur Verhinderung von zu großen Überapproximationen von Rundungen (z.B. ist bei der Übertragungszeit von Nachrichten das Ergebnis nicht ganzzahlig) sind Antwortzeiten, Perioden und Deadlines von Tasks mit dem Faktor 10 und gleiches für Nachrichten mit dem Faktor 100000 skaliert worden. Die daraus entstehenden Bitbreiten der Va-

Gleichung	Anzahl	Operationen			
		Logische	Vergleiche	Add.	Mul.
(5.23)	$\|\mathcal{G}\|$	$1 + \|K\|$	1		
(5.24)	$\|\mathcal{G}\|$	$1 + \|K\|$	1		
(5.25)	$\|PH\| \cdot \|\mathcal{G}\|$	$\sum_{p \in ph_l} (2 + 3 \cdot \|p\|)$	$\sum_{p \in ph_l} 2 \cdot \|p\|$		
(5.27)	$\|\mathcal{G}\| \cdot \|K\|$	2	1		
(5.28)	$\|\mathcal{G}\| \cdot \|K\|$	1	1		
(5.29)	$\|\mathcal{G}\|$		1	$1 + \|K\|$	
(5.30)	$\|\mathcal{G}\|$	1	$1 + \|K\|$	$1 + 2 \cdot \|K\|$	$\|K\|$
(5.31)	$\|\mathcal{G}\|$	1	$1 + \|K\|$	$\|K\|$	
(5.32)	$\|\mathcal{G}\|$	1	$1 + \|K\|$		
(5.33)	$\|PH\| \cdot \|\mathcal{G}\|$	$1 + \|\max(ph_i)\|$	$1 + \|\max(ph_i)\|$	$\|\max(ph_i)\|$	

Tabelle 5.18: Anzahl der Constraints und deren Operatoren für den Bereich der Pfadsuche

Gleichung	Anzahl	Operationen			
		Logische	Vergleiche	Add.	Mul.
(5.41)	$\|K_{tok}\| \cdot \|\mathcal{G}\|$		1		
(5.42)	$\|K_{tok}\|$		1	$\|\mathcal{G}\|$	$\|\mathcal{G}\|$

Tabelle 5.19: Anzahl der Constraints und deren Operatoren für den Bereich der Antwortzeitberechnung von Nachrichten auf einem Tokenring

riablen und Hilfsvariablen sind entsprechend oben skizzierter Berechnungsvorschrift in der Berechnung der Anzahl der Booleschen Variablen berücksichtigt worden und führen zu dem Ergebnis in Tabelle 5.20. Dort ist gleichzeitig der Vergleich mit der tatsächlich gemessenen Anzahl der Booleschen Variablen der Probleminstanz in einem HySAT-Lauf eingetragen.

Benchmark	theoretische Anzahl	praktische Anzahl
[183]	$3389 \cdot 10^3$	$174 \cdot 10^3$

Tabelle 5.20: Vergleich der theoretischen und praktischen Anzahl Boolescher Variablen für das Platzierungsproblem aus [183].

Es entsteht also theoretisch eine sehr große Menge an Booleschen Variablen für diese Probleminstanz, die aber glücklicherweise im praktischen Einsatz sehr stark reduziert werden kann; im angegebenen Beispiel in Tabelle 5.20 etwa auf ein Zwanzigstel der theoretischen Anzahl. Die Gründe für eine derart drastische Reduzierung der Komplexität liegen in zahlreichen Optimierungen, die während der Erzeugung der Gleichung für den SAT-Checker zum Einsatz kommen. Im folgenden soll anhand von kleinen Beispielen exemplarisch dargestellt werden, welche Arten der Optimierung verwendet werden und welche Auswirkungen diese haben. Auf eine detaillierte Beschreibung aller Optimierungen soll an dieser Stelle verzichtet werden.
Man kann die unterschiedlichen Optimierungen grob in 3 Gruppen klassifizieren:

1. Transformation 2-wertiger Variablen in Boolesche Variablen

2. Transformation der Gleichungen in äquivalente, aber Variablen-sparende Kodierungen

3. Reduktion der Variablenanzahl durch Ausnutzen von aus der Problemstruktur abgeleitetem Wissen

Schritt 1. ist ein sehr einfaches Verfahren, dass es ermöglicht, 2-wertige Variablen, die in der Zweierkomplementdarstellung immer 2 Bit benötigen, in ihrer Bitbreite zu reduzieren. Trotz der Einfachheit und auch relativ bescheidenen Einsparung der Bitanzahl (nur wenige Variablen sind 2-wertiger Natur) erlaubt diese Optimierung weitreichende Einsparung in Kombination mit Schritt 2. (s.u.). Anwendung findet dies beispielsweise bei den Variablen, die Prioritätsbeziehungen darstellen und bei den Variablen, die festlegen, ob ein Kommunikationsmedium auf einem Pfad genutzt wird oder nicht. Gleichzeitig können auch komplexere arithmetische Ausdrücke in logische Ausdrücke umgewandelt werden, die gemäß Tabelle 5.16 keine zusätzlichen Hilfsvariablen erzeugen (beispielsweise kann die Addition in Gleichung 5.13 auf diese Weise in einen XOR-Ausdruck umgewandelt werden).

Schritt 2. versucht Gleichungen mit teuren arithmetischen Operationen in äquivalente mit weniger teuren Operationen zu transformieren. Ein Beispiel hierfür ist die konkrete Kodierung für Gleichung 5.42, in der eine ganze Reihe von Multiplikationen vorkommen. Diese dienen jedoch nur dem Zweck, diejenigen Nachrichten aus der Buslast herauszurechnen, die diesen Bus nicht benutzen. Dies wird durch die Multiplikation mit einer binären Variable erreicht. Durch die in Schritt 1. angewendete Reduktion dieser binären Variable auf eine Boolesche Variable lässt sich hier nun eine günstigere Variante kodieren, indem das von HySAT als logischer Operator angebotenen `if-then-else`-Konstrukt (vgl. [111]) verwendet wird. Dies erspart jegliche Multiplikation und mithin auch die durch Multiplikation eingeführte große Anzahl an Hilfsvariablen.

Schritt 3. schließlich ermöglicht die größte Reduktion, indem Variablen und Gleichungen komplett gelöscht werden, die für die aktuelle Probleminstanz nicht relevant sind. So können beispielsweise Interferenzbetrachtungen von Tasks, die niemals auf einem Knoten gemeinsam allokiert werden können, komplett entfallen. Gleiches gilt in eingeschränkte Maße für Prioritäten: Da Tasks und Nachrichten nach dem Deadline Monotonen Prinzip priorisiert werden, kann für einen Großteil der Tasks- und Nachrichten-Paare bereits vorab eine Preemptionbetrachtung ausgeschlossen werden. Für Nachrichten ist dies einfach zu analysieren, für Tasks hingegen etwas aufwändiger, da ihre Deadlines sich aufgrund von lokalem Nachrichtenaustausch dynamisch ändern können. Optimiert wird speziell für Tasks hier nach einer vorausschauenden Analyse, die die möglichen Deadlinekombinationen untersucht und auch daraus in vielen Fällen Preemptionsfreiheit als Systeminvariante identifizieren kann. Nicht zuletzt ist auch die in Kapitel 5.3.5 beschriebene Nutzung eingeschränkter Pfadhüllen eines dieser Optimierungen der Gleichungserzeugung, die die Anzahl der Variablen drastisch reduzieren kann.

Gerade Schritt 3. zeigt, dass das Einsparpotential sehr eng an die Struktur des Problems gekoppelt ist. Das Beispiel in Tabelle 5.20 macht aber auch deutlich, dass in realistischen Systemen tatsächlich große Mengen an Variablen entfallen können. Gleichzeitig werden gerade die Optimierungen aus Schritt 3. auch die Zahl der Booleschen Constraints einer Probleminstanz deutlich reduziert. Auch dieses hat einen starken Einfluss auf die Laufzeit des Verfahrens. Unterstützt werden diese Optimierungen abschließend noch durch Optimierungen innerhalb des Solvers beim Übersetzen der arithmetischen Ausdrücke in Boolesche Formeln unter Verwendung vordefinierter Komponenten, wie etwa Voladdierer etc. Z.B. führen dort Operationen mit Konstanten zu einer deutlich kompakteren Kodierung (für weitergehende Erläuterungen sei hier wiederum auf [111] verwiesen). Insbesondere bei den in der Modellierung des Platzierungsproblems oftmals auftretenden Multiplikationen mit Konstanten dürfte dies einen nicht unerheblichen Effekt auf die letztendliche Kodierung und Laufzeit des Verfahrens haben.

5.6.4 Messungen für unterschiedliche Bussysteme

Nachdem in den vorangegangenen Abschnitten die Anwendbarkeit des SAT-basierten Optimierungsverfahrens vorgestellt wurde, soll dieses Verfahren nun genutzt werden, um das unterschiedliche Verhalten der Nachrichtenübermittlung über die unterschiedlichen Typen von Kommunikationsmedien zu betrachten. Dazu wird eine feste Konfiguration gegeben, und die Nachrichtenübermittlung der in dieser Konfiguration auftretenden Nachrichten werden auf unterschiedlichen Bussystemen bzgl. einer Minimierung der möglichen Deadline der Nachrichten optimiert.
Zwecks Einschätzung der Effekte auf anderen Bussystemen soll hier das Beispiel aus [183] betrachtet werden. Dabei wird als Ausgangs-Konfiguration für die Tasks diejenige gewählt, die mittels der Optimierung bzgl. minimaler TRT auf einem Tokenring gefunden wurde. Dies erlaubt es, das unterschiedliche Verhalten beispielsweise von Tokenring und TTP oder dem statischen Part von FlexRay zu betrachten und zu vergleichen. Da das betrachtete System sehr eng bzgl. der Kommunikationsbandbreite ist und die verwendeten Protokolle, anders als das einfache Tokenring-Verfahren, über komplexe Header für Frames verfügen, wurde in den folgenden Messungen von einer Übertragungsgeschwindigkeit von 250 Bytes pro Millisekunde ausgegangen.

Messungen für Architekturen mit Tokenring oder CAN-Bus als zentrales Bussystem wurden bereits in Kapitel 5.6.1 angegeben, so dass hier an dieser Stelle lediglich darauf verwiesen werden soll.

Führt man eine Optimierung bzgl. der minimal möglichen Deadline der Nachrichten für ein auf dem TTP-Prinzip basierendes Bussystem für das Beispiel von oben durch, so ergibt sich eine minimale Deadline von $\Delta_\gamma^{\min} = 4.012ms$. Als Parameter für die Nachrichtenübermittlung wurden bei dieser Optimierung Frames verwendet, die über einen 42 Bit langen Header sowie über eine Nutzdatenlast von 16 Bytes verfügen, wobei jedoch jeder Frame 16 Bytes Nutzdaten enthält, unabhängig davon, ob die Nachricht weniger Bytes benötigt. Der Abstand der ermittelten Antwortzeit der Nachrichten von der minimal erreichbaren Deadline ist in Tabelle 5.21 darge-

stellt und gibt eine Vorstellung von dem verbleibenden Freiraum.

Nachricht	Δt $[ms]$
$\tau_0 \rightarrow \tau_1$	1.02
$\tau_1 \rightarrow \tau_3$	0.41
$\tau_1 \rightarrow \tau_5$	0.27
$\tau_2 \rightarrow \tau_3$	0.89
$\tau_4 \rightarrow \tau_5$	0.0
$\tau_5 \rightarrow \tau_6$	0.0
$\tau_9 \rightarrow \tau_{11}$	0.48
$\tau_{13} \rightarrow \tau_{15}$	0.41
$\tau_{14} \rightarrow \tau_{15}$	0.21
$\tau_{16} \rightarrow \tau_{17}$	0.56
$\tau_{25} \rightarrow \tau_{26}$	0.0
$\tau_{33} \rightarrow \tau_{35}$	0.0
$\tau_{34} \rightarrow \tau_{35}$	0.27
$\tau_{35} \rightarrow \tau_{36}$	0.0
$\tau_{38} \rightarrow \tau_{40}$	0.55
$\tau_{39} \rightarrow \tau_{40}$	0.0

Tabelle 5.21: Differenz der Antwortzeiten der Nachricht auf einem TTP-basierten Bussystem zur minimalen Deadline bei einer Übertragungsrate von 250 Bytes/sec.

Bei der Interpretation des verbleibenden Freiraums ist allerdings zu berücksichtigen, dass aufgrund der Gleichungen für die Berechnung der Antwortzeit die Verlängerung eines Slots der TDMA-Round (z.B. für eine weitere hinzugekommende Nachricht) sich auf die Antwortzeiten aller Nachrichten niederschlägt. Insofern ist zwar in manchen Antwortzeiten noch eine ungenutzte Zeitspanne enthalten, diese kann aber in der Regel kaum genutzt werden.
Bei der Optimierung erweist sich auch, dass es nicht genügt, die Länge der TDMA-Round zu minimieren: Aufgrund der Ereignis-basierten Natur der Nachrichten kann die Verschickung der Nachricht sich durchaus über mehrere TDMA-Runden erstrecken und es kann ggf. günstiger sein, mehrere Frames einer Nachricht in einem Slot zu verschicken, anstelle die Slots mit einer minimalen Länge von einem Frame auszustatten. Für die oben dargestellte Probleminstanz ergeben sich beispielsweise vollkommen unterschiedliche Slotgrößen von 11 Frames bis 2 Frames pro Slot im optimalen Fall.
Im Vergleich hierzu liefert die Optimierung mit Tokenring anstelle des TTP-basierten Verfahrens eine minimale Deadline von $3.28ms$, wobei allerdings zu berücksichtigen ist, dass der Tokenring keine Header besitzt. Wird er ebenfalls mit 42-Bit Headern, allerdings pro Nachricht, ausgestattet, so ergibt sich eine minimale Deadline von $3.62ms$. Der zum TTP-Ergebnis noch verbleibende Unterschied liegt nun ausschließlich in der Überapproximation der Nachrichtenlänge durch die Verwendung von immer vollständigen Frames mit 16 Byte Nutzdaten und durch die Verwendung mehrerer Frames für Nachrichten, deren Länge 16 Bytes übersteigt (und dem damit

zusätzlich einhergehenden Frame-Overhead). Nutzt man stattdessen die Möglichkeiten des TTP-Protokolls, verschieden lange Frames zu versenden (mit bis zu 246 Bytes Nutzdaten), so kann diese Überapproximation vermieden werden, indem jede Nachricht in einem einzigen Frame verpackt und verschickt wird.

Analysiert man statt des TTP-Ansatzes eine Nachrichtenübertragung auf Basis des statischen Anteils des FlexRay-Protokolls, so muss berücksichtigt werden, dass FlexRay eine identische Größe aller Frames und aller Slots vorschreibt (alternativ darf ein Slot nicht existent sein, also die Länge 0 besitzen). Erneut verwenden wir hierfür einen 42-Bit Header und Frames mit 16 Byte Nutzdaten-Anteil, schalten aber die Möglichkeit, mehrere Slots pro TDMA-Round an einen Prozessorknoten zu vergeben, ab. Dann ergibt die Analyse des oben betrachteten Beispiels für die kleinste zu erreichende Deadline die Ergebnisse, die in Tabelle 5.22 dargestellt sind, wobei dies für 2 Versuche durchgeführt wurde: in Versuch 1 ist einer der Slots mit der Zusatzbedingung ausgestattet worden, mindestens eine Slotlänge von 2 Frames ($\psi(s_i) \geq 2$) zu besitzen. In Versuch 2 existiert diese Bedingung nicht. In beiden Fällen konnte die minimal zu erreichende Deadline von $\Delta_\gamma^{\min} = 7.75ms$ ermittelt werden.

Nachricht	$\Delta t \ [ms](s_i = 19\rho)$	$\Delta t \ [ms](s_i = \rho)$
$\tau_0 \rightarrow \tau_1$	1.02	5.44
$\tau_1 \rightarrow \tau_3$	0.85	5.37
$\tau_1 \rightarrow \tau_5$	0.82	4.22
$\tau_2 \rightarrow \tau_3$	0.83	5.31
$\tau_4 \rightarrow \tau_6$	0.55	2.25
$\tau_5 \rightarrow \tau_6$	0.48	2.86
$\tau_9 \rightarrow \tau_{11}$	0.48	2.86
$\tau_{13} \rightarrow \tau_{15}$	1.02	5.44
$\tau_{14} \rightarrow \tau_{15}$	0.82	4.22
$\tau_{16} \rightarrow \tau_{17}$	1.02	5.44
$\tau_{25} \rightarrow \tau_{26}$	1.09	5.51
$\tau_{33} \rightarrow \tau_{35}$	0.61	2.31
$\tau_{34} \rightarrow \tau_{35}$	0.27	0.61
$\tau_{35} \rightarrow \tau_{36}$	0.0	0.0
$\tau_{38} \rightarrow \tau_{40}$	1.02	6.12
$\tau_{39} \rightarrow \tau_{40}$	1.02	5.44

Tabelle 5.22: Differenz der Antwortzeiten der Nachricht auf einem FlexRay-basierten Bussystem mit nur statischem Anteil und nur einmaliger Slotbenutzung pro TDMA-Round zur minimalen Deadline bei einer Übertragungsrate von 250 Bytes/ms.

Bei diesen beiden vergleichenden Versuchen ist zu beobachten, dass zwar in beiden Varianten die gleiche minimale Deadline erreicht werden kann, dass aber im Falle größerer Slots der verbleibende Spielraum deutlich geringer wird. Dies liegt an der Länge der TDMA-Round, die vom Optimierungsverfahren bei größeren Slots auf einen Wert von 19 Frames pro Slot angehoben werden muss, um das optimale Ergebnis zu erreichen. Gemäß der Gleichungen zur Berechnung Antwortzeit von

Nachrichten (vgl. Gleichung 4.12 auf Seite 70) geht die Länge der TDMA-Round aber direkt als Blockierungsfaktor in die Antwortzeit mit ein, so dass diese bei längeren TDMA-Rounds ebenfalls verlängert wird. Demhingegen konnte das optimale Ergebnis bei freier Gestaltung der Slots für eine minimale TDMA-Round-Länge gefunden werden, bei dem alle Slots nur einen Frame versenden.
Im Vergleich zum TTP-Bus ergibt sich somit eine wesentlich unflexiblere Ausnutzung der Übertragungskapazität, die auf die strenge Vorschrift zu gleichen Slot- und Frame-Längen des FlexRay-Protokolls zurückzuführen ist. Allerdings lässt sich dieser Nachteil wiederum kompensieren, sobald die Eigenschaft, mehrere Slots innerhalb einer TDMA-Round dem gleichen Prozessor zuzuordnen, ausgenutzt wird. Im Grenzfall kann das Verhalten auf dem statischen Anteil des FlexRay-Busses an das Ergebnis auf einem TTP-Bus angenähert werden, wenn als Slotgröße genau 1 Frame verwendet wird und größere Slots auf dem TTP-Bus durch das FlexRay-Protokoll durch sukzessive Aneinanderreihung einer entsprechenden Anzahl an kurzen Slots für den gleichen Prozessorknoten emuliert werden. Insofern stellt FlexRay eine identische Flexibilität wie TTP zur Verfügung, die aber durch Einschränkungen bei der möglichen Anzahl der Slots innerhalb einer TDMA-Round aufgrund von Controller-Restriktionen beschränkt werden können. Da FlexRay ein relativ neues Protokoll ist und bis jetzt keine kommerziell verfügbaren Bus-Controller existieren, kann die Größenordnung der durch die Hardware vorgegebenen Restriktionen zum jetzigen Zeitpunkt nicht näher eingegrenzt werden[1]. Setzt man eine für die jeweilige Probleminstanz hinreichend große Menge an verfügbaren Slots auf dem FlexRay-Bus vorraus, so besteht letztlich nur noch ein Unterschied bzgl. der Anzahl der Nutzdaten eines Frames im Vergleich zum TTP-Bus: Während letzterer bis zu 246 Bytes Nutzdaten in einem Frame verpacken kann, muss auf dem FlexRay-Protokoll mit seinen maximal 16 Nutzdaten-Bytes pro Frame die Nachricht aus mehreren Frames zusammengesetzt werden, was jeweils einen entsprechenden Overhead durch den Header der Frames erzeugt.
Leider kann diese Argumentation nicht durch experimentelle Ergebnisse abgesichert werden, da die Komplexität des SAT-Checking-Problems bei der Benutzung von multipler Slotzuweisung auf einem FlexRay-Bus gemäß Kapitel 5.3.11 selbst bei kleinen Instanzen unlösbar groß wird. Hier könnten nur weitreichende, problemspezifische Optimierungen der Gleichungsstrukturen greifen, die aber im Kontext dieser Arbeit nicht mehr durchgeführt wurden. Sollte selbst dies keinen nennenswerten Gewinn bringen, könnte noch die Lösungsmethodik verändert werden, wie es in Kapitel 5.4.3 bereits diskutiert wurde, beispielsweise durch Verwendung eines speziellen RT-SAT-Checkers.

Als letztes soll hier nun noch der Vergleich der Optimierung der gegebenen Probleminstanz auf einem gemischt Ereignis- und Zeitgetriebenen Bussystem angestellt werden. Hierzu wird ein dem FlexRay-Protokoll ähnliches System evaluiert, in dem es eine Reihe von Zeit-basierten Slots (pro TDMA-Round allerdings nur maximal einen für jeden Prozessor) und einen einzelnen Ereignis-basierten Slot gibt: Der Zeitgetriebene Anteil setzt sich wiederum aus Frames mit 16 Byte Nutzdaten und 42-Bit

[1] In der FlexRay-Spezifikation ist hier eine Beschränkung von 1028 Slots vorgesehen. Es ist jedoch zu erwarten, dass es derartige Restriktionen geben wird.

Header zusammen. Ebenso gelten die beim FlexRay spezifizierten Beschränkungen bzgl. gleicher Slot- und Frame-Größen. Der Ereignis-basierte Anteil des Bussystems soll, im Gegensatz zum dynamischen Anteil des FlexRay-Busses, einer CAN-Bus Variante entsprechen, wobei die Frames äquivalent den Frames im Zeit-basierten Teil sind. Dies soll eine Vergleichbarkeit ermöglichen, auf welchem Teilbus Nachrichten unter Optimalitätsgesichtspunkten besser anzusiedeln sind. Die Messungen für die oben beschriebene Probleminstanz ergeben eine minimale Deadline für die Nachrichten von $\Delta_\gamma^{\min} = 3.67ms$, wobei keine Nachricht bei dem optimalen Ergebnis in einen Zeit-basierten Slot allokiert wurde, sondern alle in den Ereignis-basierten Slot migriert sind. Dies deckt sich mit den Beobachtungen aus [178], wobei dort allerdings verschärfend jeder Nachricht ein Slot der Länge der Nachricht zugeordnet wurde und damit ohne die Verschiebungen der Nachrichten in den dynamischen Slot das System nicht mehr lösbar war. Hier hingegen führt die Benutzung statischer Slots ebenfalls zu gültigen Konfigurationen, die aber ein deutlich schlechteres Optimierungspotential aufweisen.
Wird andererseits gefordert, dass mindestens ein statischer Slot benutzt werden soll, so migrieren alle Nachrichten bis auf eine in den dynamischen Slot. Die minimal erreichbare Deadline liegt in diesem Fall bei $\Delta_\gamma^{\min} = 6.25ms$, also deutlich über der rein dynamischen Abwicklung der Nachrichten. Aufgrund der größeren Flexibilität der Ereignis-basierten Nachrichtenübertragung ist dieses Ergebnis nicht unerwartet. Trotzdem kann es gute Gründe geben, zumindest einen Teil der Nachrichten in den Zeit-basierten Slots zu versenden, beispielsweise um sogenannte “Babbling Idiot”-Fehler einzuschränken. In derartigen Fällen muss dem Platzierungsverfahren dann explizit die Vorgabe mitgegeben werden, dass ein Zeit-basierter Slot zu benutzen ist, da ansonsten die Optimierung jeweils alle Nachrichten in den Ereignis-basierten Slot verschieben würde.

5.7 Zusammenfassung

Insgesamt zeigen die Untersuchungen der letzten Abschnitte, dass mit der Benutzung von Entscheidungsverfahren der Erfüllbarkeit ein Optimierungsverfahren für die Platzierung von Tasknetzwerken in verteilten Echtzeitsystemen zur Verfügung steht, welches tatsächlich in der Lage ist, eine optimale Lösung einer Probleminstanz zu finden. Dies ist auch angesichts der doch recht hohen Komplexität des Suchraumes von 2^{200000} und mehr möglich. Der wesentliche Grund einer effizienten Exploration derartig großer Entwurfsräume liegt an der Struktur der verwendeten Klauseln innerhalb der Modellierung des Platzierungsproblemes. Betrachtet man die in Kapitel 5.3.4 aufgestellten Ungleichungssysteme, so fällt auf, dass die Belegung eines Großteils der Integervariablen (beispielsweise Antwortzeiten, Anzahl der Preemptionen, etc.) direkt von den primären Entscheidungsvariablen (Platzierung der Tasks, Platzierung der Nachrichten und teilweise Prioritäten) abhängen. Somit lassen diese sich mehr oder minder innerhalb der Deduce-Phase des SAT-Checkers mit Belegungen versorgen. Der Suchraum, der innerhalb der Decision-Phase des Solvers exploriert werden muss, sollte demnach im wesentlichen durch die Anzahl der primären Entscheidungsvariablen determiniert sein und nicht so sehr durch die An-

zahl der Booleschen Variablen insgesamt. Als Resultat führt das zu einer deutlich geringeren Anzahl an Backtracking-Aufrufen und damit zu ertäglichen Laufzeiten des Verfahrens. Dies legt den Schluss nahe, dass bei der Variablenauswahl in der Decision-Phase evtl. eine Priorisierung der primären Variablen einen Performanzgewinn bewirken könnte, da in den bisher verwendeten Heuristiken auch sekundäre Variablen innerhalb der Decision-Phase gleichberechtigt zu den primären Variablen mit geratenen Belegungen versorgt werden können. Die Effekte auf das Erkennen und Aussortieren großer Teilräume des Suchraumes müssen dann allerdings intensiv evaluiert werden, da hier nicht klar ist, welchen Anteil an gelernter Konfliktvermeidung sekundäre Variablenbelegungen innerhalb der Decision-Phase erzeugen.
Die Abgrenzung des Ansatzes gegenüber anderen Optimierungsverfahren ist neben der Vollständigkeit insbesondere auch in der universellen Anwendbarkeit zu erkennen. Ersteres konnte im Vergleich zu heuristischen oder stochastischen Optimierungsverfahren gezeigt werden, letzteres zeigt sich in der nahezu freien Verwendung unterschiedlichster Kommunikationsmedien, und wir werden diese generelle Stärke auch in den nächsten Kapiteln wieder nutzen, wenn es um Entwurfsprozessbegleitende Optimierungen geht.
Zudem eignet sich dieses Verfahren auch in hierarchisch aufgebauten komplexen Topologien unter Verwendung unterschiedlicher Kommunikationsmedien und bildet damit typische industrielle Strukturen von Netzwerken elektronischer Steuergeräte ab. Möglich gemacht wird dies durch die inhärenten Eigenschaften von Lokalität, hervorgerufen durch Abhängigkeiten der Tasksysteme von Sensorik und Aktuatorik, so dass letztlich die zu durchsuchenden Entwurfsräume stark eingeschränkt sind, wenngleich sie für vollständige Suchverfahren immer noch viel zu groß sind. Die Parallelisierung des Optimierungsprozesses vermeidet zudem die Behinderung durch Probleminstanzen mit großen Laufzeiten, die durch die heuristische Belegungstechnik hinter dem Entscheidungsverfahren verursacht werden und ermöglicht damit Laufzeiten, die im Entwurfsprozess von eingebetteten Echtzeitsystemen tolerierbar sind. Lediglich Platzierungen für Systeme mit Bussystemen, die die multiple Slotzuweisung beinhalten, scheitern an der Komplexität der erreichten Modellierung, so dass an dieser Stelle andere Verfahren (wie etwa in Kapitel 5.5 skizziert) eingesetzt werden müssen, oder aber Erweiterungen des Ansatzes zum Einsatz kommen müssen.
Die Größenordnung der Systeme, die mit dem vorgestellten Verfahren automatisch optimiert bzw. platziert werden können, hängen natürlich von der Komplexität der Architektur und den Freiheitsgraden der Platzierung einzelner Tasks eines Systems ab. Für die gezeigten Beispiele ist mit Tasksystemen bis zu einer Größe von ca. 50 Tasks und gleichzeitig Architekturen mit bis zu 20 Prozessorknoten eine automatische Platzierung in erträglicher Zeit möglich. Komplexere Topologien erzeugen hier jedoch einen erheblichen Aufwand, wenn die zu platzierenden Tasks keine große Lokalität aufweisen. In Verbindung mit einer inkrementellen Integration, wie sie im nächsten Kapitel eingeführt wird, gelingt es jedoch, auch wesentliche größere Systeme einer automatischen Platzierung zugänglich zu machen. Weitere Optimierungen des Ansatzes können zudem dafür sorgen, dass auch größere Systeme in angemessener Laufzeit beherrschbar sind. Als Schwerpunkt dieser Optimierungstechniken, die in Kapitel 5.4.3 eingehend erläutert wurden, sollten vorrangig Methoden des Lernens

zwischen den iterativen Aufrufen der Entscheidungsprozedur innerhalb einer binären Suche, sowie die Integration eines speziell auf Echtzeitsysteme zugeschnittenen externen Solvers in das eigentliche Erfüllbarkeitstest-Verfahren als vielversprechendste Kandidaten behandelt werden.

Während der Beschreibung des Platzierungsverfahrens innerhalb dieses Kapitels, aber auch bei der Formalisierung dieses Verfahrens in Kapitel 4, wurde oftmals die inkrementelle Integration und als Ausdruck dessen die synthetischen, lokalen Deadlines von Tasks und Nachrichten erwähnt. In dem nun folgenden Kapitel soll die Platzierung von Tasknetzwerken unter dem Gesichtspunkt eines Entwicklungsprozesses für verteilte eingebettete Echtzeitsysteme betrachtet werden, und wir werden die spezifischen, im Platzierungsverfahren ausgenutzten, Eigenschaften aus der Beschreibung typischer Entwicklungsszenarien in der hier betrachteten Anwendungsdomäne heraus ableiten. Dies wird im wesentlichen von einer Ist-Analyse des Standes der Entwicklungsprozesse eingebetteter Echtzeitsysteme getrieben, wie sie nach einer Einführung abstrahierter und relozierbarer Implementierung, beispielsweise durch Prozesse wie etwa die AUTOSAR-Initiative, Stand der Technik ist. Letztlich wird dann die globale Integration des in dieser Arbeit skizzierten Vorgehens in der Exploration des Entwurfsraumes andiskutiert.

Kapitel 6

Integration in industrielle Entwurfsprozesse

6.1 Generelle Methodologie

Gemäß den einleitenden Ausführungen zum Entwurfsprozesse komplexer eingebetteter Echtzeitsysteme in Kapitel 1.3 soll im folgenden von einer Design Methodologie ausgegangen werden, wie sie beispielsweise die Rich Component Models in [37] vorschlagen. D.h. Systeme werden auf unterschiedlichen Abstraktionsebenen in Form von Komponenten spezifiziert und implementiert, die jeweils bestimmte Eigenschaften in den jeweiligen funktionalen und nicht-funktionalen Sichten sicherstellen. Wie bereits in Kapitel 1.3 beschrieben, ist die Einbettung der im Kontext dieser Arbeit vorgestellten Platzierungsverfahren in die generelle Design-Methodologie an der Schnittstelle zwischen abstrakteren Modell-basierten Spezifikationen des Designs und den Grundelementen auf dem sogenannten ECU-Layer (vgl. [37]) angesiedelt und übernimmt dort die Funktion einer Abbildung aus einem Funktionsnetzwerk auf eine Architektur und die zugehörige Platzierung der Basiskomponenten auf dieser Architektur. Wir werden folglich hier nur eben jene Abstraktionsebene darstellen, die sich mit konkreten Architekturen und Tasks bzw. Nachrichten beschäftigt und damit auch der vertikalen Komposition im Sinne von [37] keine weitere Beachtung schenken.
Komponenten sind hierbei hierarchisch aufgebaute Design-Einheiten, die auf der höheren Ebene eine Funktion oder ein Feature darstellen und im Sinne des Concurrent Engineering parallel den Entwurfsprozess, beispielsweise das V-Modell [49], durchlaufen. Der Vorteil einer solchen Vorgehensweise liegt auf der Hand: Bereits früher eingesetzte Komponenten können so relativ einfach einem Re-Use zugeführt werden und tragen damit maßgeblich zur Verkürzung von Design-Zyklen bei.
Wir gehen im folgenden von dem in Abbildung 6.1 an einem Beispiel dargestellten hierarchischen Komponenten-Modell aus, dessen Basiselemente Tasks oder Nachrichten zwischen Tasks sein können.

Die Basiselemente *Spezifikationen* in den einzelnen Komponenten spezifizieren weitere Bedingungen für die Komponente, beispielsweise End-to-End-Deadlines, die eingehalten werden müssen. Die Einhaltung derartiger Anforderungen wird mittels ei-

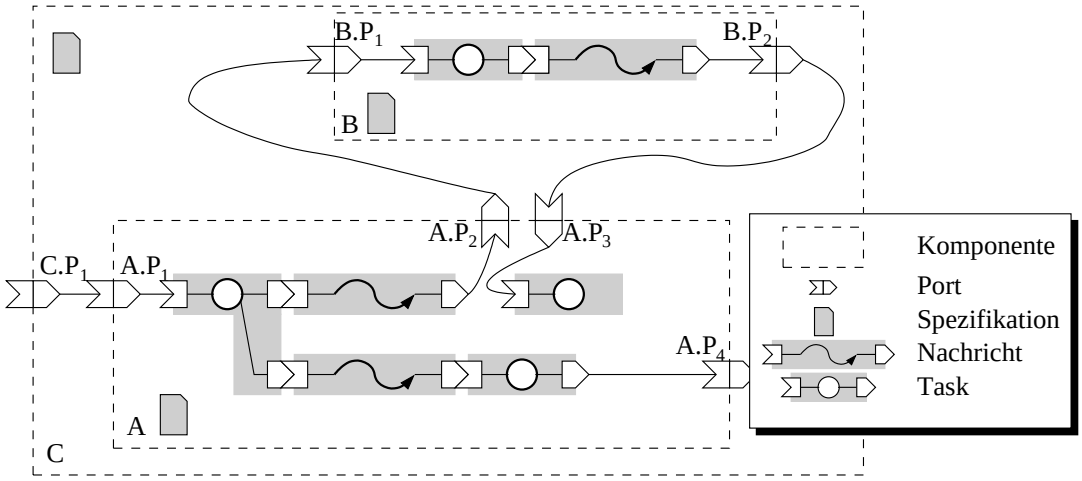


Abbildung 6.1: Beispielhaftes hierarchisches Komponentenmodell mit drei Komponenten A,B und C, dessen Blätter aus Basiselementen bestehen müssen. Jede Komponente verfügt über Ports, Verbindungen zwischen Komponenten und/oder Basiselementen und einer komponentenspezifischen Spezifikation.

ner horizontalen Komposition überprüft, die über Pfaden des Instanzgraphen der Komponenten und deren jeweilig zugesicherten Eigenschaften argumentiert. In dem Beispiel in Abbildung 6.1 könnte z.B. die Einhaltung einer End-to-End-Deadline d_{E2E} für Komponente C für den Pfad $C.P_1 \rightarrow A.P_1 \rightarrow A.P_4 \rightarrow C.P_2$ analysiert werden. Für eine genauere formale Beschreibung der horizontalen Komposition und der dafür notwendigen Analysetechniken sei hier auf [37] verwiesen, in der dies nicht nur für die Echtzeit-Sicht formal beschrieben ist. Zum generellen Verständnis genügt an dieser Stelle eine Interpretation der Pfadanalyse in Form einer Gleichung $d^A_{E2E} \geq \sum r_i + \sum r_{g_{i+1}} + \sigma$ in Komponente A, sowie eine Gleichung $d_{E2E} \geq d^A_{E2E}$, wie sie auch bei der Formalisierung des Schedulingproblems in Kapitel 4.1 verwendet wurde. d^A_{E2E} ist hierbei die Zusicherung von Komponente A bzgl. des zeitlichen Verhaltens unter gewissen Annahmen, wie beispielsweise Periodizität und ähnlichem. Nachgewiesen wird diese Zusicherung durch das Mapping der Basiselemente auf eine gegebene Architektur gemäß des Platzierungsalgorithmus, der inhärent die damit verbundene notwendige Echtzeitanalyse enthält.
Die Verbindung über Ports enthält die für die Betrachtung der Echtzeit-Sicht notwendigen Zusicherungen und Anforderungen an die jeweilige Komponente. Ports sind zunächst zwecks konsistenter Verbindungen getyped, verfügen aber noch über weitere Informationen, die sogenannten Services (vgl. wiederum [37]). Diese geben auf dem hier betrachteten Abstraktionsniveau (Tasks und Nachrichten) Informationen über die unterschiedlichen Pfade im Taskgraph einer Komponente wieder und werden genutzt, um beispielsweise End-to-End-Deadlines für derartige Pfade innerhalb einer Komponente definieren zu können, ohne die konkrete Implementierung des Pfades berücksichtigen zu müssen. Sie stellen also eine Art der Kapselung dar und unterstützen somit die sogenannte Black-Box Sicht einer Komponente, die nur über den Schnittstellen der Komponente definiert ist, aber keine weiteren Interna sichtbar macht. Im Beispiel in Abbildung 6.1 können auf diese Weise in Port $A.P_1$ zwei unterschiedliche Services angeboten und mit Eigenschaften versehen werden,

die den beiden unterschiedlichen Pfaden im Taskgraph von Komponente C entsprechen.
In Kapitel 6.2.2 wird anschließend detailliert diskutiert werden, wie Ports letztlich ausgestaltet werden müssen, um sie aus Echtzeit-Sicht in einem Assume-Guarantee-Ansatz anwenden zu können. Wir werden dabei allerdings keine weitere Betrachtung von Services erörtern; hier sei auf [37] bzw. [172] verwiesen.
Nicht gezeigt sind in Abbildung 6.1 spezifische Eigenschaften der Basiselemente und vertikale Anforderungen und Zusicherungen an höhere oder tiefere Abstraktionsebenen, wie sie in einer durchgängigen Design-Methodologie, wie beispielsweise den Rich Component Models, verwendet werden; diese werden als gegeben angenommen (vgl. die Darstellung in Kapitel 2.4).

Für den Aufbau einer konkreten Architektur kann dies vollkommen analog in Form von vernetzten Komponenten erfolgen, nur dass Basiselemente dort Steuergeräte und Kommunikationsmedien sind. Mit dieser Trennung ist aber eine Unabhängigkeit der Software-Funktionen und der Architektur gegeben, wie sie beispielsweise auch in dem AUTOSAR-Projekt [69] angestrebt wird. Dies erlaubt letztlich die Anwendung des in Kapitel 5 vorgestellten Platzierungsverfahrens und damit eine Lockerung des nahezu linearen Zusammenhangs zwischen der Anzahl der Funktionen und der Anzahl der Steuergeräte. Das Zusammenspiel zwischen Komponentenmodell und Platzierung wird in Kapitel 6.2.1 im Zusammenhang mit der inkrementellen Integration näher erläutert.
Kontrakte mit Zulieferern können auf Basis der Komponenten abgeschlossen werden, die Architekturanteile besitzen können, aber nicht müssen. Zeitliche Anforderungen werden in den Schnittstellen der Komponenten definiert und ermöglichen somit bereits in frühen Entwurfsphasen eine virtuelle Integration in Form einer horizontalen Komposition, wie oben dargestellt. Auf welche Art und Weise die zeitlichen Bedingungen in den Schnittstellen ermittelt werden können, wird in den nachfolgenden Kapiteln eruiert, insbesondere wird in Kapitel 6.5 die virtuelle Integration unter Berücksichtigung nur sehr grober Informationen diskutiert.

6.2 Inkrementelle Integration

6.2.1 Komponentenbasierung und Integration in Zulieferer-dominierten Prozessen

Ein wesentliches Merkmal in der Entwicklung komplexer technischer Systeme, wie es beispielsweise in der Automobilindustrie, aber auch im Bereich der Luftfahrt- oder Bahnindustrie anzutreffen ist, ist die enge Kopplung zwischen Systemhersteller (im folgenden OEM genannt) und Zulieferern. OEMs erstellen nur wenige Teile des Systems selbst, sondern sehen sich zunehmend in der Rolle des Integrationsverantwortlichen von Subsystemen (vgl. [25]). Dies hat zur Folge, dass es unter Umständen viele komplizierte und teure Design-Iterationen aus der Integrationsphase heraus gibt, da erst dort das Zusammenspiel der Komponenten unterschiedlicher Hersteller in einem Gesamtsystem zu testen ist. Aufgrund enger Time-to-Market-Margen findet zudem ein massives Concurrent Engineering statt, für dass insbesondere auch

die Echtzeitanforderungen vom OEM in einer frühen Entwurfsphase auf die einzelnen Subsysteme sinnvoll zugeschnitten werden müssen. Diese Vorgehensweise macht insgesamt die Gestaltung einer Software- und Hardware-Architektur mit vielen Freiheitsgraden enorm schwierig, was letztlich dazu führt, dass sowohl Zulieferer als auch OEMs bei der Implementierung neuer oder alter Funktionen oder Features lieber propreitäre Steuergeräte verwenden[52]. Dies erleichtert auf der einen Seite die Integrationsphase, hat aber auf der anderen Seite ein enormes Komplexitätsproblem zur Folge. Hinzu kommt, dass eine zunehmende Diversifizierung, beispielsweise der Fahrzeuge in der Automobilindustrie, komplexe unterschiedliche Konfigurationen nach sich zieht und zudem in naher Zukunft auch Software-Upgrades von bereits im Einsatz befindlichen Systemen ein zunehmend gefordertes Feature sein wird.

Die Kombination aus Komponentenbasierung und möglichst freier Platzierbarkeit der Basiselemente soll nun dazu beitragen, diese Komplexität der Integrationsphase beherrschbarer zu machen. Dazu haben wir bereits in Kapitel 5 die automatische und optimierende Allokation vorgestellt und wir wollen dies hier durch eine sogenannte inkrementelle Integration ergänzen. Basierend auf den in frühen Designphasen festgelegten Komponentenschnittstellen und der Beobachtung, dass zwar ein gewisser Verteilungsspielraum bei der Zuordnung der Tasks zu Steuergeräten erwünscht ist, dieser jedoch aufgrund von Domänenzugehörigkeit und Sensor-/Aktuatornähe niemals vollständig frei sein wird, schlagen wir eine inkrementelle Integration von Teilsystemen zu einem Gesamtsystem vor. Inkrementell meint hier, dass Teile des Systems nacheinander, in sinnvollen Einheiten, integriert werden. Dies erniedrigt die Komplexität der Integration, wenn gleichzeitig alle Zuliefererkomponenten ihre Schnittstellenvereinbarungen einhalten. Zudem erleichtert es den Umgang mit unterschiedlichen Lieferzeiten, die im Concurrent Engineering getriebenen Entwurf zwangsläufig entstehen. Wesentlicher Bestandteil einer inkrementellen Vorgehensweise ist die Garantie, dass neu hinzugefügte Subsysteme niemals die Eigenschaften der bereits integrierten Systeme invalidieren, aus Sicht eines Echtzeitsystems also beispielsweise niemals Deadlines überschritten werden, die im Schritt vorher als valide eingestuft wurden.

Methodologisch bedeutet dieses Vorgehen die in Abbildung 6.2 dargestellte Integration der in Abbildung 6.1 vorgestellten Komponentenmodelle für Tasksysteme und Architekturen.

Eine konkrete Implementierung besteht zunächst aus einem Komponentenmodell für Tasksysteme und einem Komponentenmodell der Architektur. Beide Modelle werden unter einer gemeinsamen Implementierungssicht vereinigt, indem Implementierungsspezifikationen das Mapping der Tasks und Nachrichten auf die Architekturkomponenten definieren. Dies ähnelt in gewisser Weise dem Vorgehen im Metropolis-Projekt [10]. Die Implementierungsspezifikation dient gleichzeitig als Repository, welches die derzeit gültige Systemkonfiguration enthält und wird als Basis weiterer inkrementeller Integrationsschritte genutzt. Integration bedeutet in diesem Zusammenhang das sukzessive Erweitern sowohl der Komponentenmodelle als auch der Implementierungsspezifikation, wobei die neu aus einer Integrationsdatenbank hin-

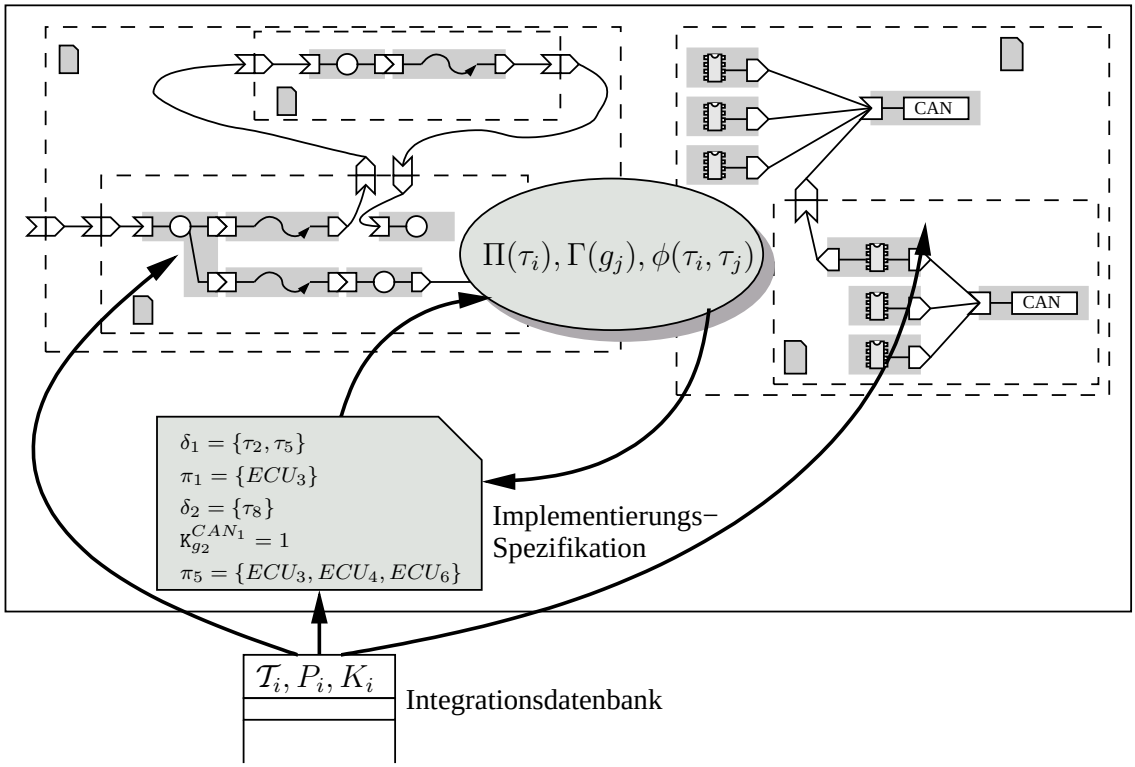


Abbildung 6.2: Zusammenführen von Komponentenmodellen, Architekturmodellen und der optimierenden automatischen Platzierung mittels inkrementeller Integration. Die Pfeile aus der Integrationsdatenbank deuten an, dass in jedem inkrementellen Schritt sowohl das Komponentenmodell des Tasksystems als auch der Architektur schrittweise erweitert werden.

zugekommenen Subsysteme noch die Freiheiten ihrer Platzierung behalten. Neu zu integrierende Subsysteme bestehen also immer aus einem Tasksystem und einem Architekturanteil, der aber auch leer sein kann.
Die Allokation von Tasks und Nachrichten auf eine Architektur (in Form einer Anwendung der Abbildungen Π, Γ und ϕ) ist in diesem Sinne eine Spezialisierung der Implementierungsspezifikation. Die Platzierung findet dabei auf einer White-Box-Sicht (vgl.[37]) der Komponenten statt und wird durch bereits in der Implementierungsspezifikation befindliche Regeln sowie durch neue Regeln, induziert durch ein zusätzlich zu integrierendes Subsystem aus der Integrationsdatenbank, beschränkt. Die White-Box-Sicht selbst bietet die Sichtbarkeit aller Basiselemente des Systems über alle Hierarchiestufen hinweg an und ist die Grundlage einer Platzierungsmethodik, wie sie in Kapitel 5 angegeben wurde. Sollen weitere gruppierend wirkende Konstrukte, wie beispielsweise Container[1] (vgl. [69]), verwendet werden, so kann dies prinzipiell mittels zusätzlicher Platzierungsbeschränkungen in die Implementierungsspezifikation mit aufgenommen werden[2]. Restriktionen bzgl. der Platzierung

[1]Container bezeichnen eine Menge von Komponenten oder Basiselemente des Tasksystems, die immer zusammen platziert werden müssen.

[2]In der aktuellen Modellierung, wie sie in Kapitel 4.1 definiert wurde, ist dies noch nicht möglich. Diese ist jedoch ohne großen Aufwand erweiterbar und auch eine Integration in die Platzierungstechnik mittels SAT-basierten Techniken oder anderen Verfahren ist problemlos. Hier ist dann

eines Subsystems beschränken sich zunächst nur auf die eigenen Architekturanteile des Subsystems sowie auf Komponenten einer bereits bestehenden Systemarchitektur. Die Aufnahme eines neuen zu platzierenden Subsystems in die Implementierungsspezifikation erweitert diese Sicht auf die White-Box-Sicht dieser Komponente. Auf diese Weise ist es möglich, Architekturanteile oder Komponentenmodelle von Tasksystemen als nicht kompatibel (im Sinne einer verbotenen Platzierung oder einer Redundanz) zu markieren, ohne jedoch die genaue Zusammensetzung der jeweils referenzierten Komponente zu kennen. Stattdessen erfolgt der diesbezügliche Informationsabgleich automatisch im inkrementellen Integrationsschritt, der zusätzlich noch durch weitere manuelle Eingriffe erweitert werden kann.

Aus algorithmischer Sicht für die Anwendung des Platzierungsverfahrens bedeutet die inkrementelle Integration ebenfalls ein sukzessives Erweitern einer initialen Architektur und eines initialen Tasksystems. Algorithmus 5 zeigt die Integration des inkrementellen Ansatzes, wobei $\mathcal{T}_i$, P_i und K_i grundsätzlich aus der White-Box-Sicht extrahiert sind.

Algorithmus 5: Inkrementelles Platzierungsverfahren

Voraussetzung: Initiale Architektur $\mathcal{A} = \left(P_{init}, \mu^{\mathcal{A}}, K_{init}, \kappa\right)$ gegeben
  $\mathcal{T}_w := \emptyset$
  $P_w := P_{init}$
  $K_w := K_{init}$
  for $i = 1$ **to** n **do**
    $T_w := \mathcal{T}_w \cup \mathcal{T}_i$
    $P_w := P_w \cup P_i$
    $K_w := K_w \cup K_i$
    `generate` $\Pi_i : \mathcal{T}_w \rightarrow P_w$
    `generate` $\Gamma_i : \mathcal{T}_w \times (\mathcal{T}_w \times \mathbb{N}) \rightarrow K^{\leq N} \cup \emptyset$
    `generate` $\phi_i : \mathcal{T}_w \times \mathcal{T}_w \rightarrow \{0, 1\}$
    for each $\tau_j \in \mathcal{T}_w$ **do**
      $\pi_j := \{\Pi_i(\tau_j)\}$
    od
    ... // andere updates der Implementierungsspezifikation
  od

In jedem inkrementellen Schritt werden mit Hilfe der in Kapitel 5 vorgestellten Verfahren für die aktuelle Konfiguration spezifische Abbildungen erzeugt, die eine optimierte Platzierung von Tasks und Nachrichten bereitstellen. Die Spezialisierung der Implementierungsspezifikation ist hier angedeutet durch die Übernahme einer strikten, ein-elementigen Menge für die Menge der erlaubten Platzierungen. Weitere Updates der Implementierungsspezifikation, z.B. bzgl. der Wegewahl in einem hierarchischen Steuergerätenetzwerk, sind notwendig, werden aber in Kapitel 6.2.6 detailliert betrachtet. Hier sollen zunächst die aus der Methodologie abgeleiteten

lediglich noch zu definieren, ob Container als Zielobjekte nur Basiselemente oder auch Architektur-Komponenten akzeptieren.

Verfeinerungen der Schnittstellen für Echtzeitfähige Komponentmodelle des Tasksystems analysiert werden.

6.2.2 Schnittstellen für Echtzeit-fähige Komponenten

Die in den vorangegangenen beiden Kapiteln beschriebene Herangehensweise des inkrementellen, auf Komponenten basierenden Entwurfsprozesses verlangt nach eindeutigen Schnittstellen der Komponenten bzgl. der Echtzeit. Die horizontale Komposition im Sinne von Rich Component Models[37] ist nur dann sinnvoll durchzuführen, wenn die einzelnen Komponenten bestimmte Eigenschaften einhalten, beispielsweise ein Ausführungszeitintervall. Auch nur unter dieser Einhaltung macht die inkrementelle Platzierung Sinn: Komponenten können ihre Eigenschaften nur erhalten, wenn die benachbarten Komponenten deren Eigenschaften zusichern. Dieses Verhalten entspricht einem klassischen Assume-Guarantee-Ansatz, und es stellt sich daher die Frage, welche Bedingungen an der Schnittstelle einer Komponente zu definieren sind, die ein solches Vorgehen erlauben. Dazu soll zunächst die Illustration in Abbildung 6.3 betrachtet werden.

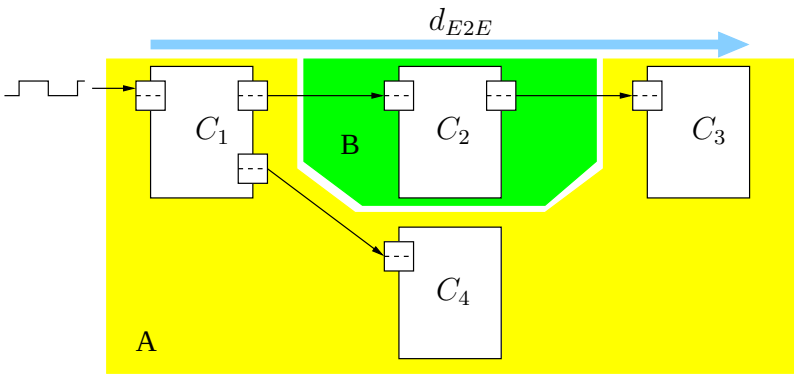


Abbildung 6.3: Inkrementelle Integration von Komponenten A und B, die aus Unterkomponenten C_1 bis C_4 bestehen und zwei Taskketten aufbauen, von denen eine durch die inkrementelle Vorgehensweise zerschnitten ist.

Dargestellt ist eine Vernetzung von vier Komponenten, die zwei Taskketten mit zugehörigen End-to-End Deadlines aufspannen. Die inkrementelle Platzierung soll nun zunächst die Komponenten in Bereich A platzieren und anschließend Bereich B. Die Überprüfung der Einhaltung der End-to-End Deadline für die obere Taskkette ist nun gemäß der klassischen Analyse, wie sie in Kapitel 2.3 eingehend dargestellt ist, nicht mehr möglich, da sowohl die Antwortzeit von Komponente C_2 als auch die Antwortzeiten der Nachrichten zu und von C_2 nicht bekannt sind. Ferner ist der Jitter, der durch C_2 und deren Nachrichten induziert wird, ebenfalls unbekannt. Damit ist es folglich im ersten Platzierungsschritt nicht nur nicht mehr möglich, die End-to-End Deadline zu testen, sondern es ergeben sich zudem Probleme bzgl. der wechselseitigen Abhängigkeiten von Antwortzeit und Jitter. Wir haben dies bereits in Kapitel 4.4.1 eingehend diskutiert und eine Lösung vorgeschlagen: Die Synthese von festen Werten für jede Komponente eines Systems. Ohne diese ist eine inkrementelle Strategie nicht durchführbar, da es zu viele Unbekannte im System gibt

und die der Platzierung zugrunde liegende Optimierung wenig Anhaltspunkte für die Optimalität einer Lösung hat.
Festgelegte Schnittstellen im obigen Sinne können für jede Komponente erzeugt werden, die als Ganzes in die inkrementelle Integration eingeht. Im Prinzip müsste dies also für die Taskkette $C_1 \rightarrow C_4$ nicht erfolgen und damit könnte der Platzierung die volle Analysegenauigkeit zur Verfügung stehen. Allerdings haben Deadlines und Jitter beispielsweise auch Einfluss auf andere Komponenten (vgl. wiederum die Ausführungen in Kapitel 4.4.1) und sollten daher ebenfalls fixiert sein. Aus diesem Grund wird im Kontext dieser Arbeit für jede Task und jede Nachricht eine entsprechende Schnittstelle definiert.
Ausschlaggebend für die Echtzeitanalyse sind in Tabelle 6.1 die notwendigen Schnittstellenobjekte für eine Task dargestellt.

In-Ports	Out-Ports
Aktivierungsperiode	Aktivierungsperiode ausgehender Nachrichten
Aktivierungsjitter	Aktivierungsjitter ausgehender Nachrichten
WCET	Aktivierungsoffset ausgehender Nachrichten

Tabelle 6.1: Mögliche In- und Out-Ports einer Komponente, die aus genau einer Task besteht.

Die einzelnen Objekte entsprechen den Notwendigkeiten einer sauberen Platzierung und einer funktionierenden horizontalen Komposition von Komponenten, wie sie in [37] vorgeschlagen wird. Wie bereits in den vorangegangenen Kapiteln ersichtlich war, definiert sich die Echtzeitanalye aus Aktivierungsperiode, Aktivierungsjitter und angegebener WCET einer Task. Die Ports Aktivierungsperiode und Aktivierungsjitter stellen Anforderungen an die horiontalen Nachbarkomponenten dar, die diese Task ansteuern. Die WCET einer Task hingegen ist eine Anforderung an eine vertikal tiefer liegende Schicht und wird beispielsweise mittels der Methoden aus Kapitel 2.4 erzeugt. Im Gegenzug garantiert die Task nach ihrer Platzierung die Echtzeiteigenschaften von allen Nachrichten, die sie versendet. Hierzu gehören Aktivierungsperiode, ebenso wie der entsprechende Jitter und ein Aktivierungsoffset. Aktivierungsperioden für ausgehende Nachrichten müssen nicht zwangsläufig der Aktivierungsperiode für Tasks entsprechen. So hat bereits Tindell in [181] gezeigt, wie Nachrichtenverschickungen mit einer geringeren Frequenz, beispielsweise weil die Task nur jede zweite Invokation diese Nachricht verschickt, in die Echtzeitanalyse eingebunden werden können. Aktivierungsoffsets entsprechen einer Garantie, zu welchem Zeitpunkt nach Aktivierung der Task diese Nachricht spätestens abgeschickt wird. Geht man davon aus, dass Nachrichten immer am Ende der Taskausführung versendet werden, so ist dies gleichzeitig die garantierte Deadline der Task. Andernfalls entspricht es den *Last*-Events, wie sie ebenfalls in [181] eingeführt wurden.
Dieses Schema kann für Nachrichten analog aufgebaut werden, wobei die WCETs dann einer Nachrichtenverschickung entsprechen und demnach nicht a priori festzulegen sind[1], sondern hier sind von einer vertikal tieferen Schicht die Informationen

[1] Nachrichten können ja, anders als Tasks, über mehrere Bussysteme versendet werden.

(Übertragungsdauer pro Paket, Anzahl der Pakete etc.) für jede Nachricht zu liefern. Da die Definition eines festen Jitterwertes sich ja, wie in Kapitel 4.4.1 auf Seite 97 bereits gezeigt wurde, auf die Deadline einer Task abstützt, verbleibt als wesentliche Aufgabe in der inkrementellen Integration die Generierung eben jener Deadlines aus dem System heraus. Dies soll in den nun folgenden Kapiteln erfolgen.

Es ist hier allerdings nochmals deutlich hervorzuheben, dass die Entscheidung für künstliche Schnittstellen bzgl. des Echtzeitverhaltens, die aufgrund der Unschärfe des Wissens zum Erstellungszeitpunkt dieser Schnittstellen nur heuristisch sein können, keine beliebig getroffene Wahl ist, sondern direkt durch die Art und Weise der Entwicklung komplexer eingebetteter Echtzeitsysteme vorgegeben ist. So ist es beispielsweise durchaus üblich und notwendig, einem Zulieferer einer Komponente noch vor dem Vorhandensein einer exakten funktionalen Spezifikation der Komponente zeitliche Garantien und Forderungen zu übermitteln (in etwa welchen Slot welcher Länge in einem TDMA-basierten Bussystem eine Nachricht der Komponente zugewiesen bekommt; vgl. auch Aussagen in [17] und [121]). Insofern stellen die im folgenden vorgestellten heuristischen Ansätze der Deadline-Generierung (und damit implizit auch der Jitter-Garantie, vgl. Kapitel 4.4.1) trotz der vorher verwendeten optimalen Verteilungsmethodik *keinen* generellen Bruch in der hier vorgestellten Methodik dar. Vielmehr bilden sie eine sinnvolle Adaptierung der Verfahren im Hinblick auf den Einsatz in industriell relevanten Entwicklungsumgebungen. Aufgrund der hier dargestellten und in der industriellen Wirklichkeit in Bezug auf die Entwicklung komplexer technischer Geräte inhärenten Zuordnung von Komponenten zu Zulieferern, ist an keiner anderen Stelle innerhalb des Entwurfsprozesses eine Aufspaltung globaler Aspekte in lokale, komponenten-zugehörige Aspekte sinnvoll. Möglich wäre hingegen noch, die heuristische Betrachtung für geschlossene Teilsysteme die in der Verantwortung eines Zulieferers liegen, zu vermeiden, jedoch sind derartig geschlossene Systeme sehr selten anzutreffen; Meist ist mindestens die Benutzung eines system- oder subsystem-weiten Kommunikationsmediums vorgesehen, womit ein Teilsystem nicht mehr als geschlossenen betrachtet werden kann. Diese Art der Erhaltung globaler Eigenschaften in Beziehung zur möglichen Optimalität der Platzierungsberechnung ist allerdings im Kontext dieser Arbeit aufgrund der besonderen Stellung geschlossener Teilsysteme keiner weitergehenden Betrachtung unterzogen worden, wird in der Literatur eingehend behandelt (vergleiche Kapitel 2.3) und ist problemlos in den hier dargestellten Ansatz integrierbar.

6.2.3 Deadline-Synthese für einfache Systeme

Echtzeitschnittstellen in einem Komponenten-orientierten Entwurfsstil, wie sie im vorangegangenen Abschnitt dargelegt wurden, basieren auf festen Zeitschranken für die Ausführungszeit von Tasks und die Übermittlungszeit von Nachrichten. Wie in den bisherigen Kapiteln an vielen Stellen bereits angedeutet und in den Abbildungen Δ_τ und Δ_γ in Kapitel 4.2 auf Seite 50 formalisiert, bedeutet dies die Synthese lokaler Deadlines für Tasks und Nachrichten aus einer aus der System-Spezifikation stammenden End-to-End-Deadline. Die Ableitung dieser lokalen Deadlines soll dabei möglichst wenig Verteilungsspielraum verschenken, damit das Optimierungsverfah-

ren in der Lage ist, Taskverteilungen einer hohen Güte zu erzeugen. Zwar werden gewisse Verluste unvermeidbar sein, aber das Ziel der hier vorgestellten Synthese-Verfahren ist die Minimierung dieser unweigerlich auftretenden Verluste bzgl. der Freiheitsgrade einer Platzierung. Da die Syntheseverfahren nur auf Heuristiken beruhen können[1], kann es keine absolute Bewertung der verschiedenen Ansätze — beispielsweise durch konkrete Zusicherungsintervalle — geben. Vielmehr wird sich jedes Verfahren an einem konkreten Verteilungsproblem beweisen müssen. Zu diesem Zweck werden im folgenden typische Szenarien verteilter, eingebetteter Echtzeitsysteme betrachtet, und die Synthese der Deadlines wird an ihnen gemessen, führt also zu einer Klassifizierung guter und schlechter Verfahren in Abhängigkeit von der Klassifizierung des Problems.

Es existieren eine ganze Reihe von Veröffentlichungen in diesem Bereich, die sich insbesondere in den frühen Neunziger Jahren dem Problem der Deadline-Synthese in End-to-End-Deadlines widmen. Beispielhaft seien an dieser Stelle die Arbeiten von [29], [2], [62] und [18] genannt. Ihnen gemein (wie auch fast allen anderen Arbeiten aus dieser Zeit) ist allerdings die Voraussetzung, dass die Zuordnung der Tasks auf Prozessoren, soweit überhaupt verteilte Systeme unterstützt werden (dies gilt beispielsweise nicht für [29]), bereits bekannt ist. Grundlage aller dieser Ansätze ist die Verwendung von Heuristiken, in denen sogenannte Execution-Windows für Tasks vergeben werden und unter diesen die Schedulability eines Systems berechnet wird. Die heuristischen Verfahren verändern hierbei, basierend auf der Analyse der Schedulability des Systems, die jeweiligen Execution-Windows in Richtung einer möglichst optimalen Verteilung der End-to-End-Deadlines auf einzelne Tasks. Da diese Arbeiten weder von einer unbekannten Platzierung ausgehen können, noch die Betrachtung von Nachrichten integrieren, sind sie für die im Kontext dieser Arbeit notwendigen Syntheseverfahren nicht geeignet. Allerdings legen sie den Grundstein für die wenigen weiterführenden Arbeiten im Bereich der Deadline-Synthese, die sich mit Systemen beschäftigen deren Taskzuordnung nicht vorab bekannt sein muss. Als wichtigste Veröffentlichungen hierzu sind [127] und [80] zu nennen, die beide Synthese-Verfahren zur Generierung von Deadlines in preemptiven, prioritätsbasierten Schedulingverfahren angeben und bewerten. Sowohl in [127] als auch in [80] werden einfache Metriken (im Gegensatz zu den komplexen heuristischen Verfahren in den vorher zitierten Arbeiten) erarbeitet, die eine gewisse Allgemeingültigkeit besitzen sollen, also versuchen durch geschickt gewählte und dem Problem angemessene Berechnungsvorschriften den unausweichlichen Verlust beim Platzierungsalgorithmus zu minimieren. [80] verfeinert hierbei die Prinzipien aus [127] und gelangt so zu durchweg besseren Ergebnissen. Wir werden im folgenden beide Methoden eingehender untersuchen und sie letztlich mit der im Kontext dieser Arbeit entstandenen Metrik vergleichen.

Sei eine End-to-End-Deadline λ gegeben durch ein $d_j^k = (\text{“}\tau_{j_0} \ldots \tau_{j_n}\text{“}, \lambda)$. Dann bezeichnet $\mathcal{C}_j^k \subseteq \mathcal{T}$ die Menge der Tasks, die in d_j^k die Taskkette bilden. [127] gibt nun zwei unterschiedliche Metriken an, die Deadlines einzelner Tasks einer Taskkette $\mathcal{C}_j^k$

[1] Optimale Verfahren können hier nicht zum Einsatz kommen, da eine Optimalität nur dann nachgewiesen werden kann, wenn die Platzierung der Tasks und Nachrichten bereits vorliegt. Dies soll aber gerade ausgeschlossen werden; Deadline-Synthese ist in diesem Sinne eine Vorstrukturierung eines Tasksystems für die Optimierungsphase.

zu berechnen:

$$R_{PURE} = \left(\lambda - \sum_{\tau_{j_i} \in \mathcal{C}_j^k} c_{j_i}\right) \frac{1}{|\mathcal{C}_j^k|} \tag{6.1}$$

$$\Delta_\tau(\tau_{j_i}) = c_{j_i} + R_{PURE} \tag{6.2}$$

$$R_{NORM} = \left(\lambda - \sum_{\tau_{j_i} \in \mathcal{C}_j^k} c_{j_i}\right) \frac{1}{\sum\limits_{\tau_{j_i} \in \mathcal{C}_j^k} c_{j_i}} \tag{6.3}$$

$$\Delta_\tau(\tau_{j_i}) = c_{j_i} \cdot (1 + R_{NORM}) \tag{6.4}$$

Die Metrik in Gleichung 6.1 und 6.2 berechnet zunächst den nicht durch die WCETs der Tasks einer Taskkette verbrauchten Anteil an der End-to-End-Deadline λ und teilt diesen anschließend in gleich großen Teilen jedem Task der Taskkette zu. Nachrichten werden nicht berücksichtigt, sondern sollen durch die Deadline der Sendertasks subsumiert werden. Die Entscheidung für ein derartiges Vorgehen begründet sich in [127] mit der Annahme, dass die Zeit für die Verschickung einer Nachricht i.a. sehr klein — gemessen an der WCET von Tasks — ist[1].
Gleichung 6.3 und 6.4 versucht durch eine Anteilsbasierte Metrik besser die unterschiedlichen Zeitbedürfnisse von Tasks zu berücksichtigen. Hier ist das Verhältnis der einzelnen Task-WCETs ausschlaggebend für die Größe des Zeitintervalls, das jede Task zusätzlich zu der Zeit für ihre WCET zugeteilt bekommt. Messungen in [127] als auch in [80] zeigen, dass dies zu durchweg besseren Ergebnissen führt.

In [80] wird die gleichmäßige Deadline-Metrik R_{PURE} erweitert, um das Wissen über interferenzfreie Parallelausführung von Tasks aufgrund einer verteilten Steuergerätearchitektur besser nutzen zu können. Die dort berichtete Methode mit den besten Performanzergebnissen nennt sich ADAPT-L und basiert auf der in den Gleichungen 6.1 und 6.2 angegebenen Metrik. Die Autoren versuchen, sowohl die anteilsbasierte Metrik als auch den möglichen Parallelitätsgrad von Tasks einer Taskkette auf einer gegebenen Architektur in einer vereinheitlichen Metrik für die Deadline-Synthese anzugeben. Zu diesem Zweck wird in Gleichung 6.1 und 6.2 die WCET c_{j_i} einer Task τ_{j_i} ersetzt durch eine den Systemgegebenheiten angepasste virtuelle Ausführungszeit $\tilde{c}_{j_i}$:

$$\tilde{c}_{j_i} = \begin{cases} c_{j_i} & \text{wenn } c_{j_i} < c_{thres} \\ c_{j_i}\left(1 + k_L\left(|\mathcal{T}| - |\mathcal{C}_j^k|\right) \cdot \frac{1}{|P|}\right) & \text{sonst} \end{cases}$$

Die Abhängigkeit von der Größe der einzelnen WCETs ist hier durch einen Threshold-Faktor c_{thres} gegeben. Dieser bewirkt, dass Tasks mit kleinen Ausführungszeiten

[1] Wir werden insbesondere bei den Messungen in Kapitel 6.2.5 und 5.6 noch sehen, dass diese Vereinfachung mitunter zu simplifizierend ist und wahres Systemverhalten nicht immer korrekt wiedergibt.

einen entsprechend kleineren Anteil an der verbleibenen Zeit einer End-to-End-Deadline zugeteilt bekommen, während Tasks mit größeren Ausführungszeiten diesen Anteil vergrößern können, indem sie einen von der möglichen Parallelität abhängigen Wert auf ihre WCET addieren und so die virtuelle Ausführungszeit berechnen. Die Größe des Threshold-Faktors c_{thres} ist üblicherweise mit der mittleren Ausführungszeit aller Tasks c_{AV} angegeben. Die Parallelität von Tasks wird durch die Differenzbildung aller Tasks des Systems mit den Tasks der betrachteten Taskkette, geteilt durch die Anzahl der Prozessorknoten ausgedrückt. D.h. es wird näherungsweise davon ausgegangen, dass Tasks einer Taskkette sich nicht unterbrechen können und dass alle Tasks gleichmäßig über alle Knoten verteilt werden können. Messungen in [80] zeigen, dass diese Metrik in den betrachteten Testsystemen den Metriken aus [127] grundsätzlich deutlich überlegen ist. Wir werden daher im folgenden alle Ergebnisse mit dieser Metrik vergleichen und die Metrik R_{NORM} aus [127] in einer abgewandelten Form übernehmen.

Bei der Übertragung dieser Metrik auf die im Kontext dieser Arbeit betrachteten Problemklasse sind jedoch zwei wesentliche Aspekte zusätzlich zu integrieren: Zum einen sind Taskketten nicht streng isoliert, sondern verschiedene Tasks können Teil mehrerer Taskkette sein. Hier ist dann eine entsprechende genauer zu differenzierende Berechnung der Deadlines dieser Task durchzuführen. Zum anderen müssen auch Deadlines für Kommunikationen integriert werden. Letzteres wird sowohl in [127] als auch in [80] nicht betrachtet bzw. führte zu schlechteren Ergebnissen. Die Idee, Kommunikationen nicht berücksichtigen zu müssen, lässt sich hierbei auf die Beobachtung zurückführen, dass Nachrichten mit Deadlines, die lokal übertragen werden, zwar keine Zeit verbrauchen, aber dennoch durch ihre Deadlines einen Teil der End-To-End-Deadline verbrauchen. Die in Kapitel 4.2 dargestellte Modellierung umgeht dieses Problem, indem die Deadline einer Task, die Nachrichten versendet, insofern dynamisch bleibt, als dass die den jeweiligen Nachrichten zugeordnete Deadline im lokalen Fall auch noch von der Task beansprucht wird. In diesem Sinne wird eine Verschwendung von Zeitanteilen einer End-To-End-Deadline vermieden und die Nachrichten können in der Deadline-Synthese ebenso wie die Tasks behandelt werden. Indirekt werden also auch in der hier vorgeschlagenen Methodik Tasks und ihre Nachrichten gemeinsam betrachtet.
Sind Tasks Teil mehrerer Taskketten, so werden ihnen zumindest in der ADAPT_L-Metrik womöglich unterschiedliche Deadlines zugeordnet. Da eine Task aber nur eine eindeutige Deadline haben kann, ist diese aus den Deadlines der unterschiedlichen Taskketten durch eine Minimumsbildung zu extrahieren. Das Minimum wird hier verwendet, um eine zeitlich enge Taskkette nicht in ihrer gesamten Ausführung durch eine große Deadline einer einzelnen Task zu schädigen. Somit ist die Deadline-Synthese zweistufig:

- In der ersten Phase werden alle Tasks, die Teil mehrerer Taskketten sind, mit einer Deadline versehen.
- Anschließend werden alle übrigen Tasks der Taskketten mit Deadlines versehen, die aus der End-To-End-Deadline minus der Deadlines aller derjenigen Tasks, die Teil mehrerer Taskketten sind, entstehen.

Bevor wir dies formalisieren, soll hier zunächst die Anteilsbasierte Metrik, wie sie [127] vorschlägt, leicht verändert werden. Sind nämlich Nachrichten mit ihrer worst-case Übertragungszeit mit berücksichtigt, bedeutet dies, dass die Summe aller WCET von Tasks und Nachrichten im schlimmsten Fall größer als die End-To-End-Deadline werden kann[1]. Daher kann Gleichung 6.4 aufgrund potentiell kleinerer Deadlines als die WCET einer Task nicht zur Anwendung kommen. Im folgenden wird die Deadline daher nicht aus der WCET plus einem additiven Faktor gebildet, sondern es werden direkt die Anteile einer Task/Nachricht an der WCET des gesamten Pfades benutzt, um den Anteil dieser Task/Nachricht an der End-To-End-Deadline zu bestimmen.
Sei $\mathcal{T}^{share} \subseteq \mathcal{T}$ die Menge der Tasks, die mehreren Taskketten mit spezifizierten End-to-End-Deadlines angehören. $\mathcal{T}^{share}$ kann durch die folgende Konstruktion angegeben werden:

$$\mathcal{T}^{share} = \left\{ \tau_i \;\middle|\; \tau_i \in \mathcal{T} \wedge \left| \bigcup_{\tau_j \in \mathcal{T}} \left\{ d_j^k \mid \forall d_j^k \in d_j \wedge d_j^k = (\text{“}\ldots\tau_i\ldots\text{“}, \lambda) \right\} \right| > 1 \right\} \tag{6.5}$$

Dann kann in der ersten Phase der Deadline-Synthese allen Tasks aus $\mathcal{T}^{share}$ eine lokale Deadline nach folgender Vorschrift zugeordnet werden:

$$\forall \tau_i \in \mathcal{T}^{share} : \Delta_\tau(\tau_i) = \min_{\forall \mathcal{C}_j^k : \tau_i \in \mathcal{C}_j^k} \left\{ \mathcal{S}_{\tau_i} \cdot (\lambda - \sigma) \right\}, \tag{6.6}$$

wobei σ dem Synchronisationsfaktor (vgl. Kapitel 4.2 auf Seite 51) entspricht. $\mathcal{S}_{\tau_i}$ bestimmt den Anteil der jeweiligen Task an der verbleibenen End-To-End-Deadline und wird durch das gewählte Verfahren der Deadline-Synthese (also welche Metrik kommt zum Einsatz) bestimmt. Anschließend kann in der zweiten Phase die dann noch verbleibene Restzeit einer End-To-End-Deadline auf alle übrigen Tasks einer Taskkette verteilt werden. Auch hier bestimmt die jeweilige Metrik über den $\mathcal{S}_{\tau_i}$-Faktor den Anteil.

$$\forall \tau_i \notin \mathcal{T}^{share} \wedge \tau_i \in \mathcal{C}_j^k : \Delta_\tau(\tau_i) = \mathcal{S}_{\tau_i} \cdot \left(\lambda - \sigma - \sum_{\tau_s \in \mathcal{T}^{share} \cap \mathcal{C}_j^k} \Delta_\tau(\tau_s) \right) \tag{6.7}$$

Dieses Verfahren kann für Kommunikationen $g_{i+1} \in \gamma_i$ analog durchgeführt werden: Zunächst wird die Menge der Nachrichten einer Taskkette in $\mathcal{G}_j^k$ konstruiert, und anschließend wird mithilfe von Anteilen an der gesamten Deadline $\mathcal{S}_{g_{i+1}}$ das in den Gleichungen 6.6 und 6.7 definierte Verfahren auch auf die Nachrichtenübermittlung angewendet[2], so dass letztlich für jede Nachricht g_{i+1} eine synthetische Deadline $\Delta_\gamma(g_{i+1})$ erzeugt wird.

Anteils-basiertes Verfahren

Das Anteils-basierte Verfahren ist ähnlich der R_{NORM}-Metrik aus [127] und ordnet einer Task bzw. Nachricht einen zum Anteil des Tasks/ der Nachricht am WCET-

[1] Dies kann beispielsweise in sehr engen Taskketten auftreten, in denen dann die Kommunikationen möglichst lokal sein müssen, um den Anforderungen zu genügen.

[2] wiederum unter Berücksichtigung der Menge von Nachrichten $\mathcal{G}^{share}$, die Teil mehrerer Taskketten sind

Pfad proportional großen Anteil an der End-To-End-Deadlines λ zu. Der WCET-Pfad $t_{wc}^{d_j^k}$ ist dabei die Summe aus den WCETs aller Tasks der Taskkette plus den worst-case Übertragungszeiten aller Nachrichten der Taskkette und wird nach folgender Vorschrift gebildet:

$$\forall d_j^k = (\text{“}\tau_0 \ldots \tau_n\text{“}, \lambda) : t_{wc}^{d_j^k} = \sum_{i=0}^{n} \max_{p \in P} \{c_i(p)\} + \sum_{i=0}^{n-1} t_{wc}^{com}(g_{i+1}) \quad \text{mit } g_{i+1} = (\tau_{i+1}, \cdot) \in \gamma_i \tag{6.8}$$

Zwecks Abbildung aller möglichen Platzierungssituationen wird für Tasks das Maximum ihrer WCETs über allen Prozessorknoten eines Systems benutzt. Nachrichten werden entsprechend dem maximalen Pfad in der Topologie des Systems berücksichtigt, indem die schlimmstenfalls auftretende Übertragungszeit (ohne Blockierungen oder Preemptionen durch das Bussystem oder andere Nachrichten) für jedes Kommunikationsmedium auf den möglichen Pfaden bestimmt wird und der sich dadurch ergebende maximale Pfad in $t_{wc}^{com}(g_{i+1})$ aufsummiert ist.
Der Anteil der Ausführungszeit $\mathcal{S}_{\tau_i}$ einer Task τ_i an einem WCET-Pfad $t_{wc}^{d_j^k}$ berechnet sich demnach durch:

$$\mathcal{S}_{\tau_i} = \alpha_{\tau}^{d_j^k} \cdot \frac{\max_{p \in P} \{c_i(p)\}}{t_{wc}^{d_j^k}} \tag{6.9}$$

Analog wird der Anteil einer Nachricht an der insgesamt für die Taskkette maximal verbrauchten Berechnungszeit $\mathcal{S}_{g_{i+1}}$ errechnet:

$$\mathcal{S}_{g_{i+1}} = \alpha_{\kappa}^{d_j^k} \cdot \frac{t_{wc}^{com}(g_{i+1})}{t_{wc}^{d_j^k}} \tag{6.10}$$

Zusätzlich zu diesen Faktoren können die beiden Teil-WCET-Pfade für Kommunikationen und Tasks durch entsprechende Faktoren $\alpha_{\tau}^{d_j^k}$ und $\alpha_{\kappa}^{d_j^k}$ gewichtet werden. Dies dient einer systemabhängigen Voreinstellung, beispielsweise um in Systemen, in denen a priori bekannt ist, dass die Kommunikation ein Engpass ist, Nachrichtenübertragungen einen größeren Anteil an den Deadlines zukommen zu lassen. Die Gewichtungsfaktoren $\alpha_{\tau}^{d_j^k}$ und $\alpha_{\kappa}^{d_j^k}$ sind voneinander abhängig und werden für jede Taskkette aus einer globalen Gewichtungskonstante für Tasks (α_{τ}) errechnet.

Lösungs-getriebene Verfahren

Anteils-basierte Verfahren haben den Nachteil, dass sie zwar bei Kommunikationen im Prinzip die gegebenen Systemtopologie berücksichtigen, dies aber bei der Taskausführung vollkommen unberücksichtigt lassen und damit keine Rücksicht auf wesentliche Optimierungspotentiale nehmen. Dies führte in [80] bereits zur Einführung beispielsweise der ADAPT_L-Metrik, um vorhandene Parallelitätseffekte ausnutzen zu können. In den hier betrachteten Anwendungsdomänen spielen zusätzlich weitere Beschränkungen, wie Speicher, Redundanztasks oder vorgegebene Platzierungsrestriktionen eine überaus wichtige Rolle. Eine vollkommen freie Verteilung der Tasks auf Prozessorknoten ist demnach hier nicht möglich, sondern es wird für jede Probleminstanz eigene Lösungsräume geben. Allgemeingültige Metriken, wie die

Anteils-basierte oder die ADAPT_L-Metrik können diesen spezifischen Eigenschaften nicht genügen. Stattdessen soll im folgenden eine Metrik vorgestellt werden, die problemspezifisch aufgrund einer abschätzenden Vorplatzierung die Deadlines vergibt: die sogenannte "Lösungs-basierte" Metrik.
Zunächst soll das Vorgehen in einfachen Architekturen, in denen Nachrichten nur auf genau einem Kommunikationsmedium übermittelt werden können, betrachtet werden. Eine Erweiterung auf komplexere Topologien wird anschließend in Kapitel 6.2.4 angegeben. Lösungsbasierung bedeutet hier, dass ausgehend von der Beschreibung des Problems zunächst eine abschätzende — im folgenden *relaxiert* genannte — Platzierung berechnet wird. Hierzu wird eine Platzierungsoptimierung durchgeführt, die nicht exakt ist, sondern auf Abschätzungen beruht:

- Platzierungsrestriktionen (π_i) und Redundanztasks (δ_i) werden exakt berücksichtigt
- Speicherverbrauch ($\mu_i(p)$) wird exakt berücksichtigt
- Antwortzeiten von Tasks werden nicht betrachtet, stattdessen wird nur die Prozessorauslastung U_p berechnet, und das Optimierungsverfahren sorgt für die Einhaltung einer festgelegten oberen Grenze ($U_p < U_{thres}$)
- Antwortzeiten von Nachrichten werden nicht betrachtet, stattdessen wird auch hier nur die Auslastung des Bussystems angegeben und unterhalb eines Schwellwertes gehalten

Ein Optimierungsverfahren berechnet bzgl. eines gegebenen Optimierungskriteriums nun eine abschätzende Platzierung, in der aber die exakt beschriebenen Eigenschaften bereits Berücksichtigung finden. Als Allokationsverfahren wurde hier wiederum das SAT-basierte Verfahren verwendet, aus Gründen der Laufzeit[1] kann aber natürlich auch jedes andere heuristische oder stochastische Verfahren eingesetzt werden. Der Aufbau der Gleichungssysteme für eine Optimierung des relaxierten Problems mittels der SAT-basierten Methode erfolgt vollkommen analog zu der in Kapitel 5.3.4 beschriebenen Vorgehensweise und soll an dieser Stelle nicht mehr explizit dargestellt werden; der interessierte Leser sei hier auf die Ausgaben der TATONO-Werkzeuge (siehe Beispiel in Anhang D.2) verwiesen.

Ausgehend von einer relaxierten Lösung des Platzierungsproblems werden anschließend die gefundenen Platzierungen benutzt, um für Tasks bzw. Nachrichten auf Basis der Formalisierung in Kapitel 4.2 Antwortzeiten $\tilde{r}_{\tau_i}$ bzw. $\tilde{r}_{g_{i+1}}$ zu berechnen. Dabei werden zunächst keine Jitter berücksichtigt und als Proritätenzuordnung wird das Raten-monotone Verfahren eingesetzt. Dies liefert letztlich eine Näherung an die zu erwartenden Antwortzeiten im platzierten System, und wir nutzen diese Infor-

[1] Eine Anwendung des SAT-basierten Verfahrens für typische Tasksysteme erzielt sehr schnell ein Ergebnis. So konnte beispielsweise für das in [183] beschriebene System innerhalb von 70 Sekunden eine relaxierte Lösung berechnet werden.

mationen als WCET in einem WCET-Pfad:

$$\forall d_j^k = (\text{“}\tau_0 \ldots \tau_n\text{“}, \lambda) : t_{wc}^{d_j^k} = \sum_{i=0}^{n} \tilde{r}_{\tau_i} + \sum_{i=0}^{n-1} \tilde{r}_{g_{i+1}} \quad \text{mit } g_{i+1} = (\tau_{i+1}, \cdot) \in \gamma_i \tag{6.11}$$

Analog dem Anteils-basierten Vorgehen werden nun auch bei der Lösungs-basierten Metrik die Anteile der Taskausführungen bzw. Nachrichtenübermittlungen genutzt, die End-To-End-Deadline zu verteilen:

$$\mathcal{S}_{\tau_i} = \alpha_\tau^{d_j^k} \cdot \frac{\tilde{r}_{\tau_i}}{t_{wc}^{d_j^k}}, \tag{6.12}$$

d.h. relaxierte Antwortzeiten werden genutzt, einen WCET-Pfad aufzubauen, der in vielen Punkten näherungsweise an der späteren optimalen und exakten Lösung des Platzierungsproblems liegt.
Analog wird der Anteil einer Nachricht an der insgesamt für die Taskkette in der relaxierten Platzierungslösung verbrauchten Berechnungszeit $\mathcal{S}_{g_{i+1}}$ errechnet:

$$\mathcal{S}_{g_{i+1}} = \alpha_\kappa^{d_j^k} \cdot \frac{\tilde{r}_{g_{i+1}}}{t_{wc}^{d_j^k}} \tag{6.13}$$

Insgesamt berücksichtigt diese Metrik viele Optimierungspotentiale und schafft damit ein realistischeres, weil an das eigentliche spezifische Platzierungsproblem angelehntes, Maß für die Ausgestaltung der lokalen Deadlines. Gleichwohl kann unter bestimmten Bedingungen die relaxierte Deadline-Synthese zu restriktiv sein. Beispielsweise weisen die Kommunikationsstrukturen in Zeit-basierten Kommunikationsmedien, wie Tokenring oder TTP die Eigenschaft aus, dass sehr kleine Deadlines aufgrund des TDMA-Verfahrens nicht möglich sind. In diesen Fällen zwingt die einfache relaxierte Lösung das Platzierungsverfahren aufgrund der vorgegebenen Deadlines in eine der relaxierten Lösung fast identische Taskallokation, die aber nicht unbedingt auch wirklich eine valide Platzierung ist. Aus diesem Grund schlagen wir eine weitere Lösungs-basierte Metrik vor, die *relaxierte Metrik ohne Kommunikationsbetrachtung*. Hierbei wird speziell für TDMA-Verfahren die Anwortzeit von Nachrichten uniform behandelt, d.h. es wird von einer maßgeblichen Antwortzeit für alle Nachrichten, egal ob lokal platziert oder nicht, ausgegangen. Im Falle des Tokenring-Protokolls wird beispielsweise die Token Rotation Time als Antwortzeit für alle Nachrichten angenommen. Anschließend wird nur für die Tasks des Systems eine Deadline gebildet, wobei die Nachrichtenübertragungen ausgeblendet werden:

$$\forall d_j^k = (\text{“}\tau_0 \ldots \tau_n\text{“}, \lambda) : t_{wc}^{d_j^k} = \sum_{i=0}^{n} \tilde{r}_{\tau_i} \tag{6.14}$$

Gemäß Gleichung 6.12 wird nun eine lokale Deadline für jede Task einer Taskkette gebildet. Die uniforme Antwortzeit von Nachrichten wird abschließend von allen lokalen Task-Deadlines wieder abgezogen, wobei hier das Maximum über alle Nachrichten-Deadlines betrachtet wird, um allen Taskketten gerecht zu werden:

$$\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset : \Delta_\tau(\tau_i) := \Delta_\tau(\tau_i) - \max_{g_{i_j} \in \gamma_i} \Delta_\gamma(g_{i_j})$$

Dieses Vorgehen macht es sich zu Nutze, dass die Taskdeadlines die Deadlines ihrer ausgehenden Nachrichten aufnehmen können, so diese alle lokal abgewickelt werden (vgl. Kapitel 4.2 auf Seite 50) und stellt damit keine generelle Einschränkung des Platzierungsproblems dar.
Welche der beiden relaxierten Metriken zum Einsatz kommt, hängt maßgeblich von der verwendeten Systemarchitektur ab (also ob beispielsweise massiv Gebrauch von Zeit-basierten Kommunikationsmedien gemacht wird). Eine Bewertung dieser beiden und anderer Metriken ist in Kapitel 6.2.5 angegeben und zeigt, in welchem Szenario die relaxierte Metrik und die relaxierte Metrik ohne Kommunikation eingesetzt werden sollte.

6.2.4 Deadline-Synthese für komplexe Systeme

Sollen, im Gegensatz zum letzten Abschnitt, komplexere Architekturen mit einer hierarchisch aufgebauten Topologie bezüglich der Deadline-Synthese betrachtet werden, so ist zunächst festzustellen, dass hierüber in der Literatur keine weitergehenden Arbeiten existieren; Alle Arbeiten, die sich mit Deadline-Metriken beschäftigen, sind zugeschnitten auf oben beschriebene einfache Architekturen mit nur einem Kommunikationsmedium oder aber betrachten die Kommunikationsmedien nur abstrakt. Dies liegt in erster Linie auch daran, dass üblicherweise (wie oben bereits dargestellt) die Latenzzeiten der Nachrichten nicht mit einer Deadline versehen werden und es daher aus Sicht der Deadline-Synthese uninteressant ist, über welche Art der Topologie eines Systems die Nachricht letztlich verschickt wird. Allerdings ist dem gegenüber zu stellen, dass selbst wenn die Zeiten für das Versenden von Nachrichten in einfachen Systemen[1] sehr klein sind, sich diese Latenzzeiten stark vergrößern, sobald hierarchische Bussysteme verwendet werden. Zeitverluste auf Gateway-Knoten und mehrfache Nachrichtenübertragung sind hierfür verantwortlich und sorgen dafür, dass die Latenzzeit von Nachrichten durchaus in die Größenordnung von Task-Antwortzeiten kommen können, auch wenn womöglich die Anzahl der zu übertragenden Datenbytes gering ist. Insbesondere bei dieser Art von Topologien ist es folglich erforderlich, Deadlines für die Nachrichtenübermittlung zu berücksichtigen.

Die beiden statischen Varianten “Anteilsbasiert” und “ADAPT-L” können zunächst einfach übernommen werden, indem die Zeit einer Nachrichtenverschickung $t_{wc}^{com}(g_j)$ gemäß der Topologie gebildet wird. Dies bedeutet, dass der maximale Pfad einer jeden Pfadhülle des Systems benutzt wird, um die Summe der schlimmsten anzunehmenden Latenzzeiten einer Nachrichtenübertragung auf den Medien in diesem Pfad errechnen zu können. Welche Probleme dies insbesondere bei der ADAPT_L-Metrik nach sich ziehen kann, wird im folgenden Kapitel im Rahmen der Evaluation der Verfahren ersichtlich.

Relaxierte Metriken hingegen können nicht so einfach auch auf komplexe Topologien übertragen werden. Da sie eine abschätzende Platzierung berechnen, müssen sie dies auch für die Benutzung der unterschiedlichen Kommunikationsmedien durch Nachrichten des Tasksystems durchführen. Hintergrund ist hierbei, wie oben darge-

[1] Was, wie wir im letzten Abschnitt gesehen haben, nicht für alle Systeme richtig ist.

legt, eine Lösung des Problems möglichst nahe der optimalen Platzierung oder aber zumindest eine ähnliche Lösung zu errechnen. Konnte dieses Ziel in einfachen Systemen durch die Abstraktion der Kommunikationen in eine Auslastungsbetrachtung erreicht werden, so genügt dies hier nicht mehr: Die Auslastung auf den einzelnen Bussystemen gibt zwar einen guten Hinweis auf zu vermeidende Überlasten, es sagt aber in hierarchisch organisierten Topologien, anders als in einfachen Systemen mit nur einem Bus, nichts über die mögliche Einhaltung der End-To-End-Deadlines aus. Nachrichten, die über einen Pfad mit mehreren Medien verschickt werden, mögen die Busauslastung insgesamt gering halten, führen aber unter Umständen zu so hohen Latenzzeiten, dass eine gegebene End-To-End-Deadline nicht mehr eingehalten werden kann. D.h. eine Formulierung des relaxierten Problems, wie sie in Kapitel 75 gegeben ist, nimmt in der Optimierung keine oder nur wenig Rücksicht auf diesen inhärent wichtigen Faktor; Er ist durch eine reine Auslastungsbetrachtung auf Kommunikationsmedien nicht präsent. Stattdessen muss in das Verfahren eine abschätzende Antwortzeitanalyse integriert werden, die in der Lage ist, zu weitreichende Verletzungen der End-To-End-Deadlines zu vermeiden. Dies sollte wiederum nur eine Abschätzung sein, da zum einen die Komplexität ganzer eingebetteter Echtzeitsysteme, wie sie in der hier betrachteten Anwendungsdomäne vorliegen, in einer Größenordnung liegt, die mit Optimierungsmethoden in der Regel nicht mehr bearbeitet werden können. Zum anderen, und sehr viel wichtiger, findet die relaxierte Optimierung zu einer Phase im Entwurfsprozess statt, in der größtenteils mit unscharfen oder geschätzten Eigenschaften der Tasks und Nachrichten, aber auch der Architektur, gearbeitet werden muss. Basis für eine Analyse der Antwortzeiten von sowohl Tasks als auch Nachrichten sollte folglich eine möglichst einfach zu berechnende und mit kalkulierbaren Fehler ausgestattete Lösung der Fixpunktgleichung aus z.B. Gleichung 4.3 sein. Eine gute Lösung des Problems ist die Anwendung des Banachschen Fixpunktsatzes auf die Antwortzeitberechnung, wie sie in [171] vorgenommen wurde: Der Banachsche Fixpunktsatz besagt, dass es für eine stetige kontrahierende Funktion $f : A \to A$, die auf einer abgeschlossenen Teilmenge $A \subset \mathbb{R}$ definiert ist, genau ein $x \in A$ gibt, so dass $f(x) = x$ gilt. Eine Funktion ist kontrahierend, wenn es ein $L < 1$ gibt, so dass gilt

$$|f(x) - f(y)| \leq L|x - y| \text{ für alle } x, y \in A$$

Die Anwendung dieses Satzes auf die Antwortzeitberechnung für eine Task τ_i führt beispielsweise zu der folgenden kontinuierlichen Funktion:

$$r_{\tau_i} = f(r_{\tau_i}) = c_i + r_{\tau_i} \cdot \sum_{\tau_j : \phi_j > \phi_i} \frac{c_j}{t_j} + \Xi,$$

wobei $0 \leq \Xi < \sum_{\tau_j : \phi_j > \phi_i} c_j$ den Übergang zu einer kontinuierlichen Funktion ermöglicht und den maximalen Fehler darstellt. Löst man diese stetige, kontinuierliche Funktion, so ergibt sich die folgende abschätzende Antwortzeitberechnung

$$r_{\tau_i} = \frac{c_i}{1 - \sum_{\tau_j : \phi_j > \phi_i} \frac{c_j}{t_j}},$$

ausgestattet mit einem Fehler von Ξ. Diese Form der Antwortzeitberechnung $f(r_{\tau_i})$ besitzt eine Ableitung mit $f'(x) < 1$, wenn die Auslastung auf dem betrachteten Prozessorknoten kleiner 100% ist, d.h. der Banachsche Fixpunktsatz ist anwendbar. Der Fehler bedeutet zunächst, dass die mit dieser Methode berechnete Antwortzeit um den Betrag des Fehlers zu klein ist. Soll hingegen eine sichere Abschätzung nach oben erreicht werden, so lässt sich dies durch einen entsprechenden additiven Faktor bewerkstelligen (beispielsweise kann der maximale Wert von Ξ in den Zähler obiger Gleichung additiv aufgenommen werden).
Auf die gleiche Art und Weise lässt sich auch die Antwortzeitberechnung von Nachrichten auf unterschiedlichen Bussystemen in eine stetige und kontinuierliche Funktion transformieren und ermöglicht es, effizient relaxierte Lösungen zu berechnen. Hierzu müssen freilich die Informationen aus den Pfadhüllen mitgenutzt werden, so dass das Routing der Nachrichten auf der Topologie der Systemarchitektur nun zusätzlich exakt erfolgen muss. Prioritätenbeziehungen erfolgen wiederum nach der Ratenmonotenen Strategie und sind somit a priori bekannt.

6.2.5 Deadline-Synthese — Vergleichende Evaluation

Eine Bewertung der jeweiligen Metrik zur Generierung von lokalen Echtzeiteigenschaften für Tasks und Nachrichten aus einer gegebenen End-to-End-Deadline werden im folgenden anhand von Messungen an zwei beispielhaften Tasksystemen vorgenommen. Beide Tasksysteme sollen auf einer Architektur, die aus 8 identischen, aber mit unterschiedlicher Speichergröße ausgestatteten, Prozessorknoten besteht, platziert werden. Die Prozessorknoten sind alle an einen gemeinsamen Bus angeschlossen. Zunächst wird mittels der jeweiligen Deadline-Metrik für jede Task und jede Nachricht eine lokale Deadline für eine festgelegte Übertragungsrate des gemeinsamen Busses bestimmt. Gewichtungsfaktoren, wie α_τ sind hier mit 1.0 festgelegt worden, die k_S-Konstante für das ADAPT_L-Verfahren wurde mit dem in [80] vorgeschlagenen Wert von 0.2 belegt. Anschließend wird das Optimierungsverfahren auf ein leicht relativiertes Problem angewendet, in dem die Übertragungsrate des Busses inkrementiert wird. Auf diese Art und Weise kann die Leistungsfähigkeit der jeweiligen Metrik abgelesen werden: Je höher die Übertragungsrate, desto entfernter ist die Architektur von der für die Deadline-Synthese vorrausgesetzten Architektur. Metriken, die erst bei höheren Übertragungsraten eine valide Platzierung errechnen können, sind folglich als schlechter geeignet zu klassifizieren.

Der erste Versuch ist [183] entnommen (siehe Beschreibung in Kapitel 5.6.1 auf Seite 171 und folgende) und besitzt als zentrales Kommunikationsmedium einen Tokenring mit einer Übertragungsrate von 90 bytes/ms. Kommunizierende Tasks sind lose gekoppelt, d.h. alle End-to-End-Deadlines in dem System sind gerade so definiert, dass sie die Ausführung einer Task inklusive der Übertragungszeit aller von dieser Task ausgehenden Nachrichten umfasst. Die Ergebnisse dieser Messung sind in Tabelle 6.2 aufgelistet, wobei *relax* der relaxierten Metrik, *relax_wc* der relaxierten Metrik ohne Kommunikationsbetrachtung, *ADAPT_L* der Metrik, wie sie in [80] vorgeschlagen wird, und *share* der anteilsbasierten Metrik entspricht. *relax** resultiert aus den Schlussfolgerungen für das Ergebnis von *relax* und wird weiter unten eingeführt.

	Übertragungsgeschwindigkeit [bytes/ms]						
Metrik	90	110	130	150	170	190	250
relax_wc	✓	✓	✓	✓	✓	✓	✓
relax	✗	✓	✓	✓	✓	✓	✓
relax*	✓	✓	✓	✓	✓	✓	✓
ADAPT_L	✗	✗	✗	✗	✓(68%)	✓(66%)	✓(68%)
share	✗	✗	✗	✗	✗	✗	✗

Tabelle 6.2: Einfluss der jeweiligen Deadline-Synthese Metrik auf die Platzierungseigenschaft an einem Beispiel aus [183]. ✗ zeigt an, dass keine Platzierung gefunden werden konnte, ✓ zeigt hingegen, dass eine gültige Lösung des Platzierungsproblems gefunden wurde. Die %-Angabe bedeutet die Güte relativ zur Lösung, die mit dem Verfahren *relax_wc* erreicht wurde. Fehlende %-Angaben bedeuten 100%.

Das betrachtete System ist in allen Belangen (Prozessorauslastung, Speicherverbrauch und Busauslastung) ein sehr enges System, dass nicht viele valide Lösungen zulässt. Dementsprechend führen schlechtere Deadline-Metriken auch sehr leicht zu einem nicht-lösbaren Problem für die Platzierung. Dies ist in Tabelle 6.2 gut an der anteilsbasierten und an der ADAPT_L-Metrik zu ersehen. Insbesondere die anteilsbasierte Lösung findet bei keiner Übertragungsgeschwindigkeit eine valide Lösung. Zwar haben wir in [116] gezeigt, dass es durchaus auch valide Lösungen für eine anteilsbasierte Deadline-Synthese des hier betrachteten Systems geben kann, allerdings bedingt dies — gerade in derart engen Tasksystemen, wie es das hier betrachtete ist — eine sehr feine Justierung der Gewichtungskonstante α_τ. Auffällig ist auch, dass die in [80] vorgeschlagenen und dort durchgehend mit gut befundene ADAPT_L-Metrik erst bei nahezu einer Verdopplung der Übertragungsrate auf dem Bus zu gültigen Lösungen findet, die dann aber zusätzlich noch deutlich schlechter ausfallen, als die Ergebnisse der relaxierten Metriken[1].
Die Gründe für eine derart auffällige Abweichung von den Beobachtungen in [80] sind in erster Linie in der notwendigen Einbeziehung der Kommunikationen in die Metrik zu suchen. Erschwerend kommt hinzu, dass der hier betrachtete Tokenring-Bus ein sehr differenziertes Verhalten im Vergleich zum Taskscheduling zeigt: Antwortzeiten von Nachrichten sind für alle Nachrichten im schlimmsten Fall gleich und entsprechen der TRT (vgl. die Abhandlungen in Kapitel 27 auf Seite 80 und folgende). Die ADAPT_L-Metrik ist aber in ihrer grundlegenden Struktur nicht für derartige Systeme geschaffen worden. Dies gilt, wenn auch eingeschränkt, gleichermaßen auch für andere Zeit-basierte Bussysteme, wie TTP oder FlexRay. Zwar ist die Antwortzeit-Analyse von Nachrichten in diesen Systemen dem Verhalten von Tasks nicht unähnlich, aber je weniger Pakete eine Nachricht umfasst, desto eher gleicht das Verhalten dem eines Tokenrings (siehe beispielsweise Gleichung 4.12 auf Seite 70).

[1] Betrachtet wurde hier eine Minimierung der Token Rotation Time; Eine Güte von beispielsweise 68% bedeutet dann, dass die unter der ADAPT_L-Metrik gefundene Lösung für die TRT um 32% über der vom besten Verfahren gefundenen minimalen Lösung liegt.

Beide Varianten der relaxierten Metrik (*relax* und *relax_ wc* in Tabelle 6.2) weisen gute Eigenschaften auf und liefern entsprechende Lösungen. Wir konnten nachweisen, dass die Lösung für die *relax_ wc*-Metrik sogar die optimale Platzierung des Systems darstellt[1]. Einzig für das Originalsystem führte auch die relaxierte Metrik zu keiner validen Platzierung. Eine Betrachtung des relaxierten Verfahrens (vgl. Kapitel 75 auf Seite 212) zeigt schnell den Grund für dieses Verhalten: Nachrichten, denen in der relaxierten Platzierung keine Übertragungszeit zugeordnet wurde, weil sowohl Sender- als auch Empfängertask auf dem selben Prozessorknoten platziert wurden, werden mit sehr kleinen Deadlines ausgestattet. Dies führt im betrachteten Testsystem dazu, dass wegen der relativ vielen Nachrichten, die aufgrund von Platzierungsrestriktionen über den Bus kommuniziert werden müssen, jene im relaxierten Problem lokale Nachrichten auch im Testsystem lokal sein müssen. Es werden folglich implizit neue Restriktionen aufgebaut, die einer Lösung des Platzierungsproblems im Wege stehen können. Wesentlicher Grund für dieses Verhalten ist im Zeitverhalten des Tokenring (aber, wie oben ausgeführt, auch anderer Zeit-basierter Kommunikationsmedien) zu suchen, da dort kurze Deadlines auf dem Bus bei vielen Nachrichten mit hoher Last grundsätzlich nicht möglich sind.
Um dieses Verhalten abzuschwächen, führen wir eine adaptierte Variante der relaxierten Metrik *relax** ein: Basierend auf dem Mittelwert aller errechneten Deadlines von Nachrichten wird jeweils eine linke und rechte Grenze für eine mögliche Deadlineverteilung angegeben. Prozentual (mit einer geeigneten Konstante, im Beispiel in Tabelle 6.2 ist dies 40%) von der mittleren Deadline wird ein Intervall von Nachrichten-Deadlines festgelegt. Fällt die Deadline einer Nachricht außerhalb dieses Intervalls an, so wird sie auf die entsprechend näher liegende Intervall-Grenze verschoben und anschließend die Deadline des sendenden Tasks entsprechend angepasst. Bei Tasks, die Teil mehrerer Taskketten sind, wird jeweils nur die minimale Anpassung über alle verschobenen Deadlines von Nachrichten dieser Task berücksichtigt.
Tabelle 6.2 zeigt, dass das erwartete Ergebnis dieser Anpassung an ein Zeit-basiertes Kommunikationsmedium in der Tat zu beobachten ist und die *relax**-Metrik auch auf dem Originalsystem eine valide Platzierung ermöglicht, die zudem der optimalen Platzierung der durch die *relax_ wc*-Metrik erfolgten Platzierung entspricht.

Als weiterer Test soll hier ein Tasksystem mit End-to-End-Deadlines und Taskketten mit einer Länge größer als 1 betrachtet werden. Die maximale Tiefe der Taskketten beträgt 5 Tasks und der maximale Verzweigungsgrad ist mit 3 angegeben (vergleiche auch die Beschreibung des Systems in Anhang E.1). Als Architektur kommt die gleiche Architektur wie im obigen Beispiel zum Einsatz, allerdings musste bei 2 Knoten die Menge an Speicher ein wenig angehoben werden, um für dieses System überhaupt eine valide Platzierung erzeugen zu können. Dementsprechend ist der Speicher auch der restringierende Faktor bei der Platzierung. Auf dem Bus hingegen müssen aufgrund der Platzierungsvorgaben nur wenige Nachrichten übermittelt

[1]Der Nachweis der Optimalität ist durch ein modifiziertes Platzierungsverfahren erbracht worden, in dem nicht die lokalen Deadlines betrachtet werden, sondern eine vollständig holistische Analyse gemäß [181] durchgeführt wird. Auch dieses ist problemlos mit dem hier dargestellten, auf SAT-Checking basierenden, Verfahren durchführbar.

werden.
Auch dieses System ist mit allen Metrik-Varianten evaluiert worden und die Ergebnisse sind in Tabelle 6.3 festgehalten.

	Übertragungsgeschwindigkeit [bytes/ms]						
Metrik	90	110	130	150	170	190	250
relax_wc	✓	✓	✓	✓	✓	✓	✓
relax	✓	✓	✓	✓	✓	✓	✓
relax*	✓	✓	✓	✓	✓	✓	✓
ADAPT_L	✗	✗	✗	✗	✗	✗	✗
share	✓	✓	✓	✓	✓	✓	✓

Tabelle 6.3: Einfluss der jeweiligen Deadline-Synthese Metrik auf die Platzierungseigenschaft an einem Beispiel mit End-to-End-Deadlines (siehe Beschreibung in Anhang E.1 auf Seite 331 und folgende).

Die Ergebnisse zeigen die Anwendbarkeit nahezu aller Metriken auf Tasksystemen mit längeren Taskketten. In diesem Beispiel ist auch die anteilsbasierte Metrik in der Lage, valide Platzierungen zu erlauben; das mögliche Risiko, Deadlines zu klein oder zu groß zu bestimmen, ist hier verteilt auf die ganze Länge der Taskkette, und einzelne kritische Ausreißer können daher gut innerhalb der Taskkette wieder aufgefangen werden. Dies ist eine den Systemen mit längeren Taskketten inhärent eigene "Gutmütigkeit" gegenüber Deadline-Syntheseverfahren.
Ein wenig überraschend mutet vielleicht das vollständige Versagen der ADPAT_L-Metrik an, die in keiner der evaluierten Architekturen eine valide Platzierung ermöglichen konnte. Bei genauerer Betrachtung ist jedoch der Grund einfach einzusehen: In diesem Tasksystem gibt es Taskketten, deren End-to-End-Deadlines nur dadurch eingehalten werden können, dass nahezu alle Nachrichten lokal erfolgen. Dies führt soweit, dass bei der Summierung aller WCETs dieser Taskketten inklusive der WCET für die Nachrichtenübertragung ein Wert jenseits der End-to-End-Deadline entsteht. Dies wiederum führt bei der Berechnung der ADAPT_L-Metrik im R_{PURE}-Faktor (vgl. Gleichung 6.1 auf Seite 209) zu negativen Werten und drängt die Deadline sowohl von Tasks als auch von Nachrichten in Bereiche kleiner als die jeweilige WCET. Für Nachrichten kann dieses Verhalten aufgefangen werden, indem diese alle lokal behandelt werden, für Tasks führt dies jedoch zu einem Schedule, der a priori infeasible ist.
Werden hierarchisch organisierte Topologien mit mehreren Kommunikationsmedien betrachtet, verstärkt sich dieses Problem drastisch, da die schlimmste Latenzzeit einer Nachricht sich aus dem Transfer über mehr als ein Bussystem zusammensetzt. Zwar kann der Overhead durch die Parallelität der Medien wieder etwas reduziert werden, aber insgesamt zeigte sich, dass die ADAPT_L-Methode in der hier beschriebenen Art und Weise in hierarchischen Systemen keine adequate Metrik für die Deadline-Synthese darstellt[1]. So werden für beide oben beschriebenen Beispiele

[1]Bei Betrachtung von Gleichung 6.1 wird klar, dass bei erhöhten Werten für den WCET-Teil R_{PURE} schneller negativ wird, der reduzierend wirkende Anteil der Parallelität $|\mathcal{C}_j^k|$ aber keinen

in den hierarchischen Architekturen, wie sie in Abbildung 5.6 in Kapitel 5.6.2 auf Seite 176 untersucht werden, Deadlines für Tasks erzeugt, die kleiner als die WCET der jeweiligen Tasks und damit a priori infeasible sind. Stattdessen sind hier die relaxierten Metriken (oder im Einzelfall — bei Verwendung von Ereignis-basierten Bussystemen und/oder viel Spielraum für die Platzierung — die anteilsbasierte Metrik) vorzuziehen (vgl. auch die Messungen in Kapitel 5.6.2, die auf der *relax_wc*-Metrik basieren).

Kommt anstatt eines Zeit-basierten Bussystems, wie dem Tokenring, ein Ereignis-basierter Bus, beispielsweise der CAN-Bus, zum Einsatz, kann die ADAPT_L-Methode partiell verwendet werden. Für das erste Beispiel aus [183] ergibt sich für eine auf ADAPT_L-basierende Deadline-Synthese auch für das Originalsystem mit CAN-Bus eine Lösung, die allerdings etwa um 15% schlechter ist, als die Platzierung unter relaxierten Deadline-Synthese-Metriken[1]. Die der ADAPT_L-Metrik zugrundeliegende Struktur macht dies möglich, da sich Nachrichten auf Ereignis-basierten Medien ähnlich Tasks im preemptiven Scheduling verhalten. Die trotzdem noch vorhandenen Unterschiede zu relaxierten Metriken dürften unter anderem auch in der Nicht-Ausnutzung von Informationen über Platzierungsrestriktionen liegen. Bei der Anwendung auf das zweite Beispiel ergibt sich aber wie oben das Problem, dass für manche Tasks eine Deadline kleiner als ihre WCET erzeugt wird, so dass für dieses Beispiel die ADAPT_L-Metrik wiederum fehlschlägt.
Die anteilsbasierte Metrik *share* kann für beide Beispiele auf einer mit CAN statt Tokenring ausgestatteter Architektur angewendet werden und liefert valide Platzierungen mit einer Güte von 100% verglichen mit dem von den relaxierten Metriken erbrachten Ergebnis. Auch hier dürfte dies wiederum an dem Prinzip der Nachrichtenübermittlung liegen, welches aufgrund der Ereignis-Basierung und partiellen Preemptivität durchaus unterschiedliche Deadlines für Nachrichten, wie sie bei der *share*-Metrik erzeugt werden, schedulen kann.

Insgesamt konnte damit gezeigt werden, dass es gute Metriken gibt, die für einen inkrementellen, auf Komponenten basierenden, Entwurfsprozess eine adequate Deadline-Synthese zur Verfügung stellen und mitunter sogar die Optimalität der letztlich folgenden Platzierung, wie am Beispiel aus [183] demonstriert, erhalten können. Folglich steht der Anwendung der inkrementellen Platzierungstechnik nichts mehr im Wege und wir wollen im folgenden kurz zeigen, wie dies in die SAT-basierte Optimierung integriert werden kann. Wie sich beispielsweise relaxierte Metriken in den Gesamtzusammenhang eines Entwicklungsprozesses für komplexe eingebettete Echtzeitsysteme einordnen, wird in Kapitel 6.5 eingehender diskutiert werden, wenn wir uns mit der virtuellen Integration im industriellen Entwurfsprozess beschäftigen.

Einfluss auf das Vorzeichen hat.

[1] Optimierungskriterium bei diesen Versuchen war die Minimierung der Busauslastung U_{CAN} des CAN-Busses

6.2.6 Integration des inkrementellen Entwurfes in die Platzierungsoptimierung mittels SAT-Techniken

Vor jedem Schritt in der inkrementellen Integration eines Gesamtsystems ist die bis dahin gültige Systemkonfiguration als Startpunkt zu restaurieren, d.h. die gefundene Konfiguration (im wesentlichen bestehend aus Platzierung der Tasks und Routing des Pfades für Nachrichten) muss in einem entsprechenden Design-Repository gesichert werden. Wie in Algorithmus 5 angegeben, kann dies durch die Restriktion bereits platzierter Tasks erreicht werden. Damit werden auch in jedem inkrementellen Platzierungsschritt alle bis dahin bereits platzierten Tasks und Nachrichten in die Berechnung ebenfalls mit einbezogen. Zwar garantiert die Deadline-Synthese und Jitter-Überapproximation, dass keine wechselseitigen Abhängigkeiten mehr bestehen (vgl. Kapitel 4.4.1), allerdings kann natürlich durchaus die Antwortzeit von bereits platzierten Tasks verändert werden, wenn beispielsweise höher priorisierte Tasks auf demselben Knoten allokiert werden[1]. Durch die auf einen Knoten beschränkte Platzierung kann zusätzlich ein exponentielles Wachstum der Laufzeit des SAT-basierten Verfahrens verhindert werden. Die Messungen in Kapitel 6.3 zeigen dies.
In Tabelle 6.4 ist aufgelistet, welche Anteile der Platzierungsmodellierung zwingend im Repository gespeichert werden müssen, um eine äquivalente Platzierung im nächsten inkrementellen Schritt zu erreichen.

Variablen
$\texttt{place}_i$
$\texttt{K}^k_{g_t}$
$\texttt{locD}^k_{g_t}$
$\texttt{s}^k_i$

Tabelle 6.4: Variablen der Platzierungsmodellierung, deren Belegung im Design-Repository gespeichert werden müssen.

Neben diesen zwingend notwendigen Variablen-Belegungen können weitere Belegungen oder einschränkende Bedingungen übernommen werden, die dazu dienen können, die Laufzeit des Optimierungsverfahrens zu minimieren. Dazu gehören beispielsweise die $\texttt{I}^j_i$-Variablen, die die Anzahl der Unterbrechungen durch höher priorisierte Tasks berechnen. Da durch neu hinzukommende Tasks auf dem jeweiligen Prozessorknoten sich der Wert für $\texttt{I}^j_i$ maximal erhöhen, aber niemals erniedrigen kann, kann im Repository eine zusätzliche Bedingung kodiert werden, die ausdrückt, dass der Wert von $\texttt{I}^j_i$ nicht kleiner als eben jene Belegung der gefundenen Platzierung eines vorherigen Laufes sein kann. Auf diese Art und Weise können für das SAT-Verfahren zusätzliche, den Suchraum einschränkende, Klauseln eingeführt werden. Weitere Beispiele sind die Belegungen für $\texttt{p}^i_j$, $\texttt{r}_i$, etc. Ein Überblick über die Realisierung des

[1]Steigende Antwortzeiten bereits platzierter Tasks erhalten trotzdem den Komponenten-Charakter, da die festgelegten Schnittstellen nur über die Deadlines definiert sind. Die Optimierungsmethode trägt auf der anderen Seite dafür Sorge, dass diese Deadlines nicht überschritten werden.

inkrementellen Platzierungsverfahrens in Form einer Software-Architektur inklusive Design-Repository ist in Anhang D gegeben und soll hier nicht weiter betrachtet werden.

6.2.7 Ausblick auf weiterführende Techniken

Mögliche Erweiterungen der Deadline-Synthese wären beispielsweise im Bereich der Metriken für ADAPT_L-ähnliche Verfahren interessant. Hier könnten insbesondere die in der relaxierten Lösung ausschlaggebenden Duplikate und Platzierungsrestriktionen benutzt werden, um den Parallelitätsgrad einer Task zu verfeinern. Evaluationen müssten klären, ob diese Erweiterungen tatsächlich ähnlich gute Ergebnisse erzielen können, wie die relaxierten Methoden. Auch ließe sich an dieser Stelle sicherlich im Bereich der Abschätzung der Latenzzeiten für Kommunikationen eine Adaptierung an die prinzipiell unterschiedlichen Verhaltensweisen Zeit- und Ereignis-basierter Bussysteme vornehmen. Zwar liefern die relaxierten Techniken sehr gute Ergebnisse für die Generierung lokaler Dadlines, jedoch sind sie darauf angewiesen, dass (zumindest in groben Zügen) die vollständige Systemarchitektur bekannt ist. Dies muss aber nicht immer der Fall sein, so dass in unvollständigen Systemen eventuell eine etwas allgemeinere, nicht so sehr auf eine konkrete Architektur bezogene, Metrik erstellt werden kann. Im besonderen Hinblick auf den Re-Use von Komponenten inklusiver ihrerer Echtzeitschnittstellen in vollkommen neuen Systemen könnte dies ein wichtiger Aspekt sein.
Ebenfalls in Richtung Re-Use könnten Arbeiten gehen, die nicht die aktuelle Platzierung konkreter Tasks und Nachrichten im Design-Repository zur Verfügung stellen, sondern eine Art von "Restkapazität" vorhalten. Restkapazitäten könnten die jeweilige Slackness, d.h. den Freiraum, den weitere Platzierungen noch haben, in Form mathematischer Funktionen ausdrücken, die in die Antwortzeitanalysen integriert werden könnten. Der Vorteil dieser Restkapazitäten ist im Kontext des IP-Schutzes zu sehen. So wäre man in der Lage, komplette Subsysteme inklusive Architekturanteilen als Komponenten für einen Re-Use im Komponenten-basierten Entwurfsprozess zur Verfügung zu stellen, ohne dass irgendwelche Informationen über das interne Verhalten der IP-Komponenten als solches erkennbar wären; lediglich das Echtzeitverhalten als Ganzes wäre eine sichtbare Schnittstelle dieser Komponente.
Weiterführende Techniken im Bereich der Datenhaltung für Systemkonfigurationen innerhalb des Design-Repositories sind bereits in Kapitel 6.2.6 beschrieben worden und dienen der Reduktion der Berechnungszeit inkrementeller Integrationsschritte.

6.3 Messungen für die inkrementelle Integration

Die inkrementelle Integration von Komponenten zu einem Gesamtsystem, wie sie in den vorangegangenen Kapiteln beschrieben wurde, basiert auf der Beobachtung von relativer Lokalität bzgl. der Freiheitsgrade der Platzierung einzelner Komponenten. Nur wenn diese, in der Automobilindustrie ebenso wie in der Flugzeugindustrie oder im Bahnbereich weit verbreitete, Eigenschaft gegeben ist, sind die zu erwartenden Verluste durch eine inkrementelle Vorgehensweise ohne die Möglichkeiten eines auto-

matisierten Backtrackings hinreichend klein. In diesen Fällen liegt, wie weiter oben schon dargestellt, das Einsparpotential während der Entwurfsphase deutlich über den Verlusten durch womöglich nicht optimal ausgenutzte Hardware-Resourcen. Es ist zu erwarten, dass diese automatisierten Platzierungsverfahren bei der Größe der betrachteten Systeme der manuellen Platzierung auch von Experten deutlich überlegen ist. Die Laufzeiten der Platzierung, üblicherweise im Bereich weniger Stunden, machen es zudem möglich, diese Technik bereits in frühen Phasen zwecks virtueller Integration und abschätzender Maßnahmen für eine Vorstrukturierung des Designs zu nutzen (vgl. auch Kapitel 6.5, wo dieses Thema detaillierter diskutiert wird).
Für ereignis-gesteuerte Systeme, beispielsweise die im Rahmen dieser Arbeit betrachteten preemptiven, prioritäten-basierten Schedules aber auch ereignis-gesteuerte Bussysteme wie etwa der CAN-Bus, ist die inkrementelle Integration aufgrund des relativ geringeren Einflusses von neu zu platzierenden Teilen auf bereits platzierte Komponenten problemlos möglich. Wir werden dies hier nicht explizit zeigen, implizit sieht man dies jedoch an der Platzierung der Tasks in den folgenden Beispielen, die eben genau diesem ereignis-basierten Schema genügen. Interessanter aus Sicht der inkrementellen Integration ist die Verwendung von Zeit-basierten Busprotokollen: Aufgrund der Theorie der Antwortzeitanalysen zeigt sich, dass kleine Veränderungen am Zeitverhalten des Zeit-basierten Bussystems *alle* Nachrichten betreffen. Demzufolge könnte man vermuten, dass eben diese Systeme große Probleme mit einer inkrementellen Integration haben. Daher werden im folgenden Abschnitt zwei Beispielssysteme evaluiert, die über einen Tokenring-Bussystem verfügen. Der Tokenring wurde als gutes Mittelmaß zwischen einem bzgl. der Framelänge relativ freiem Protokoll wie TTP und dem — aufgrund der uniformen Frames- und Slot-Längen — sehr restriktiven statischen Anteil des FlexRay-Protokolls gewählt[1].

6.3.1 Beschreibung der Fallstudien

Die Anwendbarkeit des Verfahrens soll hier nun anhand zweier Beispiele demonstriert werden, die die Eigenschaft der relativen Lokalität bzgl. der Platzierungsfreiheitsgrade aufweisen und typischen Designs der im Kontext dieser Arbeit anvisierten Anwendugsdomänen entsprechen. Als erste Probleminstanz soll das im Evaluationskapitel 5.6.1 und 6.2.5 schon mehrfach verwendete und in Anhang E.1 dargestellte System dienen. Das zweite Beispiel ist ein komplexerer Entwurf, der aus vier inkrementell zu integrierenden Komponenten besteht und zudem eine Mischung aus synchroner und asynchroner Kopplung von Tasks durch Nachrichten enthält. Abbildung 6.4 gibt einen Überblick über die Toplevel-Komponente dieser Fallstudie.

Sie besteht aus 3 Komponenten *A1, A2* und *B*, die von einer zentralen Steuerungskomponente *Ctrl* überwacht und beeinflusst werden. *A1* und *A2* sind dabei identisch und könnten beispielsweise die Fahrdynamik zweier Vorderräder eines Autmobils steuern. Komponente *B* ist einfach vorhanden und könnte z.B. eine Motorsteuerung darstellen. Die Steuerungskomponente *Ctrl* nimmt Ausgangsdaten aller Komponenten entgegen und kann auf der Basis dieser und ggf. weiterer Informationen die ein-

[1] Zur Erinnerung: Die hier betrachtete Tokenring-Variante verlangt das Versenden aller Nachrichten eines Knotens in einem Slot, ohne die Aufteilung auf mehrfache TDMA-Rounds.

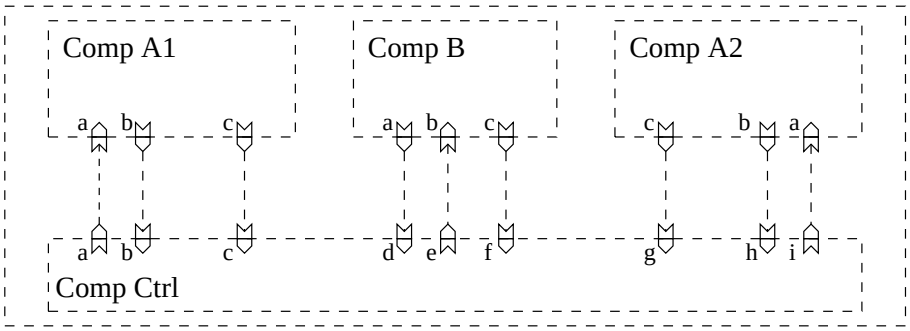


Abbildung 6.4: Toplevel-Komponente der Fallstudie. Gestrichelte Verbindungen bedeuten eine asynchrone Nachrichtenübermittlung, die die Empfänger-Task nicht triggert, sondern gepuffert wird.

zelnen Komponenten steuern. Die Kopplung der Komponenten mit der Steuerungskomponente ist hierbei durch asynchrone Nachrichten (in Abbildung 6.4 dargestellt durch gestrichelte Verbindungen der Ports) realisiert und bedeutet, dass in beide Richtungen Nachrichten keine direkte Triggerung der Empfänger-Komponenten durchführen, sondern die Nachricht bis zum nächsten zyklischen Dispatchvorgang zwischengepuffert und zum Dispatch-Zeitpunkt übergeben wird (vgl. auch das Beispiel aus [183]).
Die interne Struktur der identischen Komponenten *A1* und *A2* ist in Abbildung 6.5 bzw. 6.6 dargestellt.

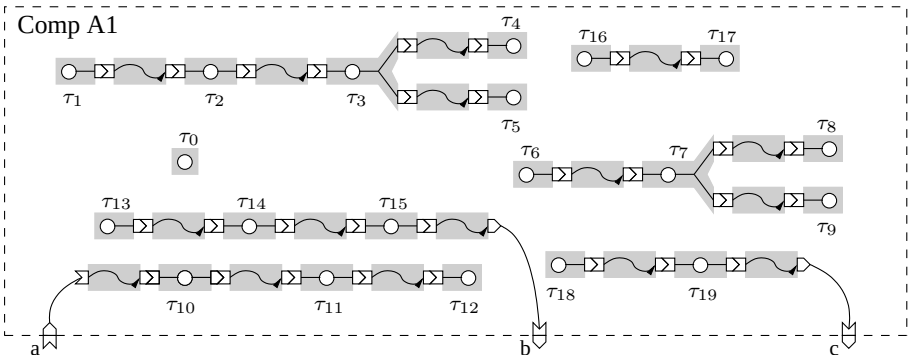


Abbildung 6.5: Komponente A1 mit mehreren Taskketten, über die jeweils eine End-to-End-Deadline gelegt wurde.

Beide Komponenten bestehen aus einer Reihe von Taskketten, die jeweils unabhängig voneinander sind. Drei der Taskketten liefern Daten an die Steuerungseinheit bzw. empfangen Daten von dieser. Alle Nachrichten zwischen Tasks innerhalb der jeweiligen Taskketten sind synchron, d.h. sie triggern die dazugehörige Empfangstask direkt an. Einige der Starttasks sind Zeit-getrieben, andere werden durch Wertänderungen an dedizierten Sensoren angesteuert. Die genaue Zuordnung wird weiter unten in der Beschreibung des Steuergerätenetzwerkes eingehend erläutert.
Komponente *B* besteht nur aus 10 Tasks, die ebenfalls in mehreren Taskketten organisiert sind und bestimmte Sensoren und Aktuatoren nutzen und ist in Abbildung

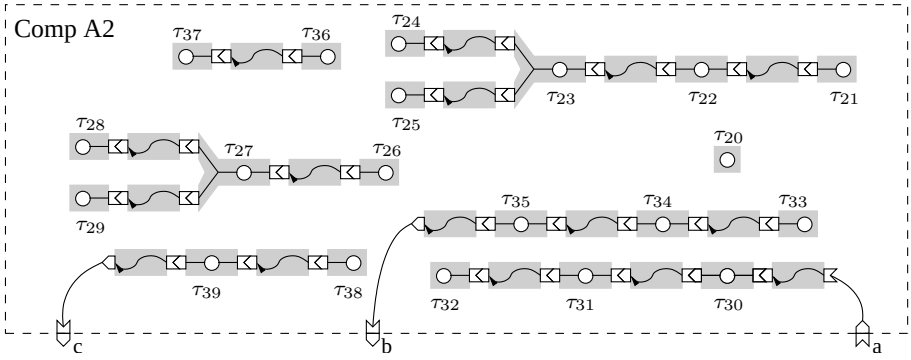


Abbildung 6.6: Komponente A2 entspricht Komponente A1.

6.7 dargestellt.

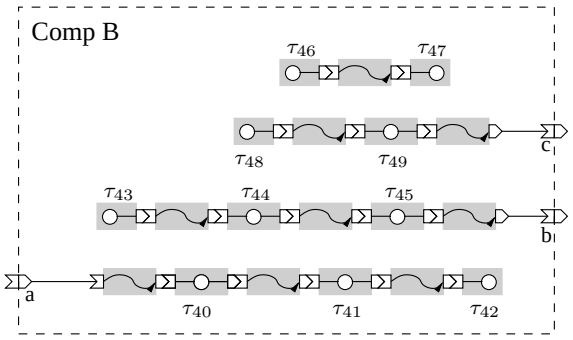


Abbildung 6.7: Komponente B

Die Steuereinheit *Ctrl*— dargestellt in Abbildung 6.8 — besteht aus 10 Tasks, von denen 9 Tasks in 6 Taskketten organisiert sind, die im Zusammenspiel jeweils 2 eine Komponente steuern. Der interne Nachrichtenaustausch erfolgt synchron, anders als der Nachrichtenaustausch mit den zu steuernden Komponenten. Nachrichten von und zu der Steuerungskomponente sind im vorliegenden Entwurf jeweils den Komponenten zugeordnet und tauchen in *Ctrl* nicht auf. Stattdessen haben die Empfänger- bzw. Sender-Tasks direkte Verbindungen zu entsprechend getypten Ports der Komponente. Dies ist an dieser Stelle eine Designfrage und könnte durchaus auch anders realisiert werden, wenn beispielsweise die Steuerungskomponente erst später im Entwurfsprozess implementiert wird und erst zu diesem Zeitpunkt die Nachrichten definiert werden. Aus Sicht der Platzierung macht dies keinen Unterschied, da fehlende Nachrichten oder aber fehlende Empfängertasks während der inkrementellen Integration ignoriert werden, ihre potentiellen Einflüsse auf die Deadlines (vgl. Gleichung 4.4 in Kapitel 4.2 auf Seite 52) aber berücksichtigt werden.

Abbildung 6.9 stellt schließlich die Architektur und Topologie des Steuergerätenetzwerkes vor, auf das die Tasks der Komponenten platziert werden sollen. Es besteht aus 16 ECU-Knoten, die über ein hierarchisch organisiertes Netzwerk aus Tokenring-Bussen k_0 bis k_2 zusammengeschlossen sind. Dabei bilden p_0 bis p_4 eine gemeinsame Struktur, die beispielsweise einer Subdomäne in einem realen physikalischem System

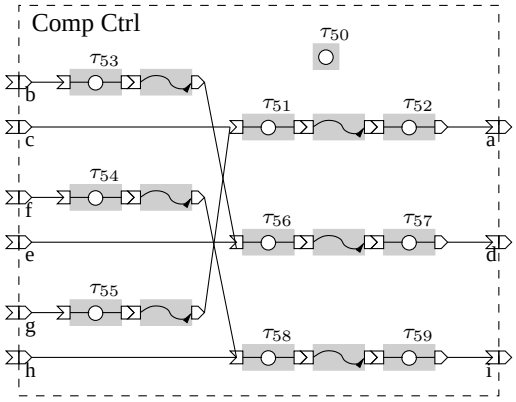


Abbildung 6.8: Die Steuerkomponente des Systems, die auf Basis der eingehenden asynchronen Nachrichten die anderen drei Komponenten steuert.

aus einer der Anwendungsdomänen entsprechen kann. Ebenso sind die anderen Knoten jeweils um ein Bussystem gruppiert. Die Bussysteme sind mit unterschiedlichen Datenübertragungsraten ausgestattet. So haben sowohl k_0 als auch k_2 eine Übertragungsrate von 50 Bytes/msec, während der Backbone k_1 mit 100 Bytes/msec eine doppelt so hohe Übertragungsrate besitzt. p_5 und p_{10} fungieren als Gateways zwischen den benachbarten Teilnetzwerken.

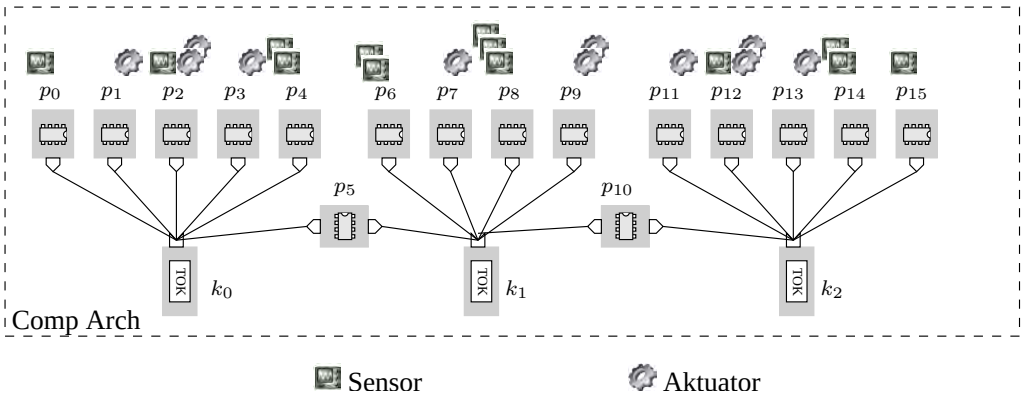


Abbildung 6.9: Gesamtarchitektur des Systems, bestehend aus 16 ECUs und 3 Tokenring-Bussystemen. Aktuatoren und Sensoren an den jeweiligen ECUs sind grafisch angedeutet und schränken den Platzierungsfreiraum einiger Tasks ein.

Jedem Knoten sind eine Menge von Sensoren und Aktuatoren zugeordnet (siehe die Piktogramme in Abbildung 6.9), die von den Tasks genutzt werden. Aktuatoren und Sensoren verfügen jeweils über eine 1:1-Verbindung zu ihren ECUs, so dass Tasks, die einen bestimmten Sensor oder Aktuator benutzen wollen, direkt auf diesem Knoten platziert werden müssen. Auf eine genauere Typisierung und Beschreibung der Aktuatorik und Sensorik soll an dieser Stelle aus Gründen der Übersichtlichkeit verzichtet werden.

Die Komponenten *A1, A2* und *B* sind partiell einzelnen Subdomänen zugeordnet. So können beispielsweise etwa 60% der Tasks von Komponente *A1* nur auf den Kno-

ten um Tokenring k_0 platziert werden, während die übrigen 40% auch auf Knoten um Tokenring k_1 platziert werden dürfen. Umgekehrt ist dies für Komponente *A2* mit Knoten für Tokenring k_2 und k_1 definiert. *B* hingegen kann mehrheitlich nur um k_1 herum angeordnet werden, während *Ctrl* nahezu keine Restriktionen bzgl. der Platzierungsfreiheit aufweist. Einschränkungen der Platzierung auf den Subdomänen sind nur für Tasks vorhanden, die dedizierte Sensorik oder Aktuatorik benutzen, also maßgeblich für die Starttasks einer Taskkette und deren letzte Task. Ein genauerer Überblick über die Platzierungseinschränkungen und die jeweiligen Kenndaten der Tasks als auch Nachrichten und ECU-Knoten ist in Anhang E.2 aufgelistet und soll hier nicht weiter betrachtet werden.

6.3.2 Ergebnisse der inkrementellen Platzierung

Zwecks Bewertung der inkrementellen Integration werden im folgenden jeweils Messungen der inkrementellen Platzierung mit den Messungen zur kompletten Platzierung der Beispielssysteme betrachtet. Dies erlaubt einen Einblick in die Größenordnung der Verluste durch den inkrementellen Prozess. Wir wollen zunächst mit dem Beispiel aus Anhang E.1 starten: Dieses Tasksystem ist aufgeteilt in zwei Komponenten, die jeweils eine relativ freie Verteilung auf Teilen des ECU-Netzwerkes erlauben. Jedes der beiden Tasksets ist zu ca. 70% auf eine Untermenge von 4 bestimmten Prozessorknoten festgelegt, die anderen 30% können nahezu frei verteilt werden. Zunächst wird in Lauf 1 die erste Teilkomponente (Task τ_0 bis τ_{15}) auf die Architektur (die dem Beispiel aus [183] entlehnt ist) platziert, anschließend inkrementell die zweite Komponente. Dieses wurde einmal mit dem Optimierungsziel Minimierung der Token-Rotation-Time und einmal mit Ziel, eine möglichst gleichmäßige Verteilung der Last auf die einzelnen Knoten zu erzielen, durchgeführt. Das Ergebnis dieser Messungen ist in Tabelle 6.5 dargestellt.

Optimierung	Komplett		Inkrementell		Differenz
	Laufzeit	Ergebnis	Laufzeit	Ergebnis	
min(TRT)	2.3 h	$TRT = 1.3ms$	1.9 h	$TRT = 1.5ms$	13%
min(σ)	104 h	$\sigma = 0.066$	2.4 h	$\sigma = 0.069$	5%

Tabelle 6.5: Inkrementelle Platzierung des Beispielssystems aus Anhang E.1 und die Differenz zu einem kompletten Platzierungslauf.

Die inkrementelle Vorgehensweise führt in beiden untersuchten Szenarien zu leicht schlechteren Ergebnissen, als eine komplette Platzierung des Gesamtsystems. Der vergleichsweise höhere Verlust bei dem Optimierungsziel "Minimierung des TRT" ist auf die geringe Nutzung des Bussystems zurückzuführen: Schon die Einsparung einzelner Nachrichten hat bei der geringen Buslast hohe prozentuale Veränderungen zur Folge. Dies ist bei der gleichmäßigen Auslastung der ECU-Knoten anders, da dort alle Knoten sehr hoch belastet sind und die WCETs der einzelnen Tasks alle relativ nahe beieinander liegen. Damit kann das Tauschen einzelner Tasks keine sehr großen Differenzen hervorrufen.

Nebenbei führt die inkrementelle Platzierung auch zu deutlich verringerten Laufzeiten des Platzierungsverfahrens. Es sei hier aber nochmals hervorgehoben, dass dieser Umstand nicht die Motivation für die inkrementelle Integration ist, sondern dass diese durch den Entwicklungsprozess vorgegeben ist, wie bereits in den vorangegangenen Abschnitten beschrieben.

Als zweite Fallstudie soll das im vorherigen Abschnitt dargestellte komplexere System inkrementell plaziert werden. Dabei werden die Komponenten *A1,A2,B* und *Ctrl* jeweils nacheinander und in dieser Reihenfolge platziert. Die Steuerungskomponente mit ihren hohen Freiheitsgraden der Platzierung und den Verbindungen zu allen anderen Komponenten ist dabei bewusst als letztes platziert worden, um auf bereits bestehende Platzierungen Rücksicht nehmen zu können. Optimierungsziel ist hierbei die Minimierung der TRT aller Tokenringe. Das Ergebnis der einzelnen Läufe und der Platzierung insgesamt ist in Tabelle 6.6 abzulesen.

ECU	Lauf 1 (*A1*)	Lauf 2 (*A1*)	Lauf 3 (*B*)	Lauf 4 (*Ctrl*)
p_0	1 2			
p_1	3 4			
p_2	5 6 7 9			
p_3	0 8			
p_4	13 18 19			53 56 57
p_5	14 15			
p_6		36 37		
p_7		30 31 32		
p_8	16 17		46 47 48 49	50 51 52 55
p_9	10 11 12		40 41 42	
p_{10}				
p_{11}		23 24 34 35		
p_{12}		20 25 26 27 29		
p_{13}		28		
p_{14}		33 38 39		54 58 59
p_{15}		21 22	43 44 45	
$\sum TRT_i$	$1.0\mu s$	$2.0\mu s$	$2.0\mu s$	$5.0\mu s$

Tabelle 6.6: Inkrementelle Platzierung des Beispielssystems mit den jeweiligen Ergebnissen der einzelnen Läufe.

Am ersten Lauf ist deutlich die Optimierung zu erkennen: Für den Taskbaum bestehend aus $\tau_1, \tau_2, \tau_3, \tau_4$ und τ_5 müssen τ_1 aufgrund der Nutzung eines Sensors auf Prozessor p_0 und Task τ_4 bzw. τ_5 aufgrund der Nutzung von Aktuatoren auf die Knoten p_1 bzw. p_2 platziert werden. Die Größe der zu übertragenden Datenpakete zwischen den Tasks sind folgendermaßen spezifiziert:

$m_{\tau_1,\tau_2} = 10$ Bytes

$m_{\tau_2,\tau_3} = 10$ Bytes

$m_{\tau_3,\tau_4} = 20$ Bytes

$m_{\tau_3,\tau_5} = 10$ Bytes

Zwecks Minimierung der Kommunikationslast ist es daher optimal, Task τ_3 mit τ_4 auf einen Knoten zu platzieren, da die Nachricht von τ_3 zu τ_4 eine höhere Buslast erzeugt als die Nachricht von τ_3 zu τ_5. Dieser gemeinsame Knoten ist letztlich durch die Benutzung des Aktuators an p_1 durch τ_4 determiniert. Task τ_2 hätte aufgrund der gleich großen Datenvolumen der Nachrichten zu und von τ_2 statt auf p_0 auch auf p_1 platziert werden können; die Wahl dieser Platzierung ist beliebig und vom Optimierungsverfahren mit irgendeinem der beiden Möglichkeiten gewählt worden. Gleiches lässt sich auch für die anderen Läufe des inkrementellen Verfahrens beobachten. Bei der Platzierung der Steuerungskomponente *Ctrl* sind entsprechend dem Optimierungskriterium die Taskketten jeweils auf einem Knoten platziert worden, wobei derjenige Knoten gewählt wurde, dessen Komponenten-Tasks in der Schnittstelle in den Nachrichten maximal sind. Damit wird erreicht, dass die Benutzung der Bussysteme minimal bleibt (Beispielsweise $m_{\tau_{19},\tau_{53}} = 40$ Bytes, $m_{\tau_{15},\tau_{51}} = 20$ Bytes, $m_{\tau_{52},\tau_{10}} = 20$ Bytes, $m_{\tau_{53},\tau_{56}} = 50$ Bytes, etc.).
Im Vergleich dazu bietet Tabelle 6.7 einen Überblick über die Ergebnisse einer kompletten Platzierung des Gesamtsystems mit demselben Optimierungsziel.

ECU	Lauf 1 (*A1 + A2 + B + Ctrl*)
p_0	0 1
p_1	2 3 4
p_2	5 6 7 9
p_3	8
p_4	13 18
p_5	14 15 19
p_6	30 31 34 35 36 37 53 56 57
p_7	32
p_8	16 17 46 47 48 49 51 52 55
p_9	10 11 12 40 41 42 59
p_{10}	20 39 43 44 45 54 58
p_{11}	22 23 24 50
p_{12}	25 26 27 29
p_{13}	28
p_{14}	33 38
p_{15}	21
$\sum TRT_i$	$4.1\mu s$

Tabelle 6.7: Atomare Platzierung des Beispielssystems.

Bei dem Ergebnis der kompletten Platzierung ist gegenüber der inkrementellen Platzierung ein um etwa 18% besseres Resultat erzielt worden. Dies liegt im wesentlichen an der Verschiebung der Nachrichten mit großen Datenmengen auf den doppelt so schnelle Backbone k_1: Während es für den ersten inkrementellen Schritt in Tabelle

6.6 noch egal war, ob beispielsweise τ_{19} auf p_0 oder p_5 platziert wird, fügt die inkrementelle Platzierung von *Ctrl* unweigerlich Nachrichtenübermittlungen zwischen dem Knoten, auf dem τ_{19} platziert ist und anderen Knoten ein. Die komplette Platzierung nutzt hier die Tatsache aus, dass p_5 als für τ_{19} erlaubter Knoten sowohl an k_0 als auch an den Backbone k_1 angeschlossen ist, so dass ein Teil der mit *Ctrl* abzuwickelnden Kommunikation über eben jene schnelle Verbindung erfolgen kann und somit die Summe aller TRTs erniedrigt. Umgehen ließe sich ein solches Verhalten des Optimierungsverfahrens nur, wenn diese Information zugänglich gemacht werden würde, beispielsweise indem Nachrichten ohne Empfängertasks im inkrementellen Platzierungsschritt insofern berücksichtigt werden, als dass der für sie wahrscheinlich schnellste Anschluss benutzt wird. Bei komplexeren hierarchischen Systemen mit unterschiedlichen Bussystemen (Ereignis-basierte und Zeit-basierte Bussysteme beispielsweise) ist dies jedoch so einfach nicht mehr möglich und stellt letztlich nur eine Heuristik dar, die eingehend evaluiert werden müsste.

Die Erwartung, dass die Verwendung zeitbasierter Kommunikationsmedien große Probleme in der inkrementellen Integration verursachen, erfüllt sich indes nicht, wie man an dem relativ moderatem Verlust von ca. 18% sehen kann (der sich womöglich durch eine intelligentere Vorplazierung noch abschwächen lässt, wie oben diskutiert). Dies gilt allerdings nur, solange wir uns bei der Implementierung der Komponenten auf die AUTOSAR-Interpretation einlassen, also Komponenten abstrakt zu spezifizieren und implementieren. Die in der Industrie momentan übliche Praxis, Zeitscheiben schon vorab bei der Spezifikation einer Komponente zuzuordnen, kann in einem solchen Prozess nur zu suboptimalen Lösungen führen. Für eine inkrementelle Integration ist also die auf Abstraktionen des physikalischen Systems basierende Implementierung, die somit auch nahezu frei verschiebbar ist, eine wesentliche Vorraussetzung. Dies schließt explizit nicht aus, über Zeiten zu reden: Es ist durchaus notwendig, bereits Deadlines und Aktivierungsschemata zu frühen Zeitpunkten festzulegen, nicht jedoch den genauen zeitlichen Ablauf, da dies den zu findenden Schedule vorwegnimmt, ohne dass dieser einer Optimierungs- oder wenigstens Validierungsphase unterzogen werden kann.

Die Minimierung der Token Rotation Time als das gewählte Optimierungskriterium in den oben dargestellten Messungen ist natürlich nur eines unter vielen. Es ermöglicht, einen möglichst großen Freiraum auf den Kommunikationsmedien für weitergehende inkrementelle Integrationen zu schaffen. Andere Beispiele wären die Maximierung der Idle-Zeiten, die Tasksysteme auf den Prozessoren und Kommunikationsmedien zu schaffen, wie es beispielsweise in [207] und [144] vorgeschlagen wird. Beide Arbeiten beschäftigen sich allerdings mit reinen zeitbasierten Systemen, sowohl seitens der Tasksysteme als auch seitens der Nachrichtenübertragung, so dass hier kein direkter Vergleich gezogen werden kann. Für eine exklusiv die Kommunikationsmedien betrachtende Optimierung ist die Minimierung der Tokenring Rotation Time übrigens den Optimierungsmethoden in [207] und [144] sehr ähnlich.

6.4 Mehrfach-Parametrisierte Optimierung

Das SAT-basierte Optimierungsverfahren ist, wie zahlreiche andere Verfahren auch, in der Lage auch für multiple, sich womöglich widersprechende, Optimierungskriterien eine Optimierung durchzuführen. Dabei obliegt es dem Benutzer (oder einem automatischen Verfahren), die Gewichtung der einzelnen Kriterien anzugeben. Mehrfach-parametrisierte Optimierungen sind wichtig, wenn nicht nur einzelne Randbedingungen in der Synthese eines Systems berücksichtigt werden müssen und dienen vornehmlich der Exploration des Entwurfsraumes.
Betrachtet man traditionelle Optimierunsgverfahren, wie beispielsweise Simulated Annealing, Genetische Algorithmen oder auch Linear Programming, so wird die Optimierung mehrfacher Kriterien in einer Zielfunktion (in stochastischen Verfahren die Energiefunktion) zusammengefasst, die kleiner oder größer als ein bestimmter Schwellwert ist. Eine allgemeine Optimierungsfunktion ist demnach durch

$$f = \alpha_1 \cdot o_1 + \ldots \alpha_n \cdot o_n$$

gegeben, wobei o_i die verschiedenen Teil-Zielfunktionen und α_i Gewichtungsfaktoren zur Priorisierung bestimmter Kriterien sind. Dieses Verfahren kann problemlos in das in Kapitel 5.4 vorgestellte Platzierungsverfahren integriert werden. Im folgenden wird dies an einem Beispiel verdeutlicht, dass neben der Minimierung der Latenzen auf einem Kommunikationsmedium zusätzlich noch den Leistungsaufnahme des Systems minimieren soll.
Hierzu führen wir zunächst eine weitere Eigenschaft von Tasks ein, die die Leistungsaufname auf unterschiedlichen Prozessorknoten abbildet:

$$\mathcal{P}_i : P \rightarrow \mathbb{N}$$

Der Einfachheit halber gehen wir davon aus, dass eine Task auf einem Prozessor eine bestimmte Leistungsaufnahme hat. Die Gesamtleistungsaufnahme des Systems beträgt dann

$$\mathcal{P} = \sum_{\forall \tau_i \in \mathcal{T}} \mathcal{P}_i(\Pi(\tau_i))$$

Dies lässt sich einfach in die Form von arithmetischen Ungleichungen überführen, indem zunächst eine Variable `power` für die globale Leistungsaufnahme und eine Variable $\texttt{power}_i$ für die Leistungsaufnahme einer Task eingeführt wird. Entsprechend der obigen Formel kann nun zunächst für alle Tasks die Leistungsaufnahme gemäß der folgenden Ungleichungssysteme aufgebaut werden.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\texttt{place}_i = p) \rightarrow \texttt{power}_i = \mathcal{P}_i(p) \tag{6.15}$$

Die Leistungsaufnahme des Gesamtsystems ist dann gegeben durch:

$$\texttt{power} = \sum_{\forall \tau_i \in \mathcal{T}} \texttt{power}_i \tag{6.16}$$

Als konkrete Probleminstanz soll nun das in Anhang E.1 dargestellte Tasksystem mit der in [183] präsentierten Architektur betrachtet werden. Die Leistungsaufnahme

der einzelnen Tasks ist global an den Knoten abzulesen und für alle Tasks auf einem gemeinsamen Knoten gleich[1]. Die folgende Tabelle 6.8 zeigt die Leistungsaufnahme von Tasks auf den jeweiligen Knoten.

Knoten	*Leistungsaufnahme*
p_0, p_1, p_3, p_4	5 mW
p_2, p_6	3 mW
p_5, p_7	2 mW

Tabelle 6.8: Leistungsaufnahme der Tasks in Abhängigkeit der Prozessorknoten

Für dieses Beispiel kann nun in jedem Lauf des SAT-Verfahrens eine Optimierungsfunktion angegeben werden, die in der hier benutzten konkreten Probleminstanz sowohl den TRT des Tokenring als auch die Leistungsaufnahme minimieren soll:

$$\alpha_{TRT} \cdot \texttt{TRT}_k + \alpha_{power} \cdot \texttt{power} < C_{opt} \tag{6.17}$$

Die Gewichtungskonstanten α_{TRT} und α_{power} sind so zu wählen, dass sie sowohl die Normalisierung als auch die Wichtigkeit der einzelnen Teiloptimierungs-Klauseln ausdrücken können (in der hier benutzten Beispielinstanz kann $\texttt{TRT}_k$ Werte von $0ms$ bis $4ms$ und `power` Werte von $60mW$ bis $150mW$ annehmen).
Zwecks besserer Beobachtbarkeit dieser Optimierungen wurde das Beispiel aus Anhang E.1 architekturseitig ein wenig erweitert: Da die Speicherbegrenzung der limitierende Faktor in diesem System ist, wurde der Speicher von Knoten p_0, p_1 und p_7 verdoppelt. Für die Optimierungsfunktion sind die Gewichtungsfaktoren so gewählt, dass die Leistungsaufnahme mit einem Faktor von etwa 3 eingeht, d.h. $\alpha_{TRT} = 10.0$ und $\alpha_{power} = 1.0$. Das Ergebnis dieser Optimierung kann Tabelle 6.9 entnommen werden.

Optimierung	*TRT*	*Leistungsaufnahme*
$10 \cdot \texttt{TRT}_k + \texttt{power}$	$1.3ms$	106 mW

Tabelle 6.9: Mehrfach-parametrisierte Optimierung mittels statischer Optimierungsfunktion.

Die auf binäre Suche basierende Art und Weise der Optimierung einer Platzierung bietet aber neben der oben dargestellten Variante der mehrfach-parametrisierten Optimierung noch eine weitere, durch andere Optimierungsverfahren nicht zu leistende, Methodik an: So können anstelle einer einfachen Gewichtung komplexere Funktionen angegeben werden, die sich die inkrementelle Natur der binären Suche zu Nutze machen. Dabei können beispielsweise zunächst nur für Teile der Optimierungskriterien Lösungen bis zu einem gewissen Schwellwert gesucht werden und anschließend werden weitere Faktoren hinzu genommen. Dies führt zu Optimierungskriterien, die während der Optimierung dynamisch verändert werden können. Auch dies wollen wir am Beispiel der Minimierung von Latenzzeiten auf dem Bussystem

[1]Dies ist natürlich eine sehr vereinfachende Sicht der Wirklichkeit, allerdings soll sie hier nur zu Demonstrationszwecken dienen und kann jederzeit realistischer modelliert werden.

und Minimierung der Leistungsaufnahme an der oben dargestellten Probleminstanz demonstrieren.
Die Optimierung wird in diesem Experiment so gesteuert, dass zunächst einseitig die Leistungsaufnahme bis unter einen gegebenen Schwellwert sinkt. Anschließend wird sie unterhalb dieses Schwellwertes gehalten und die Latenzzeit auf dem Kommunikationsmedium optimiert. Der Schwellwert wird hier mit $110mW$ eingestellt und liefert das in Tabelle 6.10 dargestellte Ergebnis. In einem zweiten Experiment wird andersherum verfahren und zunächst der TRT unter einen Schwellwert von $3ms$ optimiert mit anschließender Minimierung der Leistungsaufnahme.

Optimierung	*TRT*	*Leistungsaufnahme*
$\texttt{power} < 110mW$, dann $\texttt{TRT}_k$ minimieren	$0.9ms$	109 mW
$\texttt{TRT}_k < 3ms$, dann `power` minimieren	$2.9ms$	96 mW

Tabelle 6.10: Mehrfach-parametrisierte Optimierung mittels dynamischer Optimierungsfunktion.

Aus den Messungen ist ersichtlich, dass die dynamische Optimierung besser in der Lage ist, bestimmte vorgegebene Designmargen auszunutzen. So ist beispielsweise der Unterschied bzgl. der Leistungsaufnahme zwischen der Optimierung mittels Funktion und mittels dynamischer Klauseln nur unter 3%, jedoch wirkt sich dies gravierend auf die letztlich erzielte TRT auf dem Kommunikationsmedium aus (Verkürzung auf etwa 69%). Derartige Ergebnisse sind durch statische Optimierungsfunktionen nur sehr schwer oder gar nicht automatisch zu erreichen, sondern bedingen einen manuellen Eingriff in den Ablauf des Optimierungsverfahren. SAT-basierte Optimierungsverfahren, wie sie in dieser Arbeit vorgestellt wurden, bieten hingegen die Möglichkeit, nahezu beliebig komplexe dynamische Optimierungsfunktionen anzugeben, die eine zielgerichtetere Exploration des Entwurfsraumes erlauben.

6.5 Virtuelle Integration in frühen Phasen

6.5.1 Sinn und Zweck der virtuellen Integration

Die virtuelle Integration eines komplexen eingebetteten Systems kann dazu genutzt werden, eine Verifikation der Systemanforderungen auf einem höheren Abstraktionsniveau durchzuführen. Im folgenden wird davon ausgegangen, dass die Spezifikation des Systems bereits in modellbasierten und ausführbaren Spezifikationssprachen vorliegt, wie es zunehmend in der industriellen Praxis üblich ist[157]. Somit kann der funktionale Anteil des Systems bereits innerhalb einer virtuellen Integration durchgeführt werden, die beispielsweise eine Co-Simulation der ausführbaren Spezifikationen zur Verfügung stellt und auf diese Weise das Gesamtverhalten des Systems bzgl. der funktionalen Sicht evaluiert. Dies kann zusätzlich durch Rapid Prototyping[107] unterstützt werden, mit dem Ziel schon in frühen Phasen durch eine Variante von Hardware-In-The-Loop-Tests funktionale Anforderungen zu verifizieren und validieren, die nicht unbedingt formal zugänglich sind (Rapid Prototyping in diesem Sinn

dient also eher der Validation als der Verifikation[1]).
Bezüglich des Zeitverhaltens ist eine solche virtuelle Integration und die damit einhergehende Verifikation zeitlicher Abläufe nicht so einfach zu erreichen. Zwar existieren einige Ansätze, zeitliches Verhalten auf modellbasierten Spezifikationen in die Verifikationsaufgabe zu integrieren, aber sie sind in der Regel auf abstrakte Zeitbegriffe wie Timer oder Schrittzahlen des Modells (vgl. [200]) begrenzt. Auch eine Berücksichtigung potentieller Schrittlängen in Form von physikalischer Zeit, wie sie beispielsweise in [199] oder [48] angegeben werden, führen nicht sehr weit, da sie weder Architekturbelange, wie etwa Kommunikationen über Bussysteme, noch Preemptionen unter einem Echtzeit Betriebssystem berücksichtigen können. Effektiver ist hier die Nutzung sogenannter Restbussimulationen[2][84], die ausgehend von einer detaillierten Architekturbeschreibung modellbasierte Spezifikationen mit einer Simulation physikalischer Hardware verbinden. Hierbei ist es auch möglich, beispielsweise Betriebssystemeinflüsse wie Preemptionen, zu berücksichtigen. Beispielhaft sei hier das auf der Polis-Methodik[51] basierende Werkzeug VCC der Firma Cadence genannt, das eine derartige Co-Simulation anbietet.
Problematisch an diesem Ansatz ist zum einen die fehlende formale Basierung des Nachweises der Echtzeiteigenschaften (Simulationsbasierte Verifikationen können bekanntlich nur so gut sein, wie die betrachteten Testfälle es erlauben, sind aber niemals vollständig) und auf der anderen Seite die notwendige genaue Kenntnis der Steuergeräte-Architektur, etwa Typ und Ausgestaltung der verwendeten Bussysteme, inklusive der genauen Konfiguration des Systems. Ein solches Vorgehen ist insbesondere in explorativen Designphasen nicht einsetzbar, da zum einen jede mögliche Konfiguration des Systems simuliert werden müsste und zum anderen dies nicht automatisiert werden kann.
Einen Schritt weiter gehen an dieser Stelle komponenten-basierte Methodologien, wie beispielsweise die Rich Component Models[37]. Nicht-funktionale Sichten, wie die Echtzeit-Sicht eine ist, werden mit Eigenschaften ausgestattet, die eine abschätzende Natur haben. Können bereits auf hohen Abstraktionsebenen die Echtzeiteigenschaften nachgewiesen werden, so müssen in den tieferen Schichten nur noch die jeweiligen vertikalen Anforderungen abgesichert werden. Die Beweisführung der zeitlichen Korrektheit ist in diesem Sinne formal und kompositionell, führt also anders als Simulationen den vollständigen Nachweis korrekten Verhaltens. Der Preis für die abschätzenden Eigenschaften, beispielsweise eine Budgetierung von Deadlines auf einzelne Subsysteme, liegt in den daraus folgenden eingeschränkten Freiheitsgraden bei der Implementierung. An dieser Stelle können relaxierte Deadline-Synthesen, wie sie im vorangegangenen Kapitel beschrieben wurden, helfen: Deadline-Synthese-Techniken führen quasi eine Integration durch, die allerdings nur einen hinweisenden

[1]Die Begriffe Validation und Verifikation sind hier im klassischen Sinne des Systems- und Software-Engineering zu verstehen. Validation überprüft das funktionale Verhalten auf die Anforderungen der Stakeholder hin, Verifikation überprüft die Korrektheit des Designs bzgl. der formulierten Anforderungen.

[2]Restbussimulationen im klassischen Sinn sind gemischte HW/SW-Simulationen, in denen eine Mischung aus physikalischen Steuergeräten und modellbasierten, simulativ ablaufenden Spezifikationen getestet werden. Hier verstehen wir ebenfalls die Ersetzung der physikalischen Einheiten durch detaillgetreue Simulationsmodelle als Restbussimulation.

Charakter auf eine mögliche Implementierung aufweist und damit eine gute Metrik für beispielsweise das Deadline-Tailoring bereitstellen.
Tailoring von Zeiten ist aber aus einer ganz anderen Perspektive noch ein wichtiges Schlagwort: In Zulieferer-dominierten Prozessen, wie es beispielsweise die Automobilindustrie ist, werden in sehr frühen Phasen des Designs die Spezifikationen für Funktionen/Features und/oder Steuergeräte an die Zulieferer vergeben. Vertragliche Bestimmungen in einem solchen Prozess legen fest, welche Eigenschaften, insbesondere auch Echtzeit-Eigenschaften, das zu entwickelnde Subsystem einhalten soll. Dies geht bis hinunter zu einzelnen Slots auf einem FlexRay-Bus (vgl. [17] und [121]). Massiv eingesetztes Concurrent Engineering fordert dann, dass das Tailoring es erlaubt, die Gesamtfunktionalität, hier insbesondere bzgl. der Echtzeit, zu erhalten. In diesem Sinne implementiert die den Rich Component Models zugrunde liegende Designmethodologie unter Zuhilfenahme beispielsweise der relaxierten Deadline-Synthese die für ein solches Prozessmodell notwendigen virtuellen Integrationen in frühen Designphasen und stützt sich gleichzeitig auf formale Nachweise der Echtzeiteigenschaften mithilfe der hier vorgestellten Methodiken.

Um dies aus zwei Blickwinkeln etwas näher zu betrachten, werden wir im folgenden zunächst zeigen, wie man mit unsicheren Informationen, hier am Beispiel von Varianzen bei der Ausführungszeit von Tasks, die in Kapitel 5 vorgestellten Ansätze nutzen kann. Anschließend wird in Form eines Ausblicks skizziert, wie dieses Verfahren sich in einen explorativ organisierten Gesamtprozess integrieren ließe.

6.5.2 Einsatz am Beispiel variabler WCETs

Eine Quantifizierung zeitlicher Bedingungen in frühen Entwurfsphasen kann nur unter einer gewissen Unschärfe erfolgen, da üblicherweise noch keine Implementierung (oftmals auch noch keine ausführbare Spezifikation) vorhanden ist. Selbst wenn in einem Komponenten-basierten Prozess Re-Use von Komponenten betrieben wird, ist zunächst einmal nicht endgültig klar, ob die Architekturanteile der Komponente so erhalten bleiben. Verfolgt man die Strategie des AUTOSAR-Ansatzes[69], so wird generell von der Architekturabhängigkeit abstrahiert. In dem hier in Kapitel 6.1 vorgestellten Ansatz, ebenso wie in anderen Komponenten-basierten Ansätzen, muss dies nicht zwangsläufig so sein, sondern der Designer hat mehr Freiheitsgrade. In jedem Fall wird es aber Anteile des Designs geben, die neu sind und daher zunächst einmal nicht weiter bzgl. ihres Zeitverbrauchs spezifiziert sind. Das Tailoring im Entwurfsprozess muss aber auf der anderen Seite in eben diesen frühen Entwurfsphasen bereits die Möglichkeit haben, Echtzeit für die Zulieferer zu quantifizieren. Die Hilfestellung durch beispielsweise das relaxierte (oder auch andere) Deadline-Syntheseverfahren setzt aber voraus, dass zumindest wesentliche zeitliche Angaben, wie End-to-End-Deadlines, Perioden oder WCETs bekannt sind. Die ersteren beiden Faktoren sind aus der Anforderungsdefinition zu extrahieren, wie in Kapitel 2 eingehend diskutiert. Letzteres kann nur ein Schätzwert auf Basis der Expertise erfahrener Entwickler sein bzw. könnte aus einer Codegenerierung heraus entstehen, wenn bereits eine ausführbare modellbasierte Spezifikation vorliegt. Gleichzeitig können diese Faktoren auch genutzt werden, um explorative, virtuelle

Integrationen/Implementierungen zu testen. Dieses Thema wird im nächsten Kapitel etwas detaillierter skizziert werden.
In jedem Fall bleibt eine Rest-Unschärfe, die im Extremfall dazu führen kann, dass ein Zulieferer-Subsystem seine zeitlichen Deadlines nicht einhalten kann, während andere vielleicht ihr zur Verfügung stehendes Zeitintervall lange nicht ausnutzen. Zwar kann sich solches Verhalten im Prinzip ausgleichen, aber konkret hängt dies von der zur Verfügung stehenden Hardware-Architektur ab: Time-Triggered Bussysteme sind beispielsweise überhaupt nicht für derartige Schwankungen ausgelegt, und dies könnte dazu führen, dass komplette Schedules umgeplant werden müssen. Letzteres beeinflusst dann nicht mehr nur lokal die Subsysteme einiger weniger Zulieferer, sondern ist potentiell ein kostenintensives globales Designproblem und verlangt umfangreiche Entwurfsiterationen. Hier ist folglich das Bestreben, bei dem Tailoring der zeitlichen Bedingungen nicht zu eng zu planen, d.h. es wird eine Marge um die angegebene WCET herum geben, die noch zu validen Lösungen eines Systems führt. Dies zu bestimmen ist dann Aufgabe einer virtuellen Integration mit der Maßgabe, den Spielraum für die einzelnen WCETs zu maximieren.

Schedulinganalysen, wie sie in Kapitel 4.2 durchgeführt wurden, geben zunächst nur die Antwort, ob ein System unter einer bestimmten Konfiguration feasible ist oder nicht. Freiräume beispielsweise bzgl. einer Varianz der WCET werden nicht angegeben, sind hier aber gefordert. Die Platzierung als virtuelle Integration könnte hier auch die Möglichkeiten der Varianz jedes einzelnen Tasks untersuchen, die sich durchaus problemspezifisch von anderen Tasks unterscheiden können.
Um dieses innerhalb des in Kapitel 5 vorgestellten Verfahrens zur Verfügung zu stellen, haben wir die fixierte Definition der WCETs eines Tasks für die jeweiligen Prozessoren aufgehoben und gehen stattdessen von einem Mittelwert aus, um den herum ein Varianzintervall angegeben wird, beispielsweise ±20% des angegebenen Mittelwertes. Eines der Optimierungsziele, etwa innerhalb einer mehrfachparametrisierten Optimierung wie in Kapitel 6.4 diskutiert, ist die Maximierung der Summe der WCET-Intervalle[1]. Hierfür werden zunächst die Grundgleichungen der architekturabhängigen WCET-Bestimmung in Gleichung 5.7 auf Seite 133 in Kapitel 5 verändert, indem sie variabel gewählt werden können:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\texttt{place}_i = p) \rightarrow ((\texttt{wcet}_i \geq (1.0 - y) \cdot c_i(p)) \wedge (\texttt{wcet}_i \leq (1.0 + y) \cdot c_i(p))) \,, \tag{6.18}$$

wobei $0 \leq y \leq 1.0$ die Varainzbreite bezeichnet und von 0% bis 100% gewählt werden kann. Anschließend wird als Optimierungskriterium die Maximierung der Summe `wcet_opt` aller $\texttt{wcet}_i$-Werte gewählt.

$$\texttt{wcet\_opt} = \sum_{\forall \tau_i \in \mathcal{T}} \texttt{wcet}_i \tag{6.19}$$

Auf diese Weise ist das Verfahren in der Lage, mit variablen WCETs eine Platzierung (und damit auch eine virtuelle Integration) durchzuführen und diese gleichzeitig so

[1] Im Prinzip kann dies natürlich auch feiner strukturiert werden, also beispielsweise mittels einer Gewichtung bestimmter Tasks; dies ist jedoch momentan nicht implementiert

zu optimieren, dass der maximale Freiraum bei der Ausgestaltung der jeweiligen Ausführungszeiten von Tasks in der Implementierung genutzt werden kann, bzw. dem OEM auch dann noch ein korrektes Systemverhalten garantiert, wenn die vorgegebenen WCETs zwar nicht eingehalten wurden, aber innerhalb des durch dieses Verfahren ermittelten Intervalls liegen. Ein weiteres Anwendungsszenario ist beispielsweise die Bestimmung der Varianz von WCETs bei einer bereits fest vorgegebenen Platzierung: Im Zuge der inkrementellen Platzierung kann eine Systemkonfiguration so erweitert werden, dass als einziger Freiheitsgrad dem Repository die Variabilität der WCET-Werte hinzugefügt und anschließend eine erneute Platzierung erfolgen wird. Das Ergebnis wird zeigen, in welchem Rahmen vorgegebene WCETs überschritten werden dürfen, ohne das System infeasible zu machen.

Zwecks Demonstration des Verfahrens wird im folgenden die Anwendung auf das Beispielsystem aus [183] gezeigt. Die Varianz der WCETs wurde hierbei auf 50% festgelegt und damit eine Platzierung des Ursprungssystems durchgeführt. Optimierungskriterium war ausschließlich die Maximierung der WCETs, nicht — wie bisher — die Minimierung der TRT auf dem Tokenring-Bus. Das Ergebnis ist in Tabelle 6.11 für alle Tasks des Beispiels dargestellt.

τ_0	140%	τ_1	148%	τ_2	140%	τ_3	142%	τ_4	119%
τ_5	136%	τ_6	100%	τ_7	144%	τ_8	143%	τ_9	135%
τ_{10}	101%	τ_{11}	136%	τ_{12}	128%	τ_{13}	120%	τ_{14}	111%
τ_{15}	102%	τ_{16}	112%	τ_{17}	104%	τ_{18}	136%	τ_{19}	138%
τ_{20}	100%	τ_{21}	104%	τ_{22}	137%	τ_{23}	120%	τ_{24}	112%
τ_{25}	144%	τ_{26}	113%	τ_{27}	130%	τ_{28}	105%	τ_{29}	131%
τ_{30}	142%	τ_{31}	113%	τ_{32}	146%	τ_{33}	127%	τ_{34}	105%
τ_{35}	144%	τ_{36}	150%	τ_{37}	150%	τ_{38}	103%	τ_{39}	149%
τ_{40}	128%	τ_{41}	149%	τ_{42}	150%				

Tabelle 6.11: WCET-Varianz gegenüber der gegebenen WCET im [183]-Beispiel bei einer maximalen Abweichung von +50%.

Es ist ersichtlich, dass für einige Tasks eine deutliche Steigerung der WCET möglich ist, die nichts an der Feasibility des Gesamtsystems ändert, trotzdem dieses Beispiel als eher eng bezeichnet werden kann (Ergebnisse unter 100% wurden vermieden, indem die WCET-Bestimmung nur auf größere Werte zugelassen wurde). Insbesondere bei den "schwereren" Tasks τ_6 und τ_{10} zeigt sich, dass kaum Spielraum besteht, da eine prozentuale Abweichung natürlich im absoluten Zeitbegriff deutlich schwerer wiegt. Insgesamt zeigt sich jedoch kein Zusammenhang zwischen der vorgegebenen WCET und der möglichen Varianz; so widersprechen beispielsweise Task τ_9 als sehr schwerer Task mit einer recht großen möglichen Abweichung von 135% und Task τ_{20} mit einer sehr kleinen angegebenen WCET aber keiner möglichen Abweichung vollkommen einer solchen Gesetzmäßigkeit. Vielmehr zeigt dieses Beispiel recht gut, dass keine Platzierungs-unspezifische Metrik für die mögliche Abweichung der WCET existiert und es stattdessen vorteilhafter ist, relaxierte Metriken in einer virtuelle Integration einzusetzen, um den verbleibenden Spielraum zu quantifizieren.

6.5.3 Durchmusterung des Lösungsraumes

Virtuelle Integration kann, neben den hier bereits vorgestellten Szenarien, auch auf einer übergeordneten Entwurfsebene eingesetzt werden und dabei helfen, eine Exploration des Entwurfsraumes durchzuführen, hier unter dem Gesichtspunkt der Echtzeitfähigkeit. In diesem Sinne kann die in Kapitel 5 vorgestellte Methodik benutzt werden, um im Bereich des Hardware/Software Co-Designs unterstützende Maßnahmen zu liefern. Die Kombination von relaxierten Metriken, mehrfachparametrisierter Optimierung und der Variabilität beispielsweise der WCETs aus dem letzten Kapitel führt zu einer grundlegenden Methodik, eine Architekturexploration durchzuführen. Wir wollen in diesem Abschnitt kurz skizzieren, wie sich eine solche Integration in einen übergeordneten Gesamtentwurfsprozess durchführen ließe und welche weiteren unterstützenden Methodiken dafür vorzusehen wären.
Hierzu notwendig ist eine ausführbare Spezifikation des Systems sowie eine Abbildung der Komponenten einer solchen modellbasierten Spezifikation in die Task-Welt. Angelehnt an Arbeiten im Bereich der Highlevel-Synthese von Software unter Berücksichtigung des Stromverbrauches[141] könnte ein Verfahren gefunden werden, welches für typische Konstrukte einer automatischen Codegenerierung verschiedene Arten der Implementierung generalisiert. Daraus könnten Funktionen entstehen, die für bestimmte Parameter, beispielsweise Zeit, Stromverbrauch und Kosten, Kennfelder einer Implementierungsexploration liefern. Die grundlegende Idee ist dabei, dass z.B. die Implementierung typischer Konstrukte als Ko-Prozessor aus einem Hardware/Software-Co-Design heraus nicht nur Einflüsse auf die Ausführungszeit hat, sondern auch andere Entwurfsparameter berührt und dies in einer Abschätzung generalisiert werden kann. Für den Echtzeitnachweis könnte beispielsweise eine bezüglich der Prozessorarchitektur parametrisierbare WCET-Analyse (vgl. beispielsweise [156]) benutzt werden, verschiedene Implementierungsszenarien durchzurechnen und für eine typische Folge von Operationen eines Programms, welches aus einer Codesynthese der ausführbaren Spezifikation generiert wurde, eine generalisierende Komponente zu erzeugen. Implementierungen sind in diesem Sinne dann auch Kompositionen unterschiedlicher Komponenten, die jeweils bestimmte Eigenschaften im System induzieren. Die horizontale Komposition dieser Implementierungskomponenten legt für jede Taskkomponente daraus resultierende Eigenschaften fest, die letztlich beispielsweise die WCET eines solchen Tasks bestimmen. Bildlich stellt sich damit ein Basiselement, wie es in Kapitel 6.1 definiert wurde, als Sammlung alternativer Verschaltungen von Atomen dar, die in ihrem Zusammenspiel bzgl. der jeweils betrachteten Entwurfsgrößen analysiert werden müssen. Dies kann in diesem Beispiel bzgl. der Echtzeit durch Anwendung von parametrisierten WCET-Analysen geschehen. Die Alternativen erzeugen Paare von Implementierungsalternativen für eine Platzierung, z.B. (WCET,Power,Cost)-Paare, die vom Optimierungsverfahren entsprechend dem gewählten Optimierungskriterium ausgewählt werden können und letztlich eine gewählte Architektur evaluieren. Dieses Verhalten kann selbstverständlich auch Rückflüsse auf die Abbildung der modellbasierten Spezifikationskomponenten in Tasks haben und könnte rückkoppelnd auch diese Abbildung adaptieren.

Natürlich kann diese kurze Skizze einer potentiellen Methodik für einen übergeord-

neten Entwurfsprozess an dieser Stelle kein vollständiges Bild liefern. Aber es verdeutlicht, dass die in dieser Arbeit vorgestellten Verfahren nicht nur auf die Ebene der Tasksysteme und Steuergerätenetzwerke innerhalb eines Entwurfsprozess in ihrer Nützlichkeit beschränkt sind, sondern dass sie durchaus auch im Gesamtkontext des Entwurfes — auch und gerade in frühen Phasen — durch die zur Verfügungstellung von abschätzenden Parametern einen entscheidenden Beitrag zur Exploration des Entwurfsraumes liefern können. Eingebettet in eine Design-Methodologie, wie sie beispielsweise die Rich Component Modelle[37] darstellen, könnten sie den Entwurfszyklus eingebetteter verteilter Echtzeitsystem deutlich verkürzen und dem Designer wertvolle Hinweise liefern.

6.6 Zusammenfassung und Ausblick

Dieses Kapitel hat gezeigt, unter welcher Entwurfsmethodik ein Einsatz der in dieser Arbeit vorgestellten Verfahren gewinnbringend eingesetzt werden kann. Die Komponenten-basierte Methodologie in Verbindung mit einem inkrementellen Integrationsprozess löst dabei inhärente Probleme beim Design komplexer eingebetteter Systeme an den Schnittstellen zu Zulieferern und verbindet dies mit dem formalen Nachweis der Korrektheit zeitlichen Verhaltens. Mehrfach-parametrisierte Optimierungen sowie insbesondere die Benutzung einer virtuellen Integration lassen erahnen, welches Potential insgesamt in der Verbindung von Komponenten-orientierter Entwurfsstile und automatischer Platzierungsverfahren liegen und schlagen letztlich eine Brücke zu den weiteren Teilen dieser Arbeit. In Abschnitt 6.5.3 haben wir zudem dargestellt, wie wir uns eine Weiterentwicklung sowohl des Prozesses als auch der Integration der hier vorgestellten Ansätze vorstellen.

Bzgl. der Deadline-Synthese und deren Einbettung in den Entwurfsprozess lassen sich natürlich auch weitreichendere Ergänzungen implementieren. So könnten Deadlines nicht als festgelegte Werte, sondern als Intervalle definiert werden, und die Optimierungstechnik bekommt als weiteres Optimierungskriterium im Sinne einer mehrfach parametrisierten Optimierung die Aufgabe, Antwortzeiten einer bestimmten Güte (z.B. besonders klein) zu erzeugen. Zwar lässt sich dann nicht mehr gewährleisten, dass die gefundene Lösung tatsächlich eine valide Lösung ist, aber dieser Nachweis ist dann Aufgabe eines übergeordneten Layers durch eine vertikale Verifikation im Sinne der in [37] vorgeschlagenen Methodik und würde im Gegenzug zu einer Änderung entweder der Anforderungen an diese Teilkomponente oder zu einer Umstrukturierung des Designs führen.
Auf der anderen Seite ließe sich anstelle kleiner Antwortzeiten auch die Deadline jeder Task und Nachricht geringfügig (im Rahmen der angegebenen Intervalle) durch das Optimierungsverfahren so variieren, dass die Summe aller Deadlines einer Taskkette kleiner oder gleich der End-to-End-Deadline ist. Dies hätte zwar, wie das oben skizzierte Verfahren auch, den Nachteil einer zu großen Jitterüberapproximation, vermeidet aber andererseits eine notwendige holistische Analyse und bietet

trotzdem eine feste Schnittstelle[1].

Bezüglich der inkrementellen Integration sollten weitere Messungen auch mit anderen Optimierungsfunktionen durchgeführt werden, um den in dieser Arbeit entstandenen ersten Eindruck zu festigen. Insbesondere ist hierfür auch noch eine Integration der zeitbasierten Tasks und Nachrichtenübertragung notwendig, da diese in einer inkrementellen Platzierung am ehesten problematisch erscheinen und zudem eine bessere Vergleichbarkeit zu den wenigen bisher vorliegenden verwandten Arbeiten bieten.
Ein ganz wesentlicher Punkt, den wir hier nur durch die in Kapitel 4.1 erörterte Simplifizierung berücksichtigt haben, ist die Frage der Komposition von Komponenten bzgl. ihrer Echtzeitschnittstellen. Hier führen wir zur Zeit Arbeiten im Bereich der Rich Component Modelle durch ([192]), die basierend auf Timed Automata ähnlichen Beschreibungen von Aufrufschemata der Tasks entsprechende Aufrufschemata von Nachfolgertasks synthetisieren oder auf Kompatibilität testen können. Die grundlegende Idee geht dabei von einer Bibliothek von Basisspezifikationen aus, die sukkzessive durch das Zusammenschalten der Schnittstellen reichhaltiger wird. Insbesondere sind hier auch komplexe Mischungen aus zeit- und ereignisbasierter Ansteuerung möglich, die letztlich im Framework der Rich Component Modelle auch die Auswahl der notwendigen Analysetechniken steuern. Es bleibt abzuwarten, welcher Anteil dieses Vorhabens letztlich vollständig automatisierbar ist und wo es der menschlichen Interaktion bedarf. In jedem Fall wird es die hier vorgestellten Techniken sinnvoll ergänzen und den komponentenbasierten Entwurfsstil unterstützen.

Der nun noch folgende Teil dieser Arbeit greift auch den Ansatz aus Kapitel 6.5.3 wieder auf und beschäftigt sich mit einer spezifischen, echtzeitunterstützenden, Implementierung von Steuergeräten. Diese ist maßgeblich im Umfeld der Architekturexploration anzusiedeln, wir haben jedoch schon in Kapitel 4.1 dargelegt, dass auch im Platzierungsprozess Parameter aus Teil II von Nöten sind (Echtzeiteinflüsse von Preemptionskosten durch das Echtzeitbetriebssystem) und werden eine Integration in die in Kapitel 5 vorgestellte Methodik darstellen. Neben der Reduzierung der betriebssystembedingten Kosten wird der nachfolgende Ansatz auch die Vorhersagbarkeit der WCET-Analysen, wie sie in Kapitel 2.4 kurz vorgestellt wurden, durch architektonische Maßnahmen unterstützen und damit insgesamt auch der formalen Absicherung von Schedulinganalysen dienen.

[1] Für andere, nicht zur Taskkette gehörenden Komponenten ist die Festigkeit des Jitters die herausragende Eigenschaft. Die Deadlines hingegen sind im Prinzip nur für Tasks und Nachrichten der eigenen Taskkette interessant und bilden keine — sieht man von der Priorisierung ab — Interferenzmöglichkeit bei der Platzierung anderer Komponenten.

Teil II

Multithreading in Echtzeitsystemen

Kapitel 7

Betriebssystemkosten

7.1 Bedeutung des Scheduling für die Echtzeit

Bei der Berechnung von Ausführungszeiten einer Task, wie sie beispielsweise in Kapitel 2.4 dargestellt wurden, wird davon abstrahiert, dass eine Task in eine Umgebung, meist einem Echtzeit-Betriebssystem (RTOS), eingebunden ist. Diese Betrachtung ist für die Ausführungszeit sinnvoll, da die reinen Kosten der Taskausführung system-unabhängig sind. Einflüsse, die z.B. durch Preemptionen auf das Zeitverhalten ausgeübt werden, sind hingegen abhängig von der umgebenden Software-Schicht. Preemptionen beispielsweise sind nur dann in die Betrachtung einzubeziehen, wenn auch tatsächlich auf einem, der Task zugeordneten, Prozessor-Knoten ein preemptives Scheduling aktiviert ist. Informationen dieser Art sind daher erst *nach* einer Verteilung von Tasks auf ein Architekturnetzwerk, wie es in Teil I dieser Arbeit vorgestellt wurde, zu berücksichtigen und sollten folglich in den Analysen dort wiederzufinden sein. Eine Integration der betriebssystem-bedingten Kosten erfordert zunächst eine Analyse des Schedulings und dessen Kosten im Allgemeinen. Anschließend sind diejenigen Kosten zusätzlich zu betrachten, die aufgrund von Interferenzen konkurrierender Tasks außerhalb des direkten Einflusses des Betriebssystem auftreten werden (beispielhaft sei hier die Zerstörung von Teilen des Systemzustandes einer Task durch eine unterbrechende Task genannt).
Generell ist es für die Kostenanalyse der Taskwechselzeiten von entscheidender Bedeutung, welches Verfahren für ein Scheduling von konkurrierenden Tasks zum Einsatz kommt. Üblicherweise wird hier zwischen drei verschiedenen Typen unterschieden:

Nicht-preemptives Scheduling erlaubt keinerlei Unterbrechung einer laufenden Task durch andere Tasks. Dies hat zur Folge, dass Tasks, die zur Laufzeit einer anderen Task dispatched werden sollen, eine Blockierung erfahren, die im schlimmsten Fall der Laufzeit der aktiven Task entspricht. Dieses Verfahren eignet sich besonders für rein zeitgesteuerte Systeme, in denen das Auftreten von bereiten Tasks stringent einer zeitlichen Abfolge entspricht und keine Varianz in den Differenzen der Auftrittzeitpunkte der Tasks existieren (z.B. Time Triggered Architectures, vgl. [91]). Die Kosten des Schedulings fallen nur jeweils beim Dispatchen und Terminieren einer Task an.

Preemptives Scheduling hingegen stellt eine wesentlich höhere Flexibilität des Tasksystems zur Verfügung. Hierbei werden Tasks gemäß ihrer zeitlichen Wichtigkeit geordnet priorisiert. Höher priorisierte Tasks unterbrechen den Ablauf niedriger priorisierter, um ihre Berechnung möglichst zeitnah zu ihrem Bereit-Zeitpunkt ausführen zu können. Dieses Verfahren eignet sich im Besonderen in Systemen mit deutlichen Varianzen in den Aktivzeitpunkten, beispielsweise bei sporadisch auftretenden Ereignissen, die eine schnelle Berechnung erfordern. Die Kosten des Scheduling sind hier bei jedem ankommenden Ereignis zu betrachten, was mithin eine Erhöhung der Latenzzeit gegenüber nicht-preemptiven Schedules bedingt, aber auf der anderen Seite eine deutliche Flexibilisierung in den Reaktionszeiten ermöglicht.

Partiell preemptives Scheduling stellt eine Mischung aus den oben genannten Verfahren dar. Dabei wird der preemptive Charakter beibehalten, indem gezielt Punkte innerhalb eines Programmablaufes einer Task angegeben werden, an denen eine Unterbrechung erlaubt ist. Alle anderen Zeitpunkte verbieten eine Unterbrechung. Dieses Verfahren wird besonders dann eingesetzt, wenn die Kosten einer Unterbrechung aufgrund weitreichender Interferenzschäden durch andere Tasks nicht hinreichend genau analysiert werden können oder aber Größenordnungen erreichen, in denen der Vorteil der Flexibilität von Preemptionen sich ins Gegenteil verkehren. Die Kosten des Scheduling in diesem Verfahren gleichen denen im preemptiven Fall, da auch hier Preemptionen zulässig sind, deren Eintrittszeitpunkt allerdings beschränkt ist.

Preemptives und partiell preemptives Scheduling können auf zwei verschiedene Art und Weisen betrieben werden: In den sogenannten dynamischen Verfahren, wie beispielsweise EDF[104] oder Proportional Share Scheduling[174], werden Tasks von anderen Tasks gemäß einer zur Laufzeit bestimmten Strategie unterbrochen. Hingegen stellt das sogenannte statische Scheduling eine off-line, d.h. zur Systemerstellung generierte, Priorisierung der Tasks zur Verfügung, die bestimmt, welche Task von anderen Tasks unterbrochenen werden darf. Zwar ist das erstere Verfahren effektiver in der Ausnutzung der Prozessorauslastung, aber die zur Laufzeit zu bestimmende dynamische Priorisierung hat einen erhöhten Laufzeitbedarf des Schedulers zur Folge. Aus diesem Grund werden dynamische Verfahren in eingebetteten Systemen mit harten Echtzeitanforderungen selten eingesetzt und sollen im weiteren Verlauf dieser Arbeit nicht weiter betrachtet werden.
Diese Verfahren können auch auf einem Prozessorknoten gemischt betrieben werden, wie es z.B. im OSEKtime Protokoll (vgl. [134]) angeboten wird. Partiell preemptives und preemptives Scheduling unterscheiden sich bei der Analyse von Betriebssystemkosten nur in der Betrachtung von Interferenzkosten der Tasks untereinander[1]. Eine genauere Analyse der durch Interferenz hervorgerufenen Kosten erfolgt in späteren Kapiteln. Nicht-preemptives Scheduling kann in dieser Betrachtungsweise als Spezialfall des partiell preemptiven Vorgehens angesehen werden, folglich werden im

[1] Die durch nicht-preemptive Anteile induzierten Blockierungszeiten für höher priorisierte Tasks sollen hier nicht weiter betrachtet werden, sind aber natürlich ein wesentlicher Faktor bei der Berechnung von Antwortzeiten.

Kontext dieses Teiles der Arbeit nur die Kosten näher untersucht, die durch preemptives Scheduling verursacht werden.

Bereits in Teil I wurde die formale Charakterisierung einer Task unter anderem durch Priorität und einem — den Startwunsch einer Task anzeigenden — Ereignis definiert. Die Aufgabe des Betriebssystems ist es nun, auf ankommende Ereignisse so zu reagieren, dass diejenige Task zur Ausführung kommt, deren Priorität am höchsten ist und die bereit zur Ausführung ist. Hierzu besitzt eine Task in der einfachsten Version einen der folgenden Zustände:

Inaktiv Die Task ist nicht zur Ausführung bereit. Dieser Zustand wird verlassen, sobald das zur Task gehörige Aktivierungsreignis auftritt.

Bereit Das zur Task gehörige Ereignis ist aufgetreten, der Prozessor aber durch eine Task höherer Priorität blockiert. Dieser Zustand wird beim Freiwerden des Prozessors verlassen und auf *aktiv* gesetzt.

Aktiv Die Task wird ausgeführt. Eine Preemption durch eine höher priorisierten Task veranlasst einen Wechsel in den *bereit*-Zustand, eine Termination der Task einen Wechsel in den *inaktiv*-Zustand.

An dem Zustandsautomaten für diesen einfachsten Fall der Zustände einer Task in einem preemptiven System wird deutlich, dass sich die Kosten für das Scheduling in zwei getrennte Bereiche aufteilen lassen: dem Erkennen, dass ein Ereignis für eine Task vorliegt und dem eigentlichen Schedule, das der höchstpriorisierten Task aus der Menge an *bereiten* Tasks den Zuschlag erteilt und sie dispatched.
Diese zwei Klassen von Aktionen finden sich direkt in den Strukuren typischer Echtzeitbetriebssysteme wieder. Das Erkennen eines Ereignisses ist zumeist an einen Prozessor-Interrupt gebunden, der nach der Erkennung gegebenfalls den Scheduler des Betriebssystems aufruft. In parametriesierbaren RTOS, wie beispielsweise OSEK[133] ist diese Schnittstelle vom Ersteller einer Applikation zu spezifizieren (vgl. OSEK Implementation Language[136]). Quellen dieser Ereignisse sind entweder abgelaufene Timer auf den Prozessoren selbst oder externe Ereignisse, wie das Empfangen einer Nachricht, die als Startereignis für eine Task festgelegt wurde oder aber Änderungen an bestimmten zur Task gehörigen Sensoren. Die Abarbeitung dieser Ereignisinformationen erfolgt in sogenannten *Interrupt Service Routinen* (ISR), die ein zentraler Bestandteil jeder Echtzeitapplikation sind und umfassen üblicherweise sehr hardware-nahe Operationen[90].
Nach dem Erfassen eines Aktivierungsereignisses wird (normalerweise innerhalb einer ISR) der Scheduler des Echtzeitbetriebssystems aufgerufen, dem die neue bereite Task gemeldet wird. Dieser entscheidet anhand der Prioritäten aller bereiten Tasks, welche Task dispatched und ob die aktive Task unterbrochen wird. Diese Aktionen finden aber auch außerhalb einer ISR statt, z.B. sobald eine Task ihre Ausführung beendet hat. Für eine ausführlichere Darstellung der Abläufe sei hier auf die einschlägige Literatur sowie als Beispiel wiederum auf das OSEK[133] Modell verwiesen.

7.2 Laufzeitverhalten

Ausgehend von diesem Modell des Ablaufes innerhalb einer Echtzeitapplikation kann nun die Integration der Betriebssystemkosten bzgl. des Scheduling in die Berechnung der Antwortzeiten von Tasks erfolgen. Partiell ist dies bei der Berechnung von Kommunikationszeiten in verteilten Tasknetzwerken in Teil I dieser Arbeit bereits durch die Einführung der sogenannten Notification Tasks erfolgt[1], soll hier aber nun um die generellen Einflüsse des Schedulings erweitert werden.
Die Kosten sowohl einer ISR als auch des Schedulers selbst können mit Hilfe der im Bereich WCET-Analysen entwickelten Techniken analysiert werden und sollen im folgenden als konstant gegeben sein. Dazu sei c_{ISR} die maximale Laufzeit einer Interrupt Service Routine und c_{Sched} die maximale Laufzeit des Schedulers. Streng genommen sind diese Werte sowohl für ISRs als auch für die Schedulingkosten nicht als konstant anzusehen. ISRs bedienen unterschiedliche Peripherie oder interne Timer und können erheblich in ihrer Ausführungszeit differieren. Scheduler-Läufer hingegen unterscheiden sich stark in ihrem Zeitverhalten, abhängig davon, ob eine Task dispatched wird oder nicht. Letzteres liegt im wesentlichen an dem zu restaurierenden Systemzustand (dem sogenannten Frame) der zu dispatchenden und der evtl. zu unterbrechenden Task. ISRs können als zugehörig zu einer Task betrachtet werden, so dass hier abhängig vom Ereignis eine konstante Zeitspanne anzunehmen ist. Der Zeitverbrauch des Schedulers ist hingegen u.a. abhängig davon, ob dispatched wurde oder nicht, und dies ist in den Antwortzeitberechnungen ebenfalls leicht zu erkennen. Aufgrund dieser recht einfachen Erweiterungen zu einem konstanten Modell von unterbrechungsinduzierten Kosten soll im folgenden darauf verzichtet werden, das Schedulingverhalten in dieser feingranularen Betrachtungsweise zu verwenden. Stattdessen wird die Problematik anhand eines einfacheren, konstanten Modells erläutert und es werden Ansätze zur Steigerung der Performanz beschrieben.

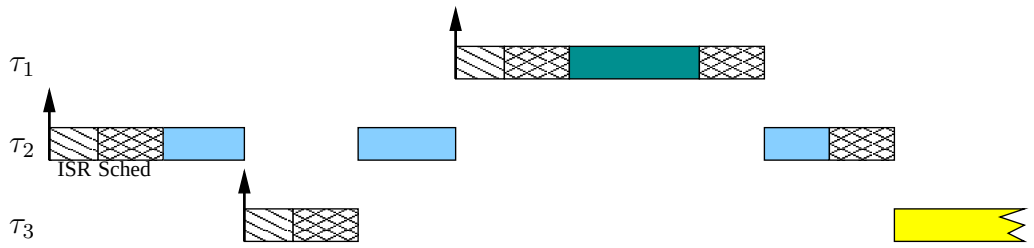


Abbildung 7.1: Durch Ereignisse induzierte Kosten des Betriebssystems. Die Pfeile entsprechen dem Auftreten des zur jeweiligen Task gehörenden Ereignisses. Die Tasks sind in absteigender Priorität nummeriert.

Die Prinzipien der entstehenden Kosten der Annahme von Ereignissen und den dazugehörigen Schedules sind in Abbildung 7.1 dargestellt. Es wird deutlich, dass neben den durch bereite, höher priorisierte Tasks entstehenden Kosten auch die

[1] Notification Tasks stellen in diesem Sinne bereits das Erkennen des Ereignisses durch eine ISR und das Starten des Schedulers dar, ohne die wirklichen Kosten des Schedulings zu betrachten.

Ereignisannahme von Tasks niedrigerer Priorität (im Beispiel aus Sicht von τ_2) zu berücksichtigen sind. Da es sich bei der Aufnahme eines Ereignissen üblicherweise um einen Interrupt handelt, der durch eine entsprechende ISR bearbeitet wird, wird diese in jedem Fall zur Ausführung kommen. Zwar gibt es in allen Echtzeit-Betriebssystemen die Möglichkeit, die Hardware-nahe Schicht der Interrupts in gewissen Graden zu blockieren, aber dies wird zumeist nur in sicherheitsrelevanten Fällen (z.B. kritische Bereiche) zur Anwendung kommen. Eine weitergehende Nutzung dieses Mechanismus würde letztlich dazu führen, dass das Schedulingverfahren dem partiell preemptiven Scheduling entspricht, welches hier nicht weiter betrachtet werden soll. Ob die zu dem auftretenden Ereignis gehörende Task zur Ausführung kommen soll, kann nur der Scheduler selbst entscheiden, da dies eine dynamische Eigenschaft ist, die von der aktuellen Menge an bereiten und aktiven Tasks abhängt. Zu diesem Zweck ist ein Aufruf des Schedulers notwendig, der im Fall höher priorisierter Tasks ein Dispatchen des Ereignisträgers durchführt und im anderen Fall den Prozessor wieder an die momentan aktive Task zurückgibt.
Verwendet man die aus Teil I bereits eingeführten Mechanismen zur Berechnung der Antwortzeit einer Task, so lassen sich die Kosten des Schedulings gemäß der Darstellung in Abbildung 7.1 einfach integrieren:

$$r_i = c_i + c_{ISR} + c_{Sched} + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_j + c_{ISR} + 2c_{Sched}\right) + \sum_{\tau_j \in lp(\tau_i)} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_{ISR} + c_{Sched}\right)$$

Dies ist wiederum eine Worst-Case-Betrachtung, die als zentrales Mittel der korrekten Überapproximation die sogenannte kritische Instanz (vgl. [104]) eines Tasksystems verwendet, d.h. zum Startzeitpunkt der zu betrachtenden Task empfangen alle anderen Tasks ebenfalls synchron ihr Aktivierungsereignis. Die Schedulingkosten für die zu analysierende Task τ_i treten nur einfach auf, da in der Antwortzeit von τ_i keine Notwendigkeit besteht, nachfolgende Scheduleraufrufe zu berücksichtigen. Stattdessen wird der durch die Termination folgende Scheduleraufruf in den Analysen anderer Tasks berücksichtigt. Die durch höher priorisierte Tasks induzierten Kosten sind in der herkömmlichen Weise integriert, wohingegen die Annahmekosten niedriger priorisierter Tasks nur durch $c_{ISR} + c_{Sched}$ ins Gewicht fallen ($lp(\tau_i)$ bezeichnet dabei die Menge aller Tasks, deren Priorität kleiner als die von τ_i ist).

Neben den direkt durch das Betriebssystem verursachten Latenzen müssen auch diejenigen Latenzen betrachtet werden, die aus einer Veränderung des Systemzustandes einer unterbrochenen Task durch eine unterbrechende Task hervorgerufen werden. Dies bezieht sich in der Regel auf Inhalte von Caches und Pipeline-Zuständen und ist in den üblichen Analysen zur Berechnung der Worst Case Execution Time von Tasks nicht enthalten. Gleichwohl spielen diese Latenzzeiten aber eine bedeutende Rolle, da die durch Interferenz induzierten Kosten durchaus in Größenordnung der WCET selbst kommen können. Veränderungen im Systemzustand durch Unterbrechungen bedeuten, dass die bisher als konstant angenommene WCET einer Task nun als nicht mehr konstant betrachtet werden kann. Dies soll zur Veranschaulichung in Abbildung 7.2 anhand des Beispieles Instruction Cache verdeutlicht werden. Eine

detailliertere Diskussion erfolgt in Kapitel 8.4.2, in dem auch die Integration in die Antwortzeitberechnung gezeigt wird.

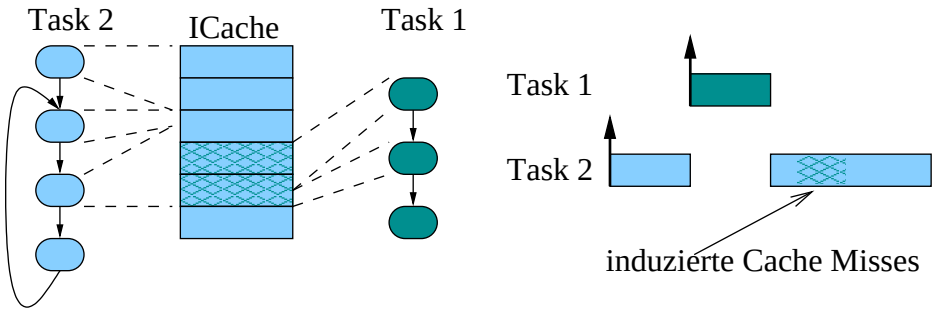


Abbildung 7.2: Durch Taskinterferenz erzeugte Kosten beim Zugriff auf Daten aus dem Instruction Cache eines Prozessors. Task 1 invalidiert Cache Lines, die Task 2 eingelagert hat.

Abbildung 7.2 zeigt die Interferenzeinflüsse einer unterbrechenden Task: Während der Ausführung von Task 1 füllt diese den Instruction Cache mit den Instruktionen ihrer Schleifenbehandlung. Jeder Durchlauf durch die Schleife nach dem ersten Durchlauf resultiert in einem Cache-Hit für die Instruktionen des Schleifenkörpers. Dieses wird so auch von den üblicherweise verwendeten Techniken zur Berechnung der WCET ermittelt. Unterbricht eine höher priorisierte Task 1 den in der Schleife verweilenden Task 2, und benutzt diese nun neu aktive Task 1 gleiche Teile des Instruction Caches, so führt dies unter Umständen zu einer Verdrängung dort bereits befindlicher Inhalte von Task 2. Nach Beendigung von Task 1 wird Task 2 erneut die Schleife durchlaufen und dabei, anders als in der vorhergesagten WCET ermittelt, nun zusätzlich einen Miss für jede verdrängte Line erzeugen. Dies bewirkt eine zusätzliche Latenz in der Antwortzeit von Task 2, die nur aus der Unterbrechung resultiert und nicht in der WCET-Analyse vorhergesagt werden kann. Gleiche Effekte sind auch bei der Betrachtung von Pipelines zu beobachten und müssen ebenso berücksichtigt werden. Beide Faktoren — sowohl Cache als auch Pipeline — können ganz erhebliche Effekte haben, da bei komplexen Prozessorarchitekturen unter Umständen ein Miss ein völlig anderes internes Instruction-Scheduling (Zuteilung zu funktionalen Einheiten) hervorruft, als ein Hit. Latenzzeiten, die sich über viele weitere Schleifendurchläufe aufschaukeln, können die Folge davon sein.

7.3 Schlussfolgernde Betrachtung

Um abschätzen zu können, welcher Anteil an Auslastung eines Systems dem Betriebssystem zufällt, müssen zunächst die typischen Latenzzeiten von Interrupt Service Routinen und Scheduleraufrufen ermittelt werden. Bei der Sichtung handelsüblicher Echtzeitbetriebssysteme fällt auf, dass zumeist nur Aussagen über die mittlere Reaktionszeit der besagten Teile des Betriebssystem gemacht werden. Dies ist zwar im Sinne einer präzisen Antwortzeitberechnung unzulänglich, genügt aber den meisten Anwendern, da heutzutage kaum so präzise Analysen gemacht werden, sondern

im wesentlichen auf Utilisation und Messungen zurückgegriffen wird, um die Feasibility eines Echtzeitsystems zu "zeigen" (mit den damit inhärent einhergehenden negativen Konsequenzen bzgl. der Sicherheit). Empirische Messreihen sind hier der einzige Weg, konkrete Daten bzgl. der WCET des Schedulers und zugehöriger ISRs zu gewinnen. In diesem Umfeld hat es eine ganze Reihe von Arbeiten gegebenen, von denen hier beispielhaft [195, 79, 208] genannt werden sollen. Allen Arbeiten gemeinsam ist die Messung von Reaktionszeiten des Betriebssystems, jedoch für Systeme unterschiedlicher Hersteller. So konnten in [79] Latenzzeiten von ca. 5000 bis 6600 Prozessorzyklen für Interrupt Service Routinen und ca. 700 Zyklen für Scheduleraufrufe auf den Betriebssystemen RTLinux[60] sowie VxWorks[198] gemessen werden. In [208] wurde ein hand-optimierter OSEK-Kernel entwickelt, dessen Reaktionszeit mit 780 Prozessorzyklen für ISRs bzw. 900 Prozessorzyklen für den Scheduler angegeben sind. In [195] wurden ebenfalls Spannweiten von 3000 bis 9000 Zyklen Latenz gemessen. Im unteren Leistungssegment wurden beispielsweise in [108] für das Betriebssystem POSIX[76] auf einem Intel Pentium III mit 1.1 GHz Taktfrequenz Schedulingkosten von $0.42\mu s$ gemessen. Dieser Wert enthält sowohl die ISR-Kosten als auch die Taskwechselkosten und ergibt in etwa 40000 Zyklen.

Einen Eindruck von der Größenordnung der Betriebssystemkosten in realen Applikationen soll das folgende Beispiel liefern. Zur Abschätzung des Schedulingoverhead wird dabei von der Utilisation des Prozessors ausgegangen und hierbei nur der Einfluss des Betriebssystem nach oben abgeschätzt. Geht man davon aus, dass jede Task eines Systems bei jeder Invokation einen Zeitverbrauch von $c_{ISR} + 2c_{Sched}$ in Schedulingkosten investiert, so ist der durch das Scheduling verbrauchte Anteil an der Auslastung des Prozessors gegeben durch:

$$U_{sched} = \sum_{\tau_i} \frac{c_{ISR} + 2c_{Sched}}{t_i} \tag{7.1}$$

Das Beispielsystem bestehe aus einem in den Anwendungsdomänen für Echtzeitapplikationen (z.B. Automobilindustrie, Luftfahrtindustrie, etc.) durchaus üblichen PowerPC MPC555[123] mit einer Taktung von 40 MHz. Als Echtzeitbetriebssystem soll das optimierte OSEK aus [208] zum Einsatz kommen, das für diesen Prozessortyp somit Latenzzeiten von $19\mu s$ für Interrupt Service Routinen und $22\mu s$ für Scheduleraufrufe liefert. Auf diesem System soll das Tasksystem (ein Steuergerät aus der Automobilindustrie) aus Tabelle 7.1 laufen. Die Priorisierung der Tasks er-

Anzahl Tasks	Periode
7	1 ms
6	5 ms
6	10 ms
3	20 ms
4	50 ms

Tabelle 7.1: Beispielhaftes Tasksystem für die Abschätzung von Schedulingoverhead. Alle Tasks sind nach dem Rate Monotonic Prinzip priorisiert.

folgt gemäß der Rate Monotonic Strategie, d.h. es existiert eine theoretische Grenze von 69% der Utilisation bis zu der man allein aufgrund der Utilisation beurteilen kann[104], ob dieses System alle Echtzeitbedingungen (hier demzufolge $\Delta_\tau(\tau_i) = t_i$ als Bedingung für die Deadlines $\Delta_\tau(\tau_i)$) einhalten kann. Untersucht man in diesem Beispiel nur die Belastung des Prozessors durch Betriebssystemkosten entsprechend der Formel (7.1), so ergibt sich eine Utilisation von 57%. D.h. 82% von der bis zur theoretisch sicheren Rate Monotonic Grenze zur Verfügung stehenden Kapazität des Prozessors wird in diesem Beispiel für die Kosten des Scheduling verbraucht, nur ein 18%iger Anteil der zur Verfügung stehenden Kapazität (also eine maximale Utilisation von 12%) kann von den eigentlichen Berechnungen der Tasks eingenommen werden. Dies ist ein klares Missverhältnis und macht einen Einsatz eines derartigen Systems sehr teuer. Auf der anderen Seite sind in diesen Berechnungen die indirekten Kosten von Interferenzen, ausgelöst durch Preemptionen, noch gar nicht mit aufgeführt, so dass in diesem Fall die mögliche Kapazität für die Applikationstasks noch weiter sinken wird. Eine detailliertere Darstellung der Interferenzeinflüsse erfolgt in Kapitel 8.4.2, in dem der Overhead quantifiziert wird, den die einzelnen Tasks bzgl. ihrer Antwortzeit durch Interferenzen auf dem Instruction Cache verlieren. Es zeigt sich, dass dies durchaus in der Größenordnung der eigentlichen WCET einer Task liegen kann, ihre Ausführungszeit also im schlimmsten Fall mehr als verdoppelt wird[86].

Zusammenfassend bedeutet dies, dass, um die Reaktionszeit von Tasks signifikant zu verbessern, im wesentlichen zwei Dinge getan werden müssen: Zum einen sollte der Aufwand, den ISR und Scheduler treiben müssen, deutlich reduziert werden, was in diesem Kontext nur heißen kann, dass deren Laufzeit um ein Vielfaches reduziert werden müsste. Zum anderen sollte es disjunkte Zustandsräume der Tasks geben, um Interferenzen weitgehend zu vermeiden. Beide Stoßrichtungen können nur dann signifikant vorangetrieben werden, wenn man dazu übergeht im klassischen Sinne Hardware/Software-Codesign zu betreiben, also die Disjunktheit und die zeitkritischen Anteile in direkte Hardware-Lösungen zu integrieren. Die handoptimierte und in Assembler geschriebene Implementierung des OSEK-Schedulers in [208] zeigt, dass software-technische Optimierungen keinen nennenswerten Gewinn mehr bringen können. Leistungsfähigere Prozessoren scheiden hingegen aufgrund der engen Kostenmargen, denen eingebettete Systeme unterliegen, aus. Eine Verlegung zeitkritischer Teile des Scheduling in Hardware bietet sich folglich in diesen Fällen an, da die entstehenden Kosten inhärent allen Systemen zugrunde liegen, die als Echtzeitapplikation definiert werden. Man schafft also eine für Echtzeitsysteme propreitäre Infrastruktur mit dem Ziel der gesteigerten Performanz[1].
Dieser Weg soll in den folgenden Kapiteln beschritten werden. Basis der Herangehensweise sind sogenannte multithreaded Prozessoren, die eben jene oben genannten Eigenschaften anbieten und damit gewinnbringend als Prozessorknoten in eingebetteten Echtzeitsystemen eingesetzt werden können.

[1] Hier ist nochmals deutlich zwischen Echtzeit-Performanz und allgemein üblicher Performanzaussagen zu unterscheiden. Im Kontext von Echtzeitsystemen wird die Performanz auch wesentlich von der Vorhersagbarkeit bestimmt, im allgemeinen Fall interessiert nur der mittlere Durchsatz.

Kapitel 8

Multithreading

8.1 Einleitung

Wendet man Hardware/Software-Codesign auf die im vorangegangenen Kapitel beschriebenen Problematiken an, so ist vor allem das Abwickeln eines Scheduleraufrufes und die Annahme von Ereignissen durch Interrupt Service Routinen zu beschleunigen. Konsequenterweise bedeutet dies, dass genau jene Aktionen in Hardware implementiert werden sollten, ohne jedoch die Flexibilität einer Software-Lösung zu verlieren. Aus diesem Grund wird in dieser Arbeit ein Verfahren vorgeschlagen, dass beide Varianten erlaubt, also eine Kombination von Hardware-gesteuerten Tasks mit klassisch Betriebssystem-gesteuerten Tasks. Erreicht wird dies durch eine Technik, die ursprünglich aus dem Bereich der Parallelrechner stammt, aber auch zunehmend im Consumer- und Embedded-Systems-Segment (siehe z.B. der TriCore 2[131]) Verwendung findet. Diese sogenannte *Multithreading* Technik[94] ist entwickelt worden, um die teilweise mehrere hundert Zyklen dauernden Remote-Zugriffe (Speicherzugriffe in Shared Memory-Systemen, die über ein komplexes Verbindungsnetzwerk abgewickelt werden müssen) zu verstecken. Speicherzugriffe dieser Art blockieren den ausführenden Prozessorknoten potentiell bis zu ihrem Abschluss. Gerade in verteilten Applikationen, die in der Klasse der Parallelrechner eine weitreichende Bedeutung haben, sind diese Art der Zugriffe sehr häufig zu finden und zerstören den Performanzgewinn, der durch die massive Parallelität eigentlich erreicht werden sollte. Um dies zu verhindern, wird davon ausgegangen, dass pro Prozessorknoten mehrere Teilprozesse (sogenannte Threads) quasi parallel ablaufen. Führt einer dieser Threads einen Speicherzugriff mit hoher Latenzzeit aus, schaltet das System automatisch auf einen anderen bereiten Thread um und versteckt die Zugriffszeit des Speicherzugriffs durch die Ausführung dieses anderen Threads. Um dieses Umschalten aus Sicht der mittleren Performanz gewinnbringend einzusetzen, muss es einer bestimmten Mindestgeschwindigkeit genügen. Dies bedeutet, dass es nicht ausreicht, per Software einen Threadwechsel zu initiieren, da dieser zu viel Zeit verbrauchen würde. Zusätzlich muss gewährleistet sein, dass der Working Set des suspendierten Threads vor Zerstörung gesichert ist. Erreicht werden diese Randbedingungen an einen Threadwechsel durch das Vorhalten einer Menge von physikalisch disjunkten Behälter für Threads, den sogenannten *Kontexten*. Grundsätzliche Idee dahinter ist, den Working Set, also Registerinhalte und Pipeline-Zustände, womöglich auch

Cache-Inhalte, parallel für mehrere Threads auf dem Prozessor physikalisch vorzuhalten und bei Bedarf durch einen schnellen Wechsel des Kontextes einen anderen Thread aktiv zu schalten. Dies bedarf auf Seiten der Kontexte eine massive Unterstützung durch die Hardware. Insbesondere wird hierfür üblicherweise das Registerfile und Teile der Pipeline, sowie in manchen Architekturen auch Teile der ersten Stufe der Cache-Hierarchie sooft dupliziert, wie Kontexte auf dem Prozessor angeboten werden. Bezüglich der Anforderungen an eine Kontextwechsel werden unterschiedliche Strategien in der Literatur diskutiert:

Block Multithreading wird eine Strategie genannt, die Kontextwechsel nur dann ausführt, wenn der aktive Thread einen Remote-Zugriff durchführt und eine lange Latenzzeit zu erwarten ist. Zu diesem Zweck muss die Speicherhierarchie einen Kontextwechsel erzwingen können, da an dieser Stelle erst das Wissen um den Zielpunkt des Speicherzugriffes verfügbar ist. Untersuchungen haben ergeben, dass auf Block Multithreading basierende Techniken mit Kontextwechselzeiten von bis zu 10 Takten immer noch gewinnbringend in Parallelrechnern eingesetzt werden können[194].

Interleaved Multithreading basiert auf einem sehr fein granularen Kontextwechsel [7], üblicherweise bei jedem Speicherzugriff oder sogar nach jeder Instruktion. Die Effektivität dieses Mechanismus übersteigt den des Block Multithreading, wird jedoch durch einen erhöhten Hardwareaufwand bezahlt, da extrem schnelle Kontextwechsel vollzogen werden müssen, um nicht den Prozessor im Ganzen bei jeder Instruktion auszubremsen. Der Kontextwechsel erfolgt damit zwangsläufig nach einem internen Schema.

Simultanes Multithreading ist eine Erweiterung des Interleaved Multithreading in Superskalaren Prozessoren[190]. Dabei wird nicht nur fein granular zwischen Kontexten umhergeschaltet, sondern während der Ausführung werden zusätzlich Instruktionen verschiedener bereiter Threads auf die freien funktionalen Einheiten der Superskalaren Architektur verteilt, also eine vollständig gemischte Ausführung mehrerer Threads unter massiver Ausnutzung der zur Verfügung stehenden Parallelität der Instruction Set Architecture.

Das Vorhalten mehrerer Containter für Threads entspricht genau der Vorstellung, den Systemzustand von Tasks in Echtzeitsystemen disjunkt von anderen Tasks zu halten, insbesondere wenn multithreaded Architekturen Teile der Cache-Hierarchie ebenfalls als Teil eines Kontextes definieren (vgl. Beispiel zur Interferenz aus dem vorherigen Kapitel in Abbildung 7.2). Zusätzlich entspricht ein schneller Kontextwechsel den geforderten Einsparungen bei der Latenzzeit eines Scheduleraufrufes.
Bei der Betrachtung der verschiedenen Multithreading-Techniken ist nun diejenige auszuwählen, die im Bereich der Echtzeitsysteme am vielversprechendsten erscheint. Interleaved Multithreading widerspricht zwar der Anschauung typischer Ablaufstrukturen in Echtzeitsystemen (also monolithische Ausführung bis eine Preemption auftritt), allerdings wurde dieses Verfahren für ein Proportional Share basiertes Scheduling[174] vorgeschlagen[93]. Ein wesentlicher Nachteil ist, dass mit diesem Verfahren tatsächlich nur soviel Tasks auf einem Prozessor parallel zur Ausführung

kommen dürfen, wie der Prozessor Kontexte besitzt. Diese Zahl ist üblicherweise aufgrund der sehr großen dafür notwendigen Hardware-Resourcen auf 4 bis 8 beschränkt. Ein zweites Problem macht sich bemerkbar, wenn — anders als in [93] — ein Cache Teil des Systems ist und gleichzeitig mehrere Tasks ausstehende Speicherzugriffe auf diesen Cache haben. In diesem speziellen Fall wird der Cache zum Flaschenhals und verdirbt die Performanz. Besonders kritisch daran ist zugleich, dass diese Stelle im Programmablauf nur sehr schwer mit den Techniken zur Vorhersage der WCET zu erfassen ist, so dass eine pessimistische Approximation des Zeitverhaltens angenommen werden muss.
Für Simultanes Multithreading gelten die gleichen Einschränkungen wie für das Interleaved Multithreading. Zudem ist der Performanzgewinn durch die echte Parallelausführung mit Hilfe der Methoden zur WCET-Analyse kaum vorhersagbar, da während des Ablaufes einer Task überhaupt nicht statisch ermittelbar ist, welche andere Task gleichzeitig bereit ist. Dies gilt erst recht in Tasksystemen die sporadische oder Jitter-behaftete Ankunftzeiten der Startereignisse besitzen. Daher ist auch diese Technik weniger geeignet, in Echtzeitsystemen den Anforderungen zu genügen, die in Kapitel 7 aufgestellt wurden.
Verbleibt als Lösung folglich das Block Multithreading, welches aufgrund der Ausführung der Tasks in Blockformen und der externen Anforderungen an einen Kontextwechsel sehr stark an das typische Ausführungsverhalten von Tasks in Echtzeitsystemen erinnert. Durch die Blockstruktur der Ausführung wird zudem der Nachteil der feingranularen Multithreadingtechniken bzgl. des Flaschenhalses Cache vermieden. Ausführungen von Tasks sind bei diesem Verfahren völlig transparent; es ist für eine Applikation nicht ersichtlich und auch in der technischen Ausführung gibt es kein Mischen der Instruktionen aufgrund fehlender echter Parallelität der Tasks untereinander (anders als beispielsweise beim Simultanen Multithreading). Ausführungszeitvorhersagen sind folglich identisch zu denen auf Prozessoren ohne Multithreading.
Angepasst werden müssen lediglich die Verfahren für Taskwechsel. In multithreaded Prozessoren, die in Parallelrechnern eingesetzt werden, bestehen typischerweise keine stringenten Anforderungen an die Reihenfolge der Threadausführungen. Standard-Schedules wie Round Robin, die eine gewisse Fairness garantieren, sind deshalb weit verbreitet. Ein Einsatz in Echtzeitsystemen ist nur dann möglich, wenn stattdessen die Kontextwechselanforderung von einem Echtzeitscheduler gesteuert wird. Dieser muss sowohl die Ereignisse aufnehmen (also ISRs ersetzen), als auch die Taskliste mit ihren jeweiligen Zuständen verwalten und bei Bedarf Tasks suspendieren bzw. dispatchen. Der Prozessor muss mit einer entsprechenden Schnittstelle ausgerüstet werden. Passen Tasks oder Ereignisaufnahme nicht in das Schema der Hardwareimplementierung oder ist die Taskanzahl größer als die Anzahl der Kontexte, so kann ein ausgezeichneter Kontext dazu verwendet werden, die überzähligen oder unpassenden Tasks aufzunehmen. Dazu wird ein Echtzeitbetriebssystem in diesen Kontext mit aufgenommen. Somit können auch gemischte Varianten von Applikationen implementiert werden, die es ermöglichen, besonders zeitkritische oder hochfrequente und damit Utilisationsbelastende (bzgl. der Schedulingkosten) Tasks mit dem Geschwindigkeitsvorteil auszustatten und in einen seperaten Kontext zu legen.
Als Beispielarchitektur soll hier die MSparc[120] betrachtet werden. Dieser auf

dem SPARC-Standard[75] basierende Prozessor ist mit Block Multithreading ausgestattet und bietet die Schnittstelle zu einem Echtzeitscheduler an, so dass die oben skizzierten und im Verlauf dieses Teils noch genauer zu analysierenden Vorteile bzgl. Reaktionszeit und besserer Vorhersagbarkeit benutzt werden können[119].

8.2 MSparc

Die MSparc ist eine am Department für Informatik der Universität Oldenburg unter maßgeblicher Beteiligung des Autors dieser Arbeit entstandene prototypische Implementierung eines RISC-Prozessors mit einem Block-Multithreading Ansatz. Der Prozessor wurde in einer $0,5\mu m$ Standardzellen-Bibliothek gefertigt und bietet damit die tiefgehenden Einblicke in die Konsequenzen von Architekturentscheidungen im Bereich des Multithreading, wie sie hier im folgenden zur Bewertung der vorgestellten Ansätze notwendig sind. Zudem wurde dort erstmalig die Kopplung zwischen einem in Hardware implementierten Echtzeitbetriebssystem und einem multithreaded Prozessor geschaffen[118]. In den folgenden Abschnitten sollen diejenigen Merkmale der MSparc beschrieben werden, die im Kontext dieser Arbeit eine wesentliche Rolle spielen. Für alle weitergehenden Informationen sowie technische Daten sei auf die ausführliche Dokumentation[114] verwiesen.

8.2.1 Generelle Architekturmerkmale

Die MSparc basiert auf dem SPARC Standard der Version 8[75] und verfügt über einen 8KByte großen, 2fach assoziativen onchip Instruction Cache, 8 Register Windows und eine 5-stufige Integer-Pipeline. Die gewählte Technik des Block-Multithreading ist für Applikationsprogramme vollkommen transparent, so dass Standardcompiler für SPARC-Architekturen verwendet werden können. Lediglich der Befehlssatz des Supervisor-Modus wurde um Kontextmangementbefehle erweitert und stellt damit dem Betriebssystem eine Schnittstelle zum Multithreading zur Verfügung. Insgesamt stellt die MSparc 4 Hardware-unterstützte Kontexte zur Verfügung, die als Container für Applikations-Tasks dienen können. Abbildung 8.1 gibt einen schematischen Überblick über die Architektur des Datenpfades.

Für das Block-Multithreading wurde der Instruction Cache statisch in 4 disjunkte Bereiche partitioniert. Zusätzlich ist das RegisterFile vierfach vorhanden. Alle übrigen Teile der Pipeline sind in einfacher Ausfertigung in den Datenpfad integriert, so dass bei einem Kontextwechsel das Leerlaufen der Pipeline als zusätzliche Kosten berücksichtigt werden müssen. Das Umschalten selbst ist innerhalb eines Taktes realisiert, so dass im schlimmsten Fall 5 Taktzyklen für einen Kontextwechsel berechnet werden müssen.
Effektive Initialisierungs- und Umschaltmechanismen sind dem Betriebssystem durch eine Entkopplung der Instruktions- von der Datensicht möglich, d.h. der Instruktionsstrom kann in einem anderen Kontext ablaufen als der Datenstrom. Mit diesem Ansatz gelingt eine effiziente Parameterübergabe zwischen Kontexten (unter zuhilfenahme der zusätzlichen Hilfsregister).

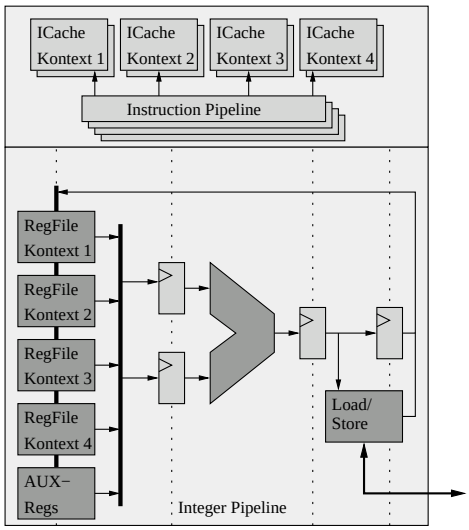


Abbildung 8.1: Übersicht über den Datenpfad der MSPARC. Die zusätzlichen Hilfsregister `AUX-Regs` dienen der Kommunikation zwischen den Kontexten auf Betriebssystem-Ebene.

Die Verwaltung der vier Kontexte unterliegt zum einen einem speziellen Statusregister (CSR) und zum anderen der Instruktionsadressen-Pipeline. Letztere ist verantwortlich für das korrekte Umschalten des Instruktionsstromes von einem Kontext in einen anderen und ist damit ebenfalls vierfach implementiert. Da die MSPARC sowohl in Echtzeitsystemen als auch in Multiprozessorsystemen eingesetzt werden kann, verfügt sie für den ersteren Fall über einen speziellen Mechanismus, den sogenannten *Embedded Control Mode* (ECM). Im letzteren Fall bestimmt, wie eingangs beschrieben, die Speicherhierarchie aufgrund von Load/Store-Befehlen, wann ein Kontextwechsel vorzunehmen ist. Die Auswahl des daraufhin zu aktivierenden Kontextes wird durch einen Round-Robin-Mechanismus über die gültigen Kontexte vorgenommen.

8.2.2 Embedded Control Mode

Der ECM bietet die Möglichkeit, einen externen Hardware-Scheduler anzuschließen, der die Kontrolle über die Kontexte übernimmt. Dabei wird nicht nur die Signalisierung eines Kontextwechsels übernommen, sondern auch der interne Status der Kontexte muss mit überwacht und beeinflusst werden können. Zu diesem Zweck existiert eine Schnittstelle, über die der Scheduler Zugriff auf die Kontexte hat:

Dispatch signalisiert dem Prozessor sowohl, dass ein Kontextwechsel erfolgen soll, als auch welcher Kontext angestoßen werden soll.

Activate signalisiert dem Prozessor, welche Kontexte neben dem angestoßenen noch bereit zur Ausführung sind (gesetztes Bereit-Flag). Dies kann für ein internes Round-Robin-Scheduling benutzt werden, wenn der externe Scheduler nicht eingreifen möchte, beispielsweise bei der kombinierten Verwendung von Echtzeit-Tasks und solchen ohne Echtzeitbedingungen.

Terminate signalisiert dem externen Scheduler, welche der auf bereit gesetzten Kontexte ihre Berechnung abgeschlossen haben und terminiert sind.

Die Ansteuerung und Architektur des externen Schedulers ist im nachfolgenden Abschnitt beschrieben. Die beschriebene Schnittstelle stellt einen Mechanismus zur Verfügung, der sowohl auf externe Ereignisse reagieren kann als auch auf interne, wie die Terminierung einer Task. Zu diesem Zweck ist der Befehlssatz der MSPARC im ECM um einen Terminierungsbefehl erweitert worden, der zum einen dafür sorgt, dass die Terminate-Schnittstelle entsprechend gesetzt wird und der zum anderen den gesamten Prozessor stilllegt, wenn nach einer Terminierung kein weiterer Kontext sein Bereit-Flag gesetzt hat. Abbildung 8.2 macht die unterschiedlichen Modi der Kontexte und ihre Interaktion mit dem externen Scheduler deutlich.

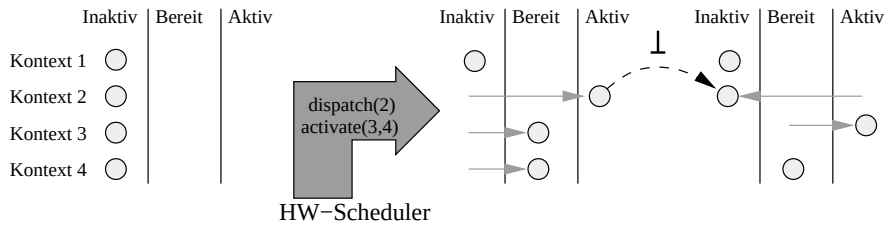


Abbildung 8.2: Die Verhaltensweisen im Embedded Control Mode der MSPARC. Im Initialfall ist der Prozessor ebenfalls inaktiv. Das ⊥-Symbol deutet die Terminierung einer Task ohne Einfluss des externen Schedulers an, das ein internes Umschalten zur Folge hat.

Insgesamt folgt damit eine Verlagerung der generellen Systemkontrolle vom Prozessor hin zum externen Scheduler. Der Hardware-Scheduler selbst ist nicht Teil der MSPARC, sondern wird, wie zum Beispiel im EVENTS-Projekt[107] in einem externen FPGA untergebracht[38] und übernimmt von dort die notwendigen Aufgaben.

8.2.3 Hardware-basierter Echtzeitscheduler

Ausgehend von der im letzten Abschnitt beschriebenen Schnittstelle des MSPARC-Prozessors soll hier nun eine Implementierung eines Echtzeitschedulers erarbeitet werden. Grundlage der Argumentation wird dabei eine ausschließliche Verwendung von Interrupts zur Taskaktivierung sein. Diese Interrupts werden dabei im Falle von Ereignis-basierten Tasks durch externe Events getriggert oder aber — im Falle von Zeit-basierten Tasks — durch einen Timer ausgelöst. Aufgrund des Fehlens von Timern in der MSPARC-Implementierung finden sich damit alle Task-initiierenden Vorgänge an der Interrupt-Schnittstelle des Prozessors wieder. Dieser wird durch entsprechend vorzuhaltende Interrupt Service Routinen die jeweiligen Ereignisse abgreifen und den Scheduler des Echtzeitbetriebssystems aufrufen, bzw. werden Teile hiervon in Hardware für eine deutlich beschleunigte Abwicklung sorgen. Maßgebliche Idee für den Einsatz des Multithreading in Echtzeit-Systemen ist dabei eine linearisierte Zuordnung der Tasks zu den Kontexten des Prozessors: Die $n-1$ höchst priorisierten Tasks werden jeweils einem einzelnen Kontext der MSPARC zugeordnet,

alle übrigen Tasks werden im letzten verblieben Kontext, dem sogenannten Sammelkontext, gesammelt. Letzterer verfügt über ein Echtzeitbetriebssystem, welches in Software auf herkömmliche Art und Weise implementiert ist. Es wird sich in den folgenden Abschnitten zeigen, dass diese Art der Taskzuordnung nahezu optimal ist und gleichzeitig die Implementierung eines Hardware-Schedulers drastisch vereinfacht.
Eine weitere wesentliche Eigenschaft ist die Rekonfigurierbarkeit des Schedulers, so dass ohne Änderungen der Hardware, vom Betriebssystem ausgehend, der Scheduler applikationsspezifisch eingestellt werden kann. Abbildung 8.3 gibt einen Überblick über die Architektur des Schedulers für die ECM-Variante des SPARC-Prozessors.

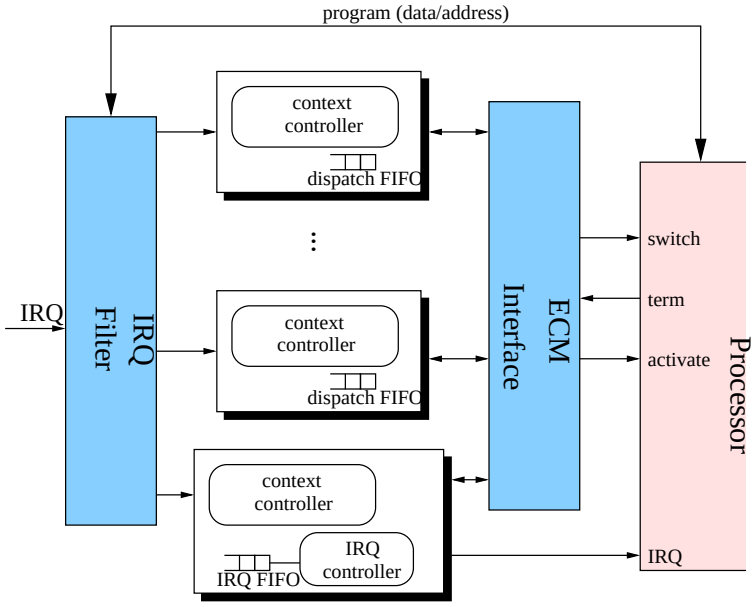


Abbildung 8.3: Implementierung des Hardware-basierten Echtzeitschedulers für die MSparc inklusive Interrupt-Filter und Interrupt FIFOs.

Da alle Task-initiierenden Aktionen über Interrupts signalisiert werden, ist zunächst ein Interrupt-Filter installiert, der die eingehenden Interrupt-Level (255 unterschiedliche Interrupts stehen bei einem SPARC-Prozessor zur Verfügung) auf die jeweils zum Interrupt gehörenden Warteschlangen der Kontexte der MSparc verteilt. Die Zuordnung der Interrupts zu den Warteschlangen ist vom Prozessor aus programmierbar. Hierzu wird auf dem Alternate Space Identifier (vgl. [75]) beruhendes Memory-mapped IO verwendet, wie es beispielsweise auch bei MMU-Programmierungen angewendet wird. Dieses Vorgehen erlaubt es, den Scheduler applikationsspezifisch zu konfigurieren, ohne die Hardware erneut erzeugen zu müssen. Da den $n-1$ Kontexte nur jeweils eine Task zugeordnet wird, entspricht dies auch genau einem Interrupt. Alle Übrigen werden an den Sammelkontext (in Abbildung 8.3 der unterste Kontextcontroller) weitergeleitet. Der IRQ-Filter erzeugt nach Weiterleitung des Interrupts ein entsprechendes Acknowledge für die Interruptauslösende Umgebung und signalisiert damit die Annahme. Dies wird verhindert, sollte der Interrupt auf-

grund der Fülle der Warteschlangen nicht weitergeleitet werden können[1].
Die Kontextcontroller nehmen die Interrupts an und tragen sie in ihre lokalen Wartschlangen ein. Gleichzeitig melden sie dem ECM-Interface das Vorliegen eines Interrupts im Falle der Sammeltasks oder einer Taskinitiierung im Falle der Einzeltasks. Da der Sammeltasks alle Interrupts aufnimmt, die nicht den $n-1$ höchst priorisierten Tasks zugeordnet sind, muss diese nicht unbedingt auch eine Taskinitiierung einleiten. Generell leitet der Sammelkontext Interrupts lediglich an die MSparc-Interrupt-Schnittstelle weiter, wohingegen die Kontexte 1 bis $n-1$ den Interrupt nicht weiterleiten und stattdessen die ECM-Schnittstelle nutzen und sofort eine Dispatch-Aktion durchführen. Aus diesem Grund wird auf den Warteschlangen auch keine Sortierung nach Prioritäten benötigt, da die Interrupts in der Reihenfolge weitergeleitet werden, in der sie von der Umgebung ausgelöst werden (FIFO-Prinzip). Dies gilt auch für die Warteschlangen in den Kontextcontrollern für die Einzelkontexte: Sukzessive Interrupts zur Initiierung ein und derselben Task müssen in ihrer Reihenfolge zwingend erhalten bleiben.
Das ECM-Interface führt schließlich den eigentlichen Schedule aus. Es wählt aus den eingehenden Anfragen der Kontextcontroller denjenigen Kontext mit der höchsten Priorität aus, vergleicht diese mit der des aktuell auf der MSparc laufenden Kontexts und initiiert ggf. einen Kontextwechsel über die ECM-Schnittstelle. Diese Aktion wird auch ausgeführt, wenn die aktuell ablaufende Task auf dem Prozessor ihren Kontext terminiert. Liegen keine Anfragen aus den $n-1$ höchst priorisierten Kontexten vor, wird im Falle einer Terminierung der interne Kontextwechsel-Mechanismus der MSparc genutzt. Zu diesem Zweck wird der Sammelkontext immer als aktiv markiert und erlaubt es somit dem internen Kontextwechsel, den Sammelkontext zu aktivieren.

Auf diese Art und Weise ist ein prioritäten-gesteuertes, preemptives Scheduling für alle Tasks eines Systems sichergestellt, wobei dies für Tasks des Sammelkontexts durch ein zusätzlich vorzuhaltendes Echtzeitbetriebssystem nur im Sammelkontext sichergestellt werden muss. Auf weitere Implementierungsdetails der einzelnen Blöcke der in Abbildung 8.3 gezeigten Architektur soll hier verzichtet werden; der interessierte Leser sei auf [44] verwiesen, in der die Implementierung und Umsetzung auf ein FPGA beschrieben ist. Es sei aber noch darauf hingewiesen, dass durch die nur wenige Zyklen dauernde Rekonfiguration des Schedulers dieses System auch ausgezeichnet in Systemen mit Moduswechseln[2] eingesetzt werden kann und den durch den Moduswechsel induzierten zeitlichen Overhead für eine Änderung des Schedulings drastisch verkürzen kann.

Die Größe der Warteschlangen der einzelnen Kontextcontroller ist applikationsspezi-

[1] Näheres zur Größe der einzelnen Warteschlangen und ihr Einfluss auf die Abweisungshäufigkeit von Interrupts siehe unten.

[2] Moduswechsel bezeichnen Vorgänge während des laufenden Betriebs eines eingebetteten Systems, in denen Teiles des Tasksets ausgeblendet, verändert oder neu aufgesetzt werden. Ein Beispiel für ein derartiges Szenario ist die Software im Bereich der Luftfahrt: Ein Flugzeug hat während eines Landeanflugs komplett andere Tasksysteme abzuarbeiten als während des Standardfluges auf festgelegter Höhe. Da diese Aufgaben vollkommen disjunkt sind, werden üblicherweise Steuergeräte für beide Aufgaben gleichzeitig eingesetzt.

fisch zu wählen. Für die $n-1$ Einzelkontexte kann sie einfach berechnet werden und ist abhängig von der Deadline einer Task: Ist diese kleiner oder gleich der Periode der Task, kann — unter der Annahme eines Systems das Feasible ist — die Größe der Warteschlange auf 1 begrenzt werden: Eine weitere Taskinitiierung bevor die vorhergehende Invokation beendet ist, würde eine Verletzung der Deadline bedeuten, und das System wäre nicht feasible. Liegt die Deadline oberhalb der Zeitspanne einer erneuten Aktivierung der Task, so ist die Größe der Warteschlange abhängig vom Verhältnis der Deadline zur Periode:

$$N_c = \left\lceil \frac{\Delta_\tau(\tau_i)}{t_i} \right\rceil \text{ für Kontext } c \in \{1, \ldots, n-1\}$$

Für Bursty-Tasksysteme hängt die Länge der Warteschlange von der Anzahl an Taskinvokationen innerhalb einer Periode ab. Da diese Anzahl auch in der Scheduling-Analyse derartiger Systeme gegeben sein muss, ist sie bekannt und kann einfach verwendet werden.
Für den Sammelkontext m_n gilt dies alles gleichermaßen, allerdings ist hier die Anzahl der Interrupts zu berücksichtigen:

$$N_{cc} = \sum_{\forall \tau_i \mapsto m_n} \left\lceil \frac{\Delta_\tau(\tau_i)}{t_i} \right\rceil + \mathcal{I}$$

$\mathcal{I}$ bezeichnet zusätzliche Warteplätze, die aufgrund von Interrupts eingefügt werden müssen, die nicht zur Aktivierung einer Task vorgesehen sind. Sie sind vom Designer des eingebetteten Systems entsprechend analysiert und hinzugefügt worden.
Insgesamt ist die jeweilige Länge der Wartschlange eines Kontextes damit applikationsspezifisch zu wählen und muss demnach ebenfalls rekonfigurierbar sein. Die Implementierung der Kontextcontroller ist zu diesem Zweck so gestaltet, dass jeder Kontextcontroller aus einem Pool von Warteschlangeneinträgen einen kontinuierlichen Bereich beanspruchen kann. Über jeweils eigene Zeiger auf Start- und Frei-Bereiche innerhalb dieses Pools erfolgt somit eine frei konfigurierbare Zuordnung von Warteschlangen nahezu beliebiger Länge zu den Kontextcontrollern. Begrenzt wird dies lediglich durch die Anzahl der Warteschlangeneinträge in dem Pool. Dieser ist bei der Erstellung der Scheduler-Hardware generisch und mit einem ausreichenden Wert zu besetzen (in der aktuellen Implementierung auf einem Xilinx-FPGA ist dieser auf den Wert 60 festgelegt). Für weitere Implementierungsdetails, insbesondere der rekonfigurierbaren Wartschlangenrealisierung, sei auch hier wiederum auf [44] verwiesen.

Zeitanalysen der Implementierung auf einem SPARTAN FPGA[203] der Firma Xilinx ergeben eine mögliche Zykluszeit des Schedulers von ca. 50MHz. Dies liegt über der Zykluszeit der verwendeten MSparc-Implementierung und ist natürlich wie die MSparc selbst durch moderne Technologie skalierbar. Insgesamt ergibt sich eine Verzögerungszeit von 6 Takten für die Annahme eines Interrupts mit folgender Initiierung einer Task. Zusammen mit der maximal 5 Zyklen dauernden Kontextwechselzeit der Integerpipeline ergibt sich eine Taskswitch-Zeit von maximal 11 Zyklen[1].

[1] Wobei in der Implementierung des HW-Schedulers noch Raum für Optimierungen ist und bei einem gründlichen Re-Design eine weitere Reduzierung der 6 Zyklen auf maximal 3-4 zu erwarten ist.

8.2.4 Weitere Hardware-unterstützte Konstrukte

Weitere durch Hardware unterstützte Eigenschaften, wie sie beispielsweise in [142] und [96] vorgestellt und für den Einsatz in Echtzeitsystemen sehr performant bewertet werden, sind problemlos in den bisher diskutierten Ansatz zu integrieren. Applikationsspezifische Timer-Realisierungen sind im Rahmen des EVENTS-Projektes auf der oben skizzierten Hardware generiert worden und zum Einsatz gekommen[107]. Ebenso ist die Verwendung von Hardware-basierten Semaphoren implementiert. Die in der EVENTS-Architektur[38] verwendeten Dualport Speicher der Firma Cypress [36] bieten einen solchen Mechanismus an. Eingeblendet in den Speicher der MSparc stellen sie einen effektiven Semaphorenmechanismus zur Verfügung, der sich nahtlos in die hier dargestellte Architektur eingliedert. Durch eine Kombination dieses Mechanismus mit dem Hardware-Scheduler ließe sich zudem eine Hardwareunterstützung für beispielsweise das Inheritant Priority Ceiling Protocol[146] einfach integrieren.
Auf eine weitergehende Darstellung dieser Themenfelder wird allerdings im folgenden verzichtet, da Untersuchungen der Auswirkungen derartiger Optimierungen nicht im Fokus dieser Arbeit liegen. Der interessierte Leser sei hier auf [142], [96] sowie die Veröffentlichungen des EVENTS-Projektes verwiesen.

8.3 Einfluss auf Antwortzeiten von Tasks

Mit Hilfe der Multithreading Technik können nun die Tasks aus dem Beispiel in Tabelle 7.1 so auf die Kontexte verteilt werden, dass diejenigen Tasks, die bezüglich der Scheduling-Kosten den höchsten Anteil haben, jeweils einem eigenen Kontext m_i zugewiesen werden. Alle übrigen Tasks werden zusammen mit einem Echtzeitbetriebssystem in dem letzten verbleibenden Kontext m_n gesammelt. Damit ergibt sich für die Tasks in einem Einzelkontext (also $\forall \tau_i$ mit $\tau_i \mapsto m_{k_i} \wedge |m_{k_i}| = 1$) eine Antwortzeit inklusive Scheduling-Overhead von:

$$r_i = c_i + c_{switch} + \sum_{\tau_j \in hp(\tau_i): \tau_j \mapsto m_{l_j} \wedge |m_{l_j}|=1} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_j + 2c_{switch}\right) + \sum_{\tau_j \in hp(\tau_i): \tau_j \mapsto m_{l_j} \wedge |m_{l_j}|>1} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_j + 2c_{switch} + c_{ISR} + 2c_{Sched}\right)$$

c_{switch} beinhaltet die Kosten für einen Kontextwechsel inklusive Ereignisannahme und Scheduling. Da diese Operationen in Hardware implementiert sind, ergeben sich ein paar wenige Zyklen für diese Operationen. In der MSparc sind diese Kosten beispielsweise mit maximal 11 Zyklen (siehe [114]) inklusive Scheduling und Leerlaufen der Pipeline angegeben (im Vergleich dazu verursacht der optimierte OSEK-Scheduler beispielsweise eine fast 200-fache Latenzzeit). In die Berechnung gehen nur höher priorisierte Tasks ein, da die niedriger priorisierten im Hardware-Scheduler parallel zur Ausführung der gerade aktiven Task abgefangen werden und den Prozessor selbst nicht belasten. Für den Einfluss höher priorisierter Tasks ist zu unterscheiden, ob sie ebenfalls einzeln in einem Kontext angesiedelt sind oder aber im verbleibenden Sammelkontext liegen. Im ersteren Fall fallen lediglich die

Kontextwechselzeiten als Kosten an, im zweiteren muss zusätzlich der durch das Betriebssystem verursachte Overhead Berücksichtigung finden.
Für alle Tasks, die sich einen Kontext mit anderen Tasks unter der Verwaltung eines RTOS teilen (also $\forall \tau_i : \tau_i \mapsto m_{k_i} \wedge |m_{k_i}| > 1$) gilt dementsprechend die folgende Berechnungsvorschrift für die Antwortzeit:

$$\begin{aligned} r_i =& c_i + c_{switch} + c_{ISR} + c_{Sched} \\ &+ \sum_{\tau_j \in hp(\tau_i):\tau_j \mapsto m_{l_j} \wedge |m_{l_j}|=1} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_j + 2c_{switch}\right) \\ &+ \sum_{\tau_j \in hp(\tau_i):\tau_j \mapsto m_{l_j} \wedge |m_{l_j}|>1} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_j + 2c_{switch} + c_{ISR} + 2c_{Sched}\right) \\ &+ \sum_{\tau_j \in lp(\tau_i) \wedge \tau_j \mapsto m_{k_i}} \left\lceil \frac{r_i}{t_j} \right\rceil \cdot \left(c_{ISR} + c_{Sched}\right) \end{aligned}$$

Zusätzlich zu der Kontextwechselzeit muss in diesem Fall im lokalen Kontext noch ein Schedule des RTOS durchgeführt werden, daher gehen in die Antwortzeit wiederum ISR und Scheduler-Aufruf mit ein. Die höher priorisierten Tasks gehen identisch zu der Berechnung der Einzel-Tasks mit ein. Niedriger priorisierte Tasks sind nur dann von Belang, wenn sie in demselben Kontext m_k angesiedelt sind, wie die zu analysierende Task selbst, da alle anderen anfallenden Ereignisse parallel vom Hardware-Scheduler abgefangen und dem jeweiligen Kontext zugeordnet werden, ohne die Ausführung der gerade aktiven und höher priorisierten Tasks in irgendeiner Art und Weise zu stören.

Angewendet auf das Beispiel in Tabelle 7.1, scheint die folgende Strategie der Verteilung optimal zu sein[1]: Sind n Kontexte gegeben, so ordne den $n-1$ ersten Kontexten jeweils eine Task mit der höchsten Priorität aus der Menge der noch verbleibenden Tasks zu. Gegeben eine multithreaded Architektur mit $n = 8$ Kontexten bedeutet dies, dass die 7 $1ms$-Tasks jeweils einem eigenen Kontext zugeordnet werden ($\tau_i \mapsto m_i$ mit $i \in \{1, \ldots, n-1\}$ und dass alle verbleibenden Tasks in den letzten verbleibenden Kontext mit einem RTOS plaziert werden ($\tau_i \mapsto m_n$ mit $i \in \{n, \ldots, 26\}$). Bezieht man die entstehenden Scheduling-Kosten wiederum auf die Utilisation des Prozessors, so erhält man für das Beispiel aus Tabelle 7.1 mit der hier angegebenen Verteilung die folgende Prozessorauslastung nur für Scheduling-Kosten:

$$U_{sched} = \sum_{\tau_i : i \in \{1,\ldots,n-1\}} \frac{2c_{switch}}{t_i} + \sum_{\tau_i : i \in \{n,\ldots,26\}} \frac{2c_{switch} + c_{ISR} + 2c_{Sched}}{t_i} \tag{8.1}$$

Tasks, die im letzten Kontext liegen, verursachen somit eine geringfügig höhere Latenzzeit durch den zusätzlichen Kontextwechsel, der im schlimmsten Fall mitzubetrachten ist. Allerdings ist der Overhead diesbezüglich so klein im Verhältnis zu den Software-Kosten für einen Taskwechsel, dass er kaum Auswirkungen auf die verbrauchte Prozessorauslastung hat. Vorausgesetzt der Prozessor hat 8 Kontexte, arbeitet mit einem Takt von 40 MHz und im Kontext m_n kommt ein optimiertes

[1] In den folgenden Abschnitten wird auf die Strategie der Zuordnung von Tasks zu Kontexten und deren Optimalität noch detailliert eingegangen.

OSEK-Betriebssystem zum Einsatz, so ergibt sich aus Gleichung (8.1) eine Prozessorauslastung U_{sched} von etwa 12%. Bezogen auf die theoretisch sichere Utilisationsgrenze für Rate Monotonic Systeme von 69% bedeutet dies, dass 82% der Utilisation für die Tasks als Berechnungszeit zur Verfügung stehen. Dies ist gegenüber dem Beispiel ohne Multithreading in Kapitel 7.3 eine Steigerung der zur Verfügung stehenden Prozessorauslastung um den Faktor 4.5. Anders herum ausgedrückt ergibt sich eine dramatische Erniedrigung der Utilisation, die vom Betriebssystem verbraucht wird, um ca. 78%. Die Auswirkungen der Interferenzfreiheit durch Disjunktheit der Systemzustände der Kontexte untereinander ist hierbei noch gar nicht erfasst und wird im Abschnitt 11.4.2 detailliert aufgegriffen und erläutert.
Multithreading als Technik auf einem Prozessor einzuführen hat aber natürlich auch negative Auswirkungen, da innerhalb des Datenpfades eines Prozessors der jeweilige Kontext selektiert werden muss. Dies bedeutet zusätzliche Multiplexer-Hierarchien, die potentiell auf dem kritischen Pfad des Datenpfades liegen können und mithin die mögliche maximale Taktfrequenz des Prozessors erniedrigen. Praktische Untersuchungen an der MSparc haben gezeigt, dass diese zusätzlichen Verzögerungen pro Hierarchiestufe der Multiplexer unter 5% der Taktfrequenz liegen (vergleiche auch die Datenblätter des verwendeten $0.5\mu m$ Standardzellenprozess[4] und die Dokumentation der MSparc[114]). Dieser Effekt kann das obige Ergebnis wiederum geringfügig verschlechtern. In der MSparc selbst sind durch die Konstruktion, dass die Pipelinestufen keine Kontextverwaltung besitzen (siehe letztes Kapitel), die Multiplexerhierarchien zum Selektieren der Kontexte *nicht* auf den kritischen Pfaden, so dass keine negativen Einflüsse auf die Zykluslänge des Prozessors auftreten. Dies macht sich zwar in der Latenzzeit des Kontextwechsels bemerkbar (4 zusätzliche Zyklen für das leerlaufen der Pipeline), ist aber aufgrund der relativ seltenen Kontextwechsel kein wesentlich negativer Faktor. Insgesammt sollte eine Implementierung, wie sie in der MSparc gewählt wurde, deshalb beim Einsatz von Multithreading in Prozessoren für Echtzeitsysteme bevorzugt werden.
Damit konnte nachgewiesen werden, welches erhebliche performanzsteigernde Potential durch den Einsatz spezialisierter Hardware in Echtzeitsystemen erreicht werden kann. Es bleibt aber noch zu klären, in welcher Weise die Tasks auf die Kontexte verteilt werden sollen, um eine optimal niedrige Utilisation bzgl. der Schedulingkosten zu erreichen. Mit diesem Thema beschäftigen sich die beiden folgenden Kapitel, wobei dort auch nocheinmal die Interferenz von Tasks thematisiert wird.

8.4 Partitionierung

8.4.1 Verringerung von Scheduling-Kosten

Wie in den vorangegangenen Kapiteln dargestellt wurde, kann der Performanzgewinn durch eine drastische Verkleinerung des Betriebssystem-Overheads dramatisch groß sein. Dies wird erreicht, in dem die einzelnen Tasks eines Tasksystems auf die Kontexte des multithreaded Prozessors so verteilt werden, dass die eingesparten Kosten, die sich aus der Differenz der Schedulingkosten des Betriebssystem und des Hardware-Schedulers ergeben, maximal werden. Ein Maß für die Bewertung der

Performanz eines Tasksystems kann die Utilisation sein, die die benötigte Berechnungszeit mithilfe der Task-Perioden auf ein Intervall der Länge 1 normiert (vgl. auch [104]). Der in Kapitel 7.2 dargestellte Anteil der Scheduling-Kosten bei der Berechnung der Antwortzeit von Tasks kann auf die Utilisation übertragen werden, wenn man annimmt, dass bei jedem Dispatchen der Task eine feste Zeitspanne für den Taskwechsel anfällt. Diese Zeitspanne soll im folgenden so gewählt sein, dass sie sowohl den Anteil einer Interrupt Service Routine als auch die eigentlichen Kosten des Schedulers enthält. Damit erhält man bei einer Berechnungszeit von c_i einer Task τ_i die folgende Gleichung für die Gesamtutilisation eines Tasksystems auf einem multithreaded Prozessor:

$$U = \sum_{\{\tau_i\} \mapsto m_j} \frac{c_i + c_{sw}^{CS}}{t_i} + \sum_{\{\tau_i, \ldots\} \mapsto m_j} \frac{c_i + c_{sw}^{OS}}{t_i}$$

$\tau_i \mapsto m_j$ bzw. $\{\tau_i, \ldots\} \mapsto m_j$ bezeichnet dabei die Zuweisung einer oder mehrerer Tasks auf einen Kontext m_j. Ist einem Kontext nur eine einzige Task zugeordnet, so muss lediglich der Kostenfaktor für einen Hardware-Scheduler $c_{sw}^{CS} = 2c_{switch}$ berücksichtigt werden. Andernfalls ist ein Betriebssystem-Scheduler mit $c_{sw}^{OS} = 2c_{switch} + c_{ISR} + 2c_{Sched}$ in die Berechnung einzubeziehen. Die Schedulingkosten des Hardware-Schedulers liegen dabei, wie im Eingangs erwähnten Beispiel dargelegt, um einige Größenordnungen unter denen des Betriebssystem-Schedulers.

Betrachtet man ein Tasksystem, das auf einem gegebenen Prozessor, der mit der Multithreading-Technik ausgestattet ist, platziert werden soll, so stellt sich die Frage, welche Tasks einzeln in einen Kontext sollen, welche Tasks sich mit mehreren anderen einen Kontext teilen, ob es leere Kontexte aus Performancegründen geben sollte, etc. D.h., es ist ein Optimierungsproblem zu lösen, dessen Ziel es ist, die globale Utilisation auf dem gegebenen Prozessor mit dem gegebenen Tasksystem zu minimieren. Der Einfachheit halber wird im folgenden davon ausgegangen, dass das Tasksystem nur periodische (oder sporadische mit minimaler Zwischenankunftzeit der Invokationen) Tasks enthalten darf und dem in [104] dargestellten Ansatz des Rate Monotonic Scheduling genügt. Dies beinhaltet im übrigen einen preemptiven Schedule, der die Tasks prioritätenorientiert dispatched.
Die optimale Zuordnung bezüglich Minimalität der Utilisation des Prozessors einer Taskmenge $\mathcal{T}$ mit $|\mathcal{T}| \geq n$, Prioritätenvergabe nach dem Rate Monotonic Prinzip und Interferenzfreiheit der Tasks untereinander, ist gegeben, wenn die $n-1$ höchst priorisierten Tasks jeweils einzeln einem Kontext und alle anderen Tasks dem letzten verbliebenen Kontext zugeordnet werden (siehe Algorithmus 6). Dabei wird eine Kontextanzahl von n vorausgesetzt.

Der Zuordnungsalgorithmus 6 entspricht folglich einer sukzessiven Verteilung der $n-1$ höchstpriorisierten Tasks auf die $n-1$ Kontexte des Prozessors und bewirkt damit, dass diese Tasks die Utilisation nur mit den Kosten eines Hardware-Schedules belasten. Alle übrigen Tasks werden hingegen dem sogenannten "Sammelkontext" m_n zugewiesen und belasten die Gesamtutilisation mit den Kosten eines Betriebssystem-Schedules. Dass dieser Algorithmus optimal hinsichtlich der Minimierung der Utilisation ist, wird in nachfolgendem Satz nachgewiesen. Da die Zuteilung von Tasks

Algorithmus 6: Linerare Zuordnung von Tasks zu Kontexten unter Interferenzfreiheit

Voraussetzung: $r = |\mathcal{T}| > n$
  for $i = 1$ **to** $n - 1$ **do**
    $\tau_i \mapsto m_i$
  end for
  for $i = n$ **to** r **do**
    $\tau_i \mapsto m_n$
  end for

zu Kontexten trivial ist, wenn die Anzahl der Tasks kleiner oder gleich der Anzahl der Kontexte ist, wird dieser Fall im folgenden keine weitere Beachtung finden.

Satz *"Optimalität der linearen Zuordnung unter Interferenzfreiheit"*
Gegeben ist ein multithreaded Prozessor P mit n Kontexten, sowie eine Taskmenge $\mathcal{T}$ mit $|\mathcal{T}| > n$, deren Tasks nach dem Rate Monotonic Prinzip priorisiert und interferenzfrei sind. Es gilt dann: U ist minimal, wenn gilt $\{\tau_0\} \mapsto m_0, \ldots, \{\tau_{n-1}\} \mapsto m_{n-1}, \{\tau_n, \ldots, \tau_r\} \mapsto m_n$, wobei die Tasks in absteigender Priorität indiziert sind.

Beweis:
Seien die Task $\tau_i \in \mathcal{T}$ so numeriert, dass die Indizes in aufsteigender Reihenfolge die absteigende Priorität der Task ergibt. Seien c_{sw}^{OS} die Kosten eines Taskwechsel durch einen Echtzeit Betriebssystem-Scheduler, sei c_{sw}^{CS} die Kosten eines Taskwechsels durch einen internen Kontextwechsel des Prozessors. c_i seien die Kosten der Taskausführung im schlimmsten anzunehmenden Fall (WCET). Sei ferner die Ausführungszeit einer Task inklusive Taskwechselkosten gegeben durch $c_i^{OS} = c_i + c_{sw}^{OS}$ bzw. $c_i^{CS} = c_i + c_{sw}^{CS}$. Sei $\{m_i | i \in \{1, \ldots, n\}\}$ die Menge der Kontexte. Der Beweis zerfällt in 3 Teile:

1. Bleibt ein Kontext unbelegt, ist die Utilisation nicht minimal. Sei $U_\emptyset$ die Utilisation für den Fall $(\tau_0 \mapsto m_0, \ldots, \emptyset \mapsto m_i, \tau_i \mapsto m_{i+1}, \ldots, \{\tau_{n-1}, \ldots\} \mapsto m_n)$, sei U die Utilisation für den Fall $(\tau_0 \mapsto m_0, \ldots, \{\tau_n, \ldots\} \mapsto m_n)$. Dann gilt $U_\emptyset \geq U$.

2. Liegt eine niedrig priorisierte Task in einem Kontext $m_i, i \neq n$ und eine hoch priorisierte im Sammelkontext m_n, ist die Utilisation nicht minimal. Sei U_T die Utilisation für den Fall $(\tau_0 \mapsto m_0, \ldots, \tau_k \mapsto m_i, \tau_{i+1} \mapsto m_{i+1}, \ldots, \{\tau_n, \ldots \tau_{k-1}, \tau_i, \tau_{k+1}, \ldots\} \mapsto m_n)$ mit $i < k$ und U die Utilisation im Fall $\tau_0 \mapsto m_0, \ldots, \{\tau_n, \ldots\} \mapsto m_n$, dann gilt $U_T \geq U$.

3. Sind $n - 1$ Kontexte mit den jeweils höchst priorisierten Tasks einfach gefüllt, und wird eine niedriger priorisierte Task aus dem Sammelkontext m_n in einen anderen Kontext m_i verschoben, ist die Utilisation nicht minimal. Sei U_M die Utilisation für den Fall $(\tau_0 \mapsto m_0, \ldots, \{\tau_i, \tau_k\} \mapsto m_i, \tau_{i+1} \mapsto m_{i+1}, \ldots, \{\tau_n, \ldots \tau_{k-1}, \tau_{k+1}, \ldots\} \mapsto m_n)$ mit $n \leq k$ und U die Utilisation im Fall $(\tau_0 \mapsto m_0, \ldots, \{\tau_n, \ldots\} \mapsto m_n)$, dann gilt $U_M \geq U$.

Grundsätzlich gilt für die Ausführungszeit einer Task τ_i mit $\mathcal{T} \supseteq \theta, \theta \mapsto m_j$ wobei

$\tau_i \in \theta$:

$$c(\tau_i) = \begin{cases} c_i^{OS} & \text{wenn } |\theta| > 1 \\ c_i^{CS} & \text{sonst} \end{cases}$$

Damit lassen sich nun die 3 Fälle beweisen:
zu 1.):
Beweis durch Widerspruch, dann muss gelten:

$$\begin{aligned} & U_\emptyset < U \\ \Leftrightarrow \quad & \frac{c_0^{CS}}{t_0} + \ldots + \frac{c_{n-2}^{CS}}{t_{n-2}} + \frac{c_{n-1}^{OS}}{t_{n-1}} + \ldots \frac{c_r^{OS}}{t_r} < \frac{c_0^{CS}}{t_0} + \ldots + \frac{c_{n-1}^{CS}}{t_{n-1}} + \frac{c_n^{OS}}{t_n} + \ldots \frac{c_r^{OS}}{t_r} \\ \Leftrightarrow \quad & \frac{c_{n-1}^{OS}}{t_{n-1}} < \frac{c_{n-1}^{CS}}{t_{n-1}} \\ \Leftrightarrow \quad & c_{sw}^{OS} - c_{sw}^{CS} \not< 0 \end{aligned}$$

Dies ist, da $c_{sw}^{OS} > c_{sw}^{CS}$, offensichtlich ein Widerspruch zur Annahme.

zu 2.):
Zunächst sei $n = 2$, dann gilt laut Behauptung $U_T \geq U$. Beweis durch Widerspruch, dann muss gelten:

$$\begin{aligned} & U > U_T \\ \Leftrightarrow \quad & \frac{c_1^{OS}}{t_1} + \ldots + \frac{c_r^{OS}}{t_r} + \frac{c_0^{CS}}{t_0} > \frac{c_0^{OS}}{t_0} + \ldots + \frac{c_{k-1}^{OS}}{t_{k-1}} + \frac{c_k^{CS}}{t_k} + \frac{c_{k+1}^{OS}}{t_{k+1}} + \ldots + \frac{c_r^{OS}}{t_r} \\ \Leftrightarrow \quad & \frac{c_0^{CS}}{t_0} + \frac{c_k^{OS}}{t_k} > \frac{c_0^{OS}}{t_0} + \frac{c_k^{CS}}{t_k} \\ \Leftrightarrow \quad & \frac{c_0^{OS} - c_0^{CS}}{t_0} + \frac{c_k^{CS} - c_k^{OS}}{t_k} < 0 \\ \Leftrightarrow \quad & \frac{c_{sw}^{OS} - c_{sw}^{CS}}{t_0} + \frac{c_{sw}^{CS} - c_{sw}^{OS}}{t_k} < 0 \\ \Leftrightarrow \quad & \frac{t_k \cdot c_{sw}^{OS} - t_0 \cdot c_{sw}^{OS} + t_0 \cdot c_{sw}^{CS} - t_k \cdot c_{sw}^{CS}}{t_0 \cdot t_k} < 0 \\ \Leftrightarrow \quad & \frac{\overbrace{(t_k - t_0)}^{a} c_{sw}^{OS} + \overbrace{(t_0 - t_k)}^{-a} c_{sw}^{CS}}{t_0 \cdot t_k} < 0 \\ \Leftrightarrow \quad & \frac{a\left(c_{sw}^{OS} - c_{sw}^{CS}\right)}{t_0 \cdot t_k} < 0 \end{aligned}$$

Da $c_{sw}^{OS} > c_{sw}^{CS} \wedge a \geq 0 \wedge t_0 \cdot t_k > 0$ gilt aber $\frac{a\left(c_{sw}^{OS} - c_{sw}^{CS}\right)}{t_0 \cdot t_k} \geq 0$.
Dies ist ein Widerspruch zur Annahme.

Induktiv läßt sich dieser Beweis auf mehrere Kontexte erweitern, indem jeweils ein neuer Kontext hinzugenommen wird und nur dieser einer Betrachtung wie oben unterzogen wird. Alle anderen Kontexte mit nur einer zugeordneten Task außer dem Sammelkontext m_n leisten sowohl für U_T als auch für U den identischen Beitrag $\frac{c_i^{CS}}{t_i}$ und können somit aus der Ungleichung $U_T < U$ entfernt werden.

zu 3.)
Beweis durch Widerspruch, dann muss gelten:

$$
\begin{aligned}
& & U_M &< U \\
\Leftrightarrow & \frac{c_0^{CS}}{t_0} + \ldots \frac{c_i^{OS}}{t_i} + \frac{c_j^{OS}}{t_j} + \frac{c_{i+1}^{CS}}{t_{i+1}} + \ldots + \frac{c_n^{OS}}{t_n} + \cdots + \frac{c_{j-1}^{OS}}{t_{j-1}} + \frac{c_{j+1}^{OS}}{t_{j+1}} + \ldots \frac{c_r^{OS}}{t_r} \\
& & &< \frac{c_0^{CS}}{t_0} + \ldots \frac{c_{n-1}^{CS}}{t_{n-1}} + \frac{c_n^{OS}}{t_n} + \ldots + \frac{c_r^{OS}}{t_r} \\
\Leftrightarrow & & \frac{c_i^{OS}}{t_i} + \frac{c_j^{OS}}{t_j} &< \frac{c_i^{CS}}{t_i} + \frac{c_j^{OS}}{t_j} \\
\Leftrightarrow & & \frac{c_i^{OS} - c_i^{CS}}{t_i} + \frac{c_j^{OS} - c_j^{OS}}{t_j} &< 0 \\
\Leftrightarrow & & c_{sw}^{OS} - c_{sw}^{CS} &\not< 0
\end{aligned}
$$

Dies ist, da $c_{sw}^{OS} > c_{sw}^{CS}$, offensichtlich ein Widerspruch zur Annahme. □

Damit ist ein linearer Algorithmus gegeben, der die Zuordnung von Tasks auf einen mit multithreading Techniken ausgestatteten Prozessor berechnet, solange Interferenzfreiheit der Tasks untereinander vorrausgesetzt wird. Letzteres bedeutet, dass die benutzten Cache-Sets der Tasks vollständig disjunkt sind, d.h. es bei einer Preemption von τ_i durch τ_j zu keiner Invalidierung der zu τ_i gehörenden Cache-Inhalte kommt. Zwar ist diese Vorraussetzung im allgemeinen nicht von einem Tasksystem zu erwarten und massiv abhängig von der Größe der Caches und der Working Sets der jeweiligen Tasks, jedoch gibt es Techniken, eine Interferenzfreiheit herzustellen. In [124] und [202] wird eine Methodik vorgestellt, die es ermöglicht, die Working Sets der jeweiligen Tasks bzgl. des Instruction Caches so im Speicher zu verteilen, dass tatsächlich eine Interferenzfreiheit bei preemptiven Scheduling erreicht werden kann. Dies ist jedoch unter Umständen mit einer erhöhten Laufzeit verbunden, da Programmfragmente im Speicher verlagert werden und somit der Kontrollfluss einer Task an den Bruchstellen dieser Fragmente mit zusätzlichen Sprungbefehlen erweitert werden muss. Diese Vorgehensweise ist direkt vom auf dem betrachteten ECU-Knoten platzierten Tasksystem abhängig. Somit ist die WCET einer Task auf einem Knoten nicht konstant, sondern abhängig von den konkurrierenden Tasks auf diesem Knoten. Eine dynamische WCET widerspricht aber dem in Teil I vorgestellten inkrementellen Entwurfsprinzipien und verkompliziert die Suche nach einer optimalen Verteilung der Tasks auf eine gegebene Architektur in starkem Maße. Zudem ist nicht sichergestellt, dass die Tasks tatsächlich mithilfe dieser Technik interferenzfrei gestellt werden können, da dies nicht unerheblich von den Größen des Caches und der Working Sets der Tasks abhängt. In multithreaded Prozessoren mit einer statischen Cacheaufteilung wird zudem die Größe der einzelnen Caches nur $\frac{1}{n}$ der üblichen Cachegröße betragen. Im besonderen für den Sammelkontext m_n wird sich daher aller Vorraussicht nach eine Interferenzfreiheit mittels der Partitionierungstechniken aus [124, 202] nur schwer herstellen lassen. Um den Einfluss von Cache-Interferenzen auf die Verteilung der Tasks zu den Kontexten zu qantifizieren, wird im folgenden Abschnitt ein Algorithmus entwickelt, der den Umstand der nicht vorhandenen Interferenzfreiheit berücksichtigt.

8.4.2 Einfluss von Cache-Interferenzen auf die Laufzeit

Statische Partitionierungen des Instruction-Caches auf die vorhandenen Tasks ist eine Technik, die schon Ende der 80er Jahre von Kirk in [88] vorgeschlagen wurde. Allerdings war die Zielrichtung des dort vorgestellten Ansatzes nur die Vermeidung von Cacheinterferenzen der Tasks untereinander und betraf nicht den generelleren Ansatz die Utilisation zu minimieren, den Multithreading für Echtzeitsysteme liefern kann. Cache-Interferenzen sind ein generelles Problem in Systemen mit preemptiven Schedulingverfahren und die Auswirkungen müssen quantifiziert werden, um mithilfe der herkömmlichen Scheduling-Analyse-Verfahren korrekte Antwortzeiten von Tasks zu berechnen. Betrachtet man die in Teil I verwendeten Analysen des Zeitverhaltens, so kann man den Effekt der Verdrängung von Cache-Inhalten durch Preemptionen hoch priorisierter Tasks leicht in die Fixpunktgleichung zur Berechnung der Antwortzeit einbauen. Generell ist für diese Betrachtung der Einfluss von Taskpaaren auf ihr jeweiliges Cacheverhalten zu berücksichtigen, d.h. es gibt für jede Task τ_j, die eine niedriger priorisierte Task τ_i unterbricht, einen konstanten Kostenfaktor γ_{ij}, der i.a. durch eine überapproximierende Analyse des Taskpaares gewonnen wird. In diesem Kostenfaktor spiegeln sich die Anzahl der Cache-Verdrängungen bzgl. der unterbrochenen Task und die Größenordnung der Kosten eines Cache Misses in der gegebenen Architektur wider. Jede Preemption von τ_i durch τ_j verursacht potentiell diese Kosten, und somit kann die Antwortzeitberechnung einer Task erweitert werden zu:

$$r_i = c_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{r_i}{t_j} \right\rceil (c_j + \gamma_{ij}) \tag{8.2}$$

Aus der Literatur sind Verfahren bekannt, für Tasksysteme die jeweiligen Interferenz-Kostenfaktoren mehr oder minder genau zu quantifizieren: Die in 8.2 angegebene Form der Integration von Interferenzkosten wurde erstmals in [26] entwickelt und formal fundiert. [158] erweitert diesen Ansatz um Mechanismen, die Kosten einer Interferenz bzgl. Pipeline und Caches zu berechnen und somit den Faktor γ_{ij} applikationsabhängig korrekt im Sinne der worst-case Annahme zu liefern. Einen Schritt weiter gehen die Autoren in [95], indem die konstante Natur der Interferenzkosten ersetzt wird durch einen dynamisch zur Berechnung der Antwortzeit gefundenen Kostenfaktor, der von der bis zur Unterbrechung vorangeschrittenen Zeit abhängt. Über sogenannte "useful cache blocks" wird ermittelt, welche Interferenzkosten ab dem aktuellen r_i maximal zu erwarten sind. Hierfür ist für jeden Schritt der Gleichung 8.2 ein ILP-Problem zu lösen, welches basierend auf den Antwortzeiten höher priorisierter Tasks und dem durch eine Datenflussanalyse erzeugten potentiellen Interferenzkosten an jedem Punkt des Kontrollflusses eine Task die maximale Interferenz berechnet.

Derartige Analysen liefern problemspezifische Lösung, die a priori keine generellen Rückschlüsse zu einem Verteilungsvorgehen der Tasks auf die Kontexte eines multithreaded Prozessors zulassen. Zudem ist bei einem Verteilungsproblem, wie es in Teil I dieser Arbeit vorgestellt wurde, jede mögliche Kombination von Taskpaarungen in dieser Analyse zu betrachten. Aus Komplexitätsgründen ist dies bei Tasksystemen der Größenordnungen, wie sie üblicherweise in den in Teil I dargestellten Anwen-

dungsdomänen verwendet werden, mit den in [26, 158] vorgestellten Methoden nicht mehr machbar (die Anzahl der Kombinationsmöglichkeiten wächst bekanntlich exponentiell mit der Anzahl der Tasks). Für den Ansatz aus [95] sind nicht nur alle Taskpaare zu betrachten, sondern zusätzlich noch alle möglichen Paarungen von Taskmengen auf einem Knoten[1].
Um eine generelle Aussage bzgl. der Partitionierungsrichtlinien für Tasks auf Kontexte zu bekommen und um dem Komplexitätsproblem der Analyse aller möglichen Paarungen zu entgehen, wird im Kontext dieser Arbeit eine Überapproximation der Kosten, die durch Interferenz entstehen, aufgebaut, mit deren Hilfe sich schließlich ein Algorithmus ableiten läßt, der eine Verteilungslösung von Tasks auf Kontexten mit einer hohen Güte erlaubt. Die Kosten der Interferenz zweier Tasks wird dazu vereinheitlicht, d.h. wann immer eine Task τ_j eine andere Task τ_i unterbricht, wird sie einen konstanten Kostenanteil erzeugen, der aus dem Maximum aller paarweisen Kosten entsteht[2], also:

$$\gamma_j = \max\{\gamma_{ij} \mid j < i\} \tag{8.3}$$

Der wesentlich Vorteil des Einsatzes von disjunkten Caches für die Kontexte liegt, wie oben schon angedeutet, in der Vermeidung von Intereferenzen von Tasks, die unterschiedlichen Kontexten zugeordnet sind. Es gilt also:

$$\forall \tau_i, \tau_j \in \mathcal{T}, \theta_i \mapsto m_l, \theta_j \mapsto m_k : \tau_i \in \theta_l \wedge \tau_j \in \theta_k \wedge \theta_l \neq \theta_k \Rightarrow \gamma_{ij} = 0$$

Zur Analyse eines solchen Systems wird wiederum die Utilisation auf dem Prozessor als Optimierungskriterium betrachtet. Als obere Grenze kann angenommen werden, dass bei jedem Aufruf einer Task eine Task niedrigerer Priorität unterbrochen und die Interferenzkosten induziert werden. Folglich werden die Kostenfaktoren γ_j (wie schon die Scheduling-Kosten im letzten Abschnitt) in die Utilisationsberechnung einer Task aufgenommen. Dabei muss zusätzlich beachtet werden, dass eine Task, die aufgrund ihrer niedrigen Priorität keine anderen unterbrechen kann, keine zusätzlichen Interferenzkosten verursacht. Die dadurch entstehende differenziertere Betrachtung der Ausführungszeit einer Task liefert dann die folgenden Kostenfaktoren:

$$\begin{array}{ll} c_j^{OS} = c_j + c_{sw}^{OS} + \gamma_j & \\ \check{c}_j^{OS} = c_j + c_{sw}^{OS} & \text{für die niedrigst priorisierte Task in einem Kontext} \\ c_j^{CS} = c_j + c_{sw}^{CS} & \text{für Tasks, die alleine einem Kontext zugeordnet sind} \end{array}$$

Offensichtlich gilt der Satz *Optimalität der lineraren Zuordnung unter Interferenzreiheit* nicht so ohne weiteres, wenn die Interferenzfreiheit nicht vorausgesetzt werden kann. Der Beweis dieses Satzes läßt sich unter Einbeziehung von Interferenzen nicht mehr führen: Es ist durchaus möglich, dass eine niedrig priorisierte Task so hohe Interferenzkosten im Kontext m_n verursacht, dass ein Vertauschen mit einer höher priorisierten Task, die in einem Kontext m_l mit $l < n$ liegt, eine erniedrigende Wirkung auf die Gesamtutilisation hat. Folglich ist die Aussage 2.) im Satz *Optimalität*

[1] Wie oben beschrieben, berechnen sich die Interferenzkosten mit dieser Methode basierend auf u.a. den Antwortzeiten *aller* höher priorisierten Tasks des Knotens.

[2] Man könnte, wie im Eingangs gezeigten Beispiel, auch die Anzahl der benutzten Cachezeilen einer Task als obere Grenze annehmen.

der lineraren Zuordnung unter Interferenzreiheit nicht mehr nachweisbar. Stattdessen soll hier ein Regelwerk abgeleitet werden, mit dem es möglich ist, eine nahezu optimale Verteilung der Tasks auf die Kontexte — abhängig von den jeweils induzierten Interferenzkosten und Betriebssystemkosten — zu berechnen.
Die zugrunde liegende Idee eines Verteilungsalgorithmus ist, zunächst initial mit einer linearen Zuordnung (wie im letzten Abschnitt beschrieben) zu beginnen. Anschließend werden mögliche Vertauschungen bzgl. ihrer Wirkung auf die Gesamtutilisation untersucht und ggf. vorgenommen. Hierzu werden im folgenden Funktionen der Grenzfälle betrachtet, d.h. ab welcher Parameterbelegung bewirkt eine bestimmte Vertauschung der Zuordnung von Tasks zu Kontexten eine Verringerung der Gesamtutilisation. Analog zu den 3 Aussagen in der Beweisführung des Satzes von der *Optimalität der lineraren Zuordnung unter Interferenzreiheit* werden dabei 3 Fälle untersucht:

1. Ab welcher Bedingung ist eine Vertauschung einer niedriger priorisierten Task aus dem Sammelkontext m_n mit einer hoch priorisierten Task in einem Einzelkontext m_i günstiger?

2. Ab welcher Bedingung lohnt es sich, eine niedriger priorisierte Task aus dem Sammelkontext m_n zusätzlich in einen Einzelkontext m_i zu verschieben, wenn dort schon eine hoch priorisierte Task allokiert ist?

3. Ab welcher Bedingung lohnt es sich, den Sammelkontext komplett bis auf eine Task zu leeren?

Die Bedingungen für diese 3 Kriterien sollen nun anhand der Utilisationsformeln abgeleitet werden und dienen dann zur Bestimmung des Verteilungsalgorithmus. Zur Herleitung wird der Fall untersucht, dass eine Tauschung eine geringere Utilisation hervorruft, also gilt $U_T < U$.

Zu 1.):
Die Herleitung beschränkt sich oBdA. auf einen Prozessor mit $n = 2$. Da zusätzliche Kontexte von der betrachteten Tauschoperation nicht betroffen sind, liefern sie auf beiden Seiten der Gleichung identische Anteile an der Utilisation und können somit ignoriert werden.

$$\frac{c_1^{OS}}{t_1} + \ldots + \frac{\breve{c}_r^{OS}}{t_r} + \frac{c_0^{CS}}{t_0} > \frac{c_0^{OS}}{t_0} + \ldots + \frac{c_{k-1}^{OS}}{t_{k-1}} + \frac{c_k^{CS}}{t_k} + \frac{c_{k+1}^{OS}}{t_{k+1}} + \ldots + \frac{\breve{c}_r^{OS}}{t_r}$$

$$\Leftrightarrow \quad \frac{c_0^{OS} - c_0^{CS}}{t_0} + \frac{c_k^{CS} - c_k^{OS}}{t_k} < 0$$

$$\Leftrightarrow \quad \frac{c_{sw}^{OS} - c_{sw}^{CS} + \gamma_0}{t_0} + \frac{c_{sw}^{CS} - c_{sw}^{OS} - \gamma_k}{t_k} < 0$$

$$\Leftrightarrow \quad \frac{t_k\big(c_{sw}^{OS} - c_{sw}^{CS} + \gamma_0\big) + t_0\big(c_{sw}^{CS} - c_{sw}^{OS} - \gamma_k\big)}{t_k \cdot t_0} < 0$$

$$\Leftrightarrow \quad \big(t_k - t_0\big)c_{sw}^{OS} + \big(t_0 - t_k\big)c_{sw}^{CS} + t_k\gamma_0 - t_0\gamma_k < 0$$

Sei die durch das Betriebssystem bzw. den internen Kontextwechsel induzierte Differenz der Schedulingkosten durch $\delta_{sw} = c_{sw}^{OS} - c_{sw}^{CS}$ gegeben.

$$\Leftrightarrow \quad t_k\big(\delta_{sw} + \gamma_0\big) - t_0\big(\delta_{sw} + \gamma_k\big) < 0$$

Sei das Verhältnis der Perioden der beiden betrachteten Tasks gegeben durch $t_k = l \cdot t_0$ mit $l > 0$.

$$\begin{aligned} &\Leftrightarrow \quad l \cdot t_0\big(\delta_{sw} + \gamma_0\big) - t_0\big(\delta_{sw} + \gamma_k\big) < 0 \\ &\Leftrightarrow \quad l\delta_{sw} - \delta_{sw} + l\gamma_0 - \gamma_k < 0 \\ &\Leftrightarrow \quad \gamma_k > \big(l-1\big)\delta_{sw} + l\gamma_0 \end{aligned}$$

Damit ist eine Bedingung basierend auf den Interferenzkosten der betrachteten Tasks und den Schedulingkosten formuliert, die den Grenzfall darstellt, ab wann eine Vertauschung einer niedriger priorisierten Task aus dem Sammelkontext m_n mit einer hoch priorisierten Task aus einem Einzelkontext eine Verringerung der Gesamtutilisation zur Folge hat:

$$\gamma_k > \big(\frac{t_k}{t_0} - 1\big)\delta_{sw} + \frac{t_k}{t_0}\gamma_0 \tag{8.4}$$

Dieses Ergebnis formalisiert die oben angesprochene Möglichkeit der Verbesserung der Utilisation durch einen Tausch. Sobald die durch die niedriger priorisierte Task erzeugte Interferenzlatenz bei anderen Tasks größer ist als die Schedulingkosten des Betriebssystems plus die neu induziertes Interferenzlatenz der höher priorisierten Task relativ zum Verhältnis der Perioden, lohnt sich eine Vertauschung. Betrachtet man dieses Kriterium für ein paar ausgewählte Beispielkonfigurationen, so kann man die Linearität abhängig von den Parametern erkennnen (siehe Abbildung 8.4).

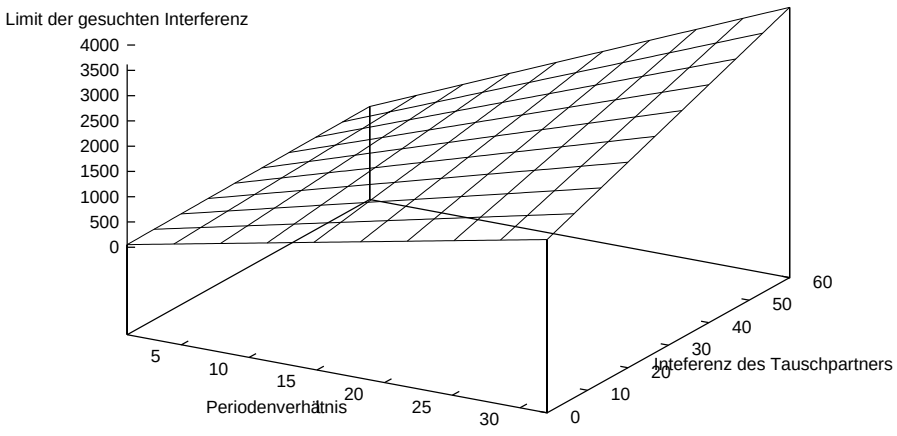


Abbildung 8.4: Beispiele für das Tauschkriterium bei $\delta_{sw} = 53\mu s$ Zeiteinheiten. Unterhalb der jeweiligen Grenzlinie ist das lineare Verfahren günstiger, oberhalb die Tauschung. γ_0 wird von $0\mu s$ bis $60\mu s$ variiert. Schon bei geringem Wachstum des Periodenverhältnisses muss γ_k ein Vielfaches von δ_{sw} betragen, bevor ein Wechsel lohnt.

Zu 2.):
OdBA wird wiederum die Konfiguration mit $n = 2$ betrachtet. Dann gilt:

$$\begin{aligned}
& \frac{c_0^{OS}}{t_0} + \frac{\breve{c}_k^{OS}}{t_k} + \frac{c_n^{OS}}{t_n} + \ldots + \frac{c_{k-1}^{OS}}{t_{k-1}} + \ldots \frac{c_{k+1}^{OS}}{t_{k+1}} + \frac{\breve{c}_r^{OS}}{t_r} < \frac{c_0^{CS}}{t_0} + \frac{c_n^{OS}}{t_n} + \ldots + \frac{\breve{c}_r^{OS}}{t_r} \\
\Leftrightarrow \quad & \frac{c_0^{OS}}{t_0} + \frac{\breve{c}_k^{OS}}{t_k} < \frac{c_0^{CS}}{t_0} + \frac{c_k^{OS}}{t_k} \\
\Leftrightarrow \quad & \frac{c_0^{OS} - c_0^{CS}}{t_0} + \frac{\breve{c}_k^{OS} - c_k^{OS}}{t_k} < 0 \\
\Leftrightarrow \quad & \frac{c_{sw}^{OS} - c_{sw}^{CS} + \gamma_0}{t_0} - \frac{\gamma_k}{t_k} < 0 \\
\Leftrightarrow \quad & \frac{t_k}{t_0}\big(c_{sw}^{OS} - c_{sw}^{CS} + \gamma_0\big) < \gamma_k
\end{aligned}$$

Damit ist die Grenzfunktion gefunden, ab der sich ein Verschieben aus dem Sammelkontext in einen anderen Kontext lohnt. Mit der Differenz der Schedulingkosten δ_{sw} ergibt sich:

$$\gamma_k > \frac{t_k}{t_0}\big(\delta_{sw} + \gamma_0\big) \tag{8.5}$$

Wie erwartet müssen die Interferenzkosten der niedrig priorisierten Task im Sammelkontext m_n größer sein als die durch das Verhältnis der Perioden neu hinzukommenden Betriebssystemkosten und Interferenzkosten der höher priorisierten Task im Kontext m_l, der dann zwei statt nur noch eine Task enthält.
Diese Herleitung ist ohne weiteres auf eine Situation übertragbar, in der beiden betrachteten Kontexten mehrere Tasks zugeweisen sind. In diesem Fall ist die Grenzfunktion unabhängig von der Differenz in den Taskwechsel-Prozeduren und hängt nur noch vom Verschieben der Interferenzkosten ab. Geht man davon aus, dass die zu verschiebende Task im neuen Zielkontext m_i die niedrigste Priorität hat, ergibt sich:

$$\begin{aligned}
& \frac{c_0^{OS}}{t_0} + \ldots \frac{c_l^{OS}}{t_l} + \frac{\breve{c}_k^{OS}}{t_k} \frac{c_n^{OS}}{t_n} + \ldots \frac{c_{k-1}^{OS}}{t_{k-1}} + \frac{c_{k+1}^{OS}}{t_{k+1}} + \ldots + \frac{\breve{c}_r^{OS}}{t_r} \\
& \qquad < \frac{c_0^{OS}}{t_0} + \ldots \frac{\breve{c}_l^{OS}}{t_l} + \frac{c_n^{OS}}{t_n} + \ldots + \frac{\breve{c}_r^{OS}}{t_r} \\
\Leftrightarrow \quad & \frac{c_l^{OS} - \breve{c}_l^{OS}}{t_l} + \frac{\breve{c}_k^{OS} - c_k^{OS}}{t_k} < 0 \\
\Leftrightarrow \quad & \frac{\gamma_k}{t_k} - \frac{\gamma_l}{t_l} < 0
\end{aligned}$$

Damit ergibt sich nur noch eine Abhängigkeit von Interferenzkosten in den beiden Kontexten:

$$\gamma_k < \frac{t_k}{t_l}\gamma_l \tag{8.6}$$

D.h. ein Schieben von Task τ_k in den anderen Kontext ist nur dann gewinnbringend, wenn die neu hinzukommenden Interferenzkosten der vormals niedrigst priorisierten Task τ_l in dem neuen Kontext m_i geringer sind als die Interferenzkosten, die durch das Verschieben von τ_k aus m_n heraus gewonnen werden (natürlich wiederum abhängig vom Periodenverhältnis, da der Optimierungsparameter die Utilisation ist).

Zu 3.)
In dieser Bedingung sind wiederum 2 Fälle zu unterscheiden, nämlich

a Der Zielkontext der Verschiebung m_i beinhaltet nur eine Task

b Der Zielkontext der Verschiebung m_i beinhaltet mehrere Tasks

Die Rechenschritte sind den obigen sehr ähnlich, so dass hier nur das Ergebnis betrachtet werden soll. Für den Fall a) ergibt sich eine Grenzfunktion von

$$\gamma_k > \frac{t_k}{t_0}\big(\delta_{sw} + \gamma_0\big) - \frac{t_k}{t_r}\delta_{sw} \tag{8.7}$$

Um eine Erniedrigung der Utilisation zu erreichen, müssen also die Interferenzkosten γ_k größer sein als die zusätzlichen Schedulingkosten plus den neu entstandenen Interferenzkosten γ_0 im Kontext m_i minus dem Gewinn bei den Schedulingkosten in m_n.
Für den Fall b) ergibt sich analog:

$$\gamma_k > \frac{t_k}{t_l}\gamma_l - \frac{t_k}{t_r}\delta_{sw} \tag{8.8}$$

Da in diesem Fall im Zielkontext bereits ein Betriebssystem Schedule notwendig war, ergibt sich die Grenze für γ_k aus den neu auftretenden Interferenzkosten in m_i minus dem Gewinn bei den Schedulingkosten in m_n.

Insgesamt ist damit γ_k in Differenz zu den jeweiligen Bedingungen auch ein Maß für die Güte der Verbesserung, falls eine Verschiebung zustande kommt. Aufbauend auf den Kriterien (11.3) bis (11.7) kann nun der Algorithmus (siehe Algorithmus 7) angegeben werden, nach dem eine Taskmenge auf die Kontexte eines multithreaded Prozessors mit statischer Instructioncache Partitionierung verteilt wird.

In Zeile 1 des Algorithmus wird initial mit der linearen Zuordnung aller Tasks zu den Kontexten begonnen, wie sie bereits aus dem letzten Abschnitt bekannt ist. Hieran schließt sich im 2. Schritt die Vertauschungsphase an, in der für jede Task geprüft wird, ob sich gemäß Gleichung (11.3) ein Tauschen in einen Einzelkontext lohnt. Existieren mehrere Möglichkeiten des Tauschens für eine Task, so wird die Güte-Eigenschaft der Differenz zwischen Interferenzkosten und Bedingung (11.3) ausgenutzt und derjenige Kontext ausgewählt, der die maximale Ersparniss erbringt. Ab Zeile 3 beginnt die Verschiebung von Tasks aus dem Sammelkontext m_n in andere Kontexte. Wiederum wird anhand der Güte der Verbesserung, die durch die Differenz von γ_k und der jeweiligen Bedingung (11.4) bis (11.7) zu erhalten ist, entschieden, in welchen Kontext die Task τ_k verschoben wird. Um den Maximalwert des Gewinnes möglicher Verschiebungen zu gewinnen, werden alle Verschiebungen in der Menge M zusammen mit der Güte aufgesammelt. In Zeile 5 schließlich wird für jede Task aus m_n derjenige Kontext ausgewählt, für den die Güte d maximal ist. Die Aufnahme einer Verschiebung in M kann außer dem Fall in Zeile 4 für jede mögliche Verschiebung erfolgen, da im Nichterfüllungsfalle $d < 0$ gilt. Diese Voraussetzung gilt nicht für den Fall, dass beide betroffenen Kontexte mehr als eine Task beinhalten

Algorithmus 7: Zuordnung von Tasks zu Kontexten mit Interferenzen

Voraussetzung: $\theta_i \mapsto m_i, r = |\mathcal{T}| > n$

1: $\forall i \in \{1, \ldots, n-1\} : \theta_i = \{\tau_i\}$,
$\theta_n = \{\tau_j \mid j \in \{n, \ldots, r\}$
do
2: **for** $k = n$ **to** r **do**
if $\exists \gamma_k > \left(\left(\frac{t_k}{t_j} - 1\right)\delta_{sw} + \frac{t_k}{t_j}\gamma_j\right)$ mit $\tau_j \in \theta_l \wedge l \in \{1, \ldots, n-1\}$ **then**
Tausche τ_k mit τ_j, für die $\gamma_k - \left(\left(\frac{t_k}{t_j} - 1\right)\delta_{sw} + \frac{t_k}{t_j}\gamma_j\right)$ maximal ist
end if
end for
3: **for all** $\tau_k \in \theta_n$ in Reihenfolge ihrer Prioritäten **do**
$M := \emptyset$
for $j \in \{1, \ldots, n-1\}$ **do**
if $|\theta_n| > 2$ **then**
if $|\theta_j| > 1$ **then**
4: **if** $\gamma_k < \frac{t_k}{t_l}\gamma_l$ mit $\theta_j = \{\ldots, \tau_l\}$ **then**
$M := M \cup \left\{\left(j, \gamma_k - \frac{t_k}{t_l}\gamma_l\right)\right\}$ mit $\theta_j = \{\ldots, \tau_l\}$
end if
else
$M := M \cup \left\{\left(j, \gamma_k - \frac{t_k}{t_l}\left(\delta_{sw} + \gamma_l\right)\right)\right\}$ mit $\theta_j = \{\tau_l\}$
end if
else
if $|\theta_j| > 1$ **then**
$M := M \cup \left\{\left(j, \gamma_k - \frac{t_k}{t_l}\gamma_l + \frac{t_k}{t_r}\delta_{sw}\right)\right\}$ mit $\theta_j = \{\ldots, \tau_l\}$
else
$M := M \cup \left\{\left(j, \gamma_k - \frac{t_k}{t_l}(\delta_{sw} + \gamma_l) + \frac{t_k}{t_r}\delta_{sw}\right)\right\}$ mit $\theta_j = \{\tau_l\}$
end if
end if
end for
if $\exists (l, d) \in M : d > 0$ **then**
5: $\theta_l := \theta_l \cup \{\tau_k\}$, für den d mit $(l, d) \in M$ maximal ist
$\theta_n := \theta_n - \{\tau_k\}$
end if
end do
until (no_change)

(vgl. Kriterium (11.5)), da in diesem speziellen Fall eine Kleiner-gleich-Beziehung zu γ_k gilt. Dieser Umstand wird in Zeile 4 des Algorithmus gesondert abgefangen. In dem Fall, dass keine Verschiebung einer Task aus m_n heraus eine Verringerung der Utilisation zur Folge hat, werden in M alle Güten einen Wert kleiner 0 aufweisen.

8.5 Evaluation

Grundsätzliches Ziel der Einführung von Multithreading ist es, die Gesamtutilisation einer Echtzeit-Applikation dadurch abzusenken, dass die Anteile des Betriebssystems an der Prozessorauslastung drastisch reduziert werden. Das Beispiel in Kapitel 7.3 (Tabelle 7.1) zeigt bereits auf, welche Performanzgewinne in extremen Fällen zu erwarten sind. Diese Analyse soll hier nun auf eine allgemeine Aussage erweitert werden, indem Mengen von Tasksystemen mit charakeristischen Profilen im Hinblick auf den Einsatz von Multithreading untersucht werden. Dazu werden jeweils Sätze von 40 synthetischen Tasksätzen erzeugt, die in ihrer Verteilung der Perioden jeweils eine charakteristische Häufigkeit annehmen und damit unterschiedliche Einsatzprofile abdecken, die unterschiedliche Performanzgewinne erwarten lassen. Jedes Tasksystem besteht aus einer zufällig ausgewählten Taskanzahl zwischen 15 und 25 Tasks, deren Perioden gemäß des charakteristischen Häufigkeitsprofils zufällig ermittelt werden. Die Ausführungszeit jeder Task wird aus dem Anteil dieser Task an der mittleren Utilisation des Gesamtsystems (als Basisutilisation wurde hier 0.6 gewählt) mit einer Abweichung um ± 30% zufällig errechnet. Basierend auf diesem Wert und den Werten für die zu analysierende Architektur des Instruction-Caches (Linesize, Miss-Penalty, etc.) wird mit einem zufälligen Einfluss die Anzahl der Lines bestimmt, die diese Task im Cache benötigt. Dabei wird davon ausgegangen, dass ca. 70% der Ausführung innerhalb von Schleifen verbraucht wird. Ein zufällig gewählter Anteil von 30% - 70% dieser Lines wird als Interferenzkosten einer jeden Task festgesetzt.
Architekturbasis für die folgende Evaluation soll ein Prozessor mit 40MHz Takfrequenz und einem 64KB großen Instruction-Cache sein. Die Miss-Penalty soll 4 Zyklen betragen und es kommt eine Critical-Word-First Strategie zum Einsatz. Die Größe des Instruction-Caches gilt für den Prozessor ohne Multithreading. Beim Einsatz von mehr als einem Kontext wird dieser Cache statisch in Anzahl der Kontexte Teile aufgeteilt, die dann entsprechend eine kleinere Größe aufweisen.
Für die Evaluation des Betriebssystems sollen zwei Varianten untersucht werden: Als Vertreter für ein hochoptimierte, auf die Applikation zugeschnittenes RTOS werden die Angaben für ISRs und Scheduleraufrufe von $c_{ISR} = 19\mu s$ und $c_{Sched} = 22\mu s$ verwendet. Die worst-case Zeiten von $c_{ISR} = 125\mu s$ und $c_{Sched} = 100\mu s$ dienen als Referenz für ein durchschnittliches Standard-Echtzeitbetriebssystem, beispielsweise VxWorks oder POSIX.
In den folgenden Untersuchungen wurde eine Kontextanzahl zwischen 2 und 32 gewählt, wobei die Anzahl aufgrund der technischen Limitierungen der Cache-Partitionierung und Kontextauswahl immer auf eine Zweierpotenz beschränkt ist. Die Folgen der Cachepartitionierungen sind in die WCETs der einzelnen Tasks entsprechend der jeweiligen Cachegröße eingearbeitet, indem gemäß der Line-Anzahl die Miss-Rate erhöht wurde und dies multipliziert mit der Miss-Penalty der WCET aufaddiert wurde. Damit wird ein realistisches Abbild des Effektes verkleinerter Caches gewonnen und die Messungen werden nicht durch unrealistische Annahmen verfälscht. Bei der Analyse der Cache-Interferenz wurde ähnlich vorgegangen, da verkleinerte Caches auch bedeuten, dass die Anzahl der interferierenden Lines sinkt, sobald sie die Größe des Caches übersteigt.

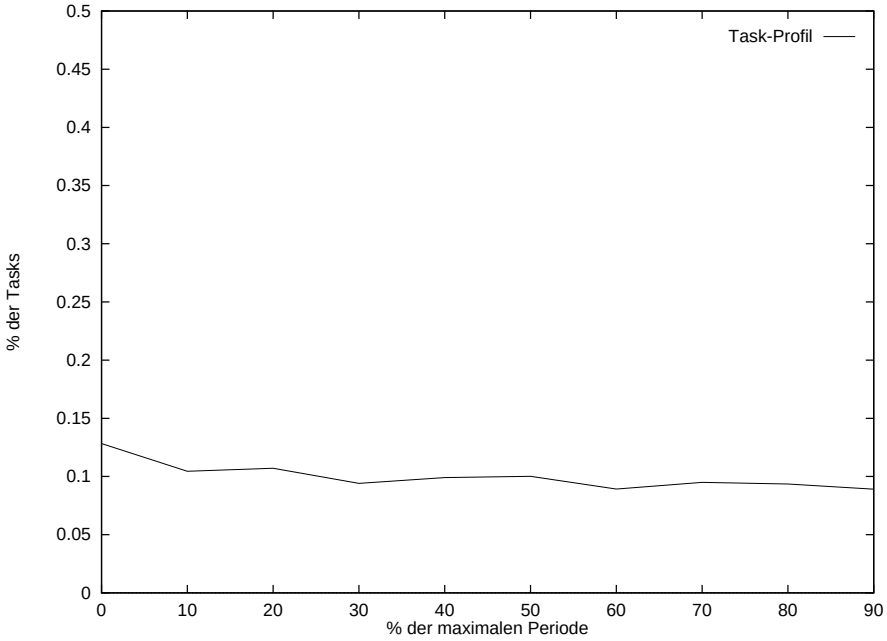


Abbildung 8.5: Task-Profil der synthetisierten Benchmark-Sätze. In diesem Fall sind die Tasks über das Intervall der minimalen bis zur maximalen Periode des Systems gleichverteilt.

Um unterschiedliche Anwendungsszenarien zu betrachten, wird zunächst ein lineares Verteilungsprofil der Tasks über dem Intervall der Perioden des Systems untersucht. Abbildung 8.5 auf Seite 278 verdeutlicht, dass die Tasks nahezu gleichmäßig verteilt sind, so dass in jedem 10%-Intervall (gemessen an der maximalen Periode) etwa 10% der Tasks liegen.

Diese Tasksysteme werden nun mit den OSEK-Kosten (Abbildung 8.6 auf Seite 279) und den Kosten für einen Einsatz des Standard-Betriebssystems (Abbildung 8.7 auf Seite 279) untersucht. Dargestellt ist der Anteil der Utilisation unter Verwendung von Multithreading mit unterschiedlicher Kontextanzahl an der Utilisation des Systemes in einem Prozessor ohne Multithreading. Zur Verdeutlichung des Einflusses der Cache-Interferenz ist jeweils eine Messkurve ohne die Interferenzkosten und eine mit Interferenzkosten dargestellt. Die eingesparte Prozessorauslastung ist erheblich, besonders in dem Standard-Betriebssystemen und erreicht 23% bzw. 69%. In den Bereichen mit 16 und 32 Kontexten stagniert der Performanzgewinn aus zwei Gründen. Zum einen bewirkt eine weitergehende Vergößerung der Kontextzahl eine drastische Reduzierung der Cachegrößen für jeden Kontext. Das wiederum hat, wie eingangs dargestellt, eine erhöhende Wirkung auf die WCET der Tasks. Zum anderen ist bei den hier betrachteten Tasksystemen bei 32 Kontexten jede Task mit einem Kontext versorgt, so dass keine weitere Reduzierung der Prozessorauslastung durch mehr Kontexte zu erwarten ist. Wie zu erwarten war, wird der Einfluss der Cacheinterferenz mit steigenden Kontextzahlen geringer, da die Interferenz nur noch lokal auf den Kontext wirkt, in dem die auslösende Task platziert ist.

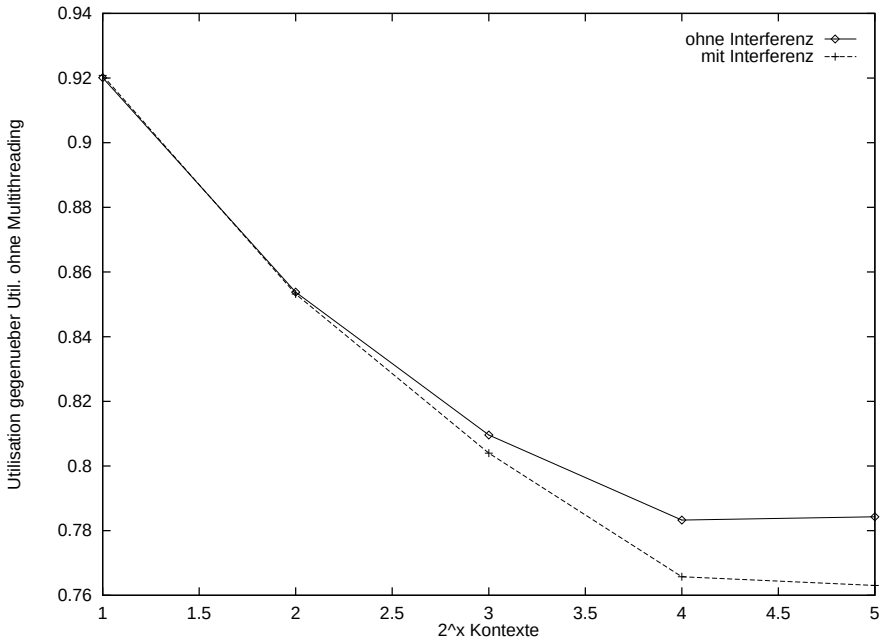


Abbildung 8.6: Performanzgewinn mit und ohne den Einfluss von Cacheinterferenzen durch die Einführung von Multithreading. Hier wurde das RTOS OSEK[208] eingesetzt. Das Task-Profil aus Abbildung 8.5 wurde verwendet.

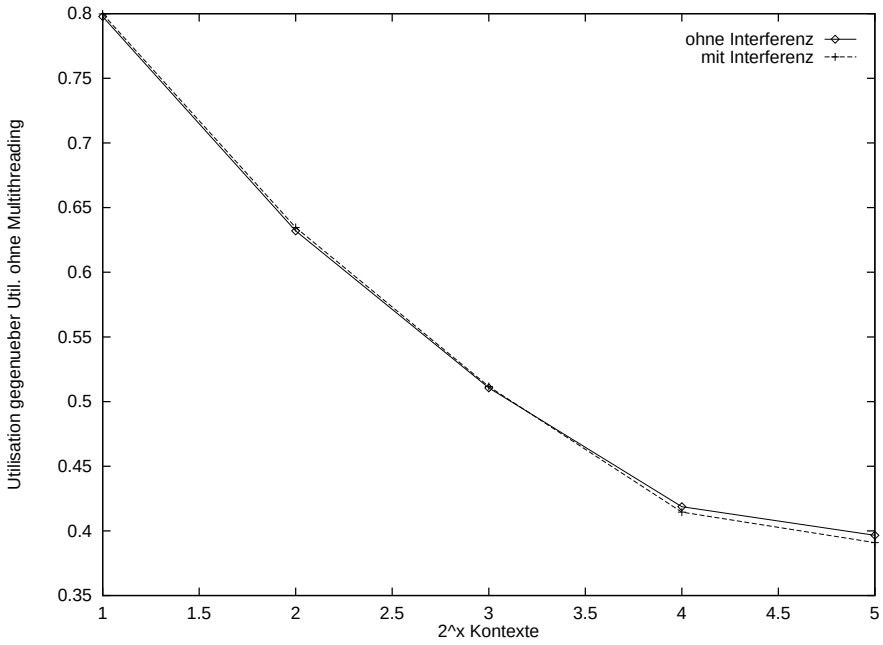


Abbildung 8.7: Performanzgewinn durch die Einführung von Multithreading mit dem Standard-RTOS (Task-Profil aus Abbildung 8.5).

Alle Messungen in den Abbildungen 8.6 und 8.7 basieren auf der Linearen Zuordnung gemäß Algorithmus 6. Da dieser Algorithmus unter der Betrachtung von Interferenzen nicht optimal sein muss, ist in Abbildung 8.8 auf Seite 280 der Performanzgewinn dargestellt, den eine Anwendung von Algorithmus 7 auf die gleichen Tasksysteme erzielt.

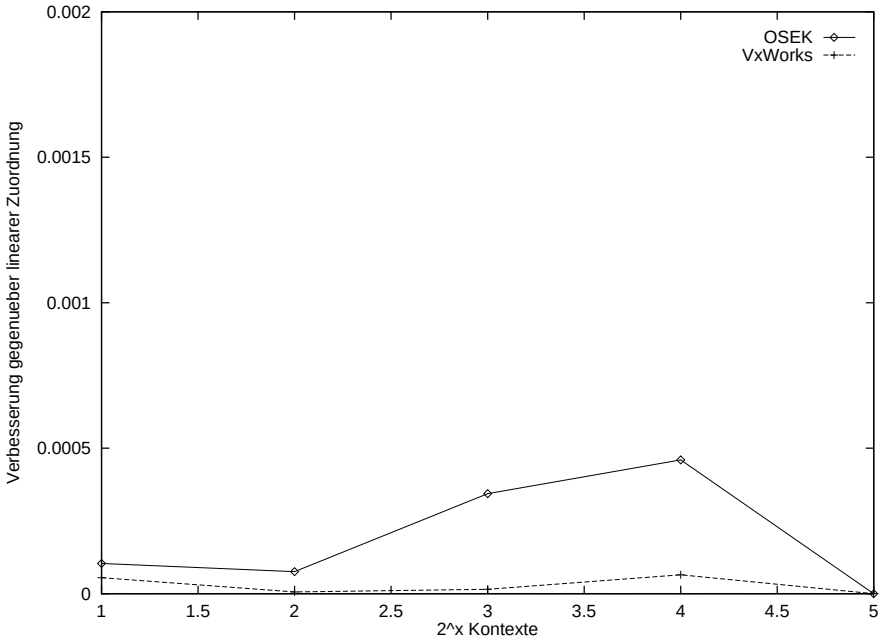


Abbildung 8.8: Performanzgewinn durch die Anwendung des Algorithmus 7 mit Cache-Interferenzen auf sowohl OSEK- als auch Standard-Betriebssystem-basierte Tasksysteme.

Die Reduzierung der Auslastung erreicht im besten Fall gerade einmal 0.05% gegenüber der Verteilung mit der linearen Zuordnung. Dies liegt im wesentlichen an der Struktur der Grenzwertfunktionen (11.4) bis (11.8), die für eine Tauschaktion fordern, dass der Gewinn durch Interferenzvermeidung den Verlust durch Verlagerung von Hardware-Schedules in RTOS-Schedules übertrifft. Aufgrund der hohen Kosten für einen Software Scheduler findet der Tausch gemäß der angegebenen Kriterien höchst selten statt und wenn, dann liegen die Interferenzkosten nur geringfügig über denen eines RTOS Scheduleraufrufes. Damit ist auch der viel höhere Performanzgewinn bei Applikationen mit den OSEK-Zeiten zu erklären.
Insgesamt zeigt sich, dass die lineare Zuordnung in den allermeisten Fällen ausreichend ist. Gleichwohl ließe sich natürlich ein Tasksystem erzeugen, dass nach einer Anwendung von Algorithmus 7 verlangt, um Optimalität zu gewährleisten.

Als nächstes wird ein Task-Profil untersucht, in dem die Verteilung der Tasks nicht gleichmäßig über das Intervall der Perioden erfolgt, sondern in dem es eine Häufung bei hochfrequenten und eine Häufung bei niedrig frequenten Tasks gibt. Abbildung 8.9 auf Seite 281 zeigt diese Häufigkeitsverteilung: In den unteren bzw. oberen Periodenintervallen sind etwa 30% der Tasks angesiedelt, im restlichen Bereich gleichverteilt die übrigen Tasks.
Auch hierfür werden wiederum die Perfomrmanzgewinne bzgl. OSEK-basiertem und Standard-Betriebssystem-basiertem Scheduling, sowie die Auswirkungen des Algorithmus 7 betrachtet.

Die Reduzierung der Utilisation des Prozessors beträgt in diesen Fällen zwischen

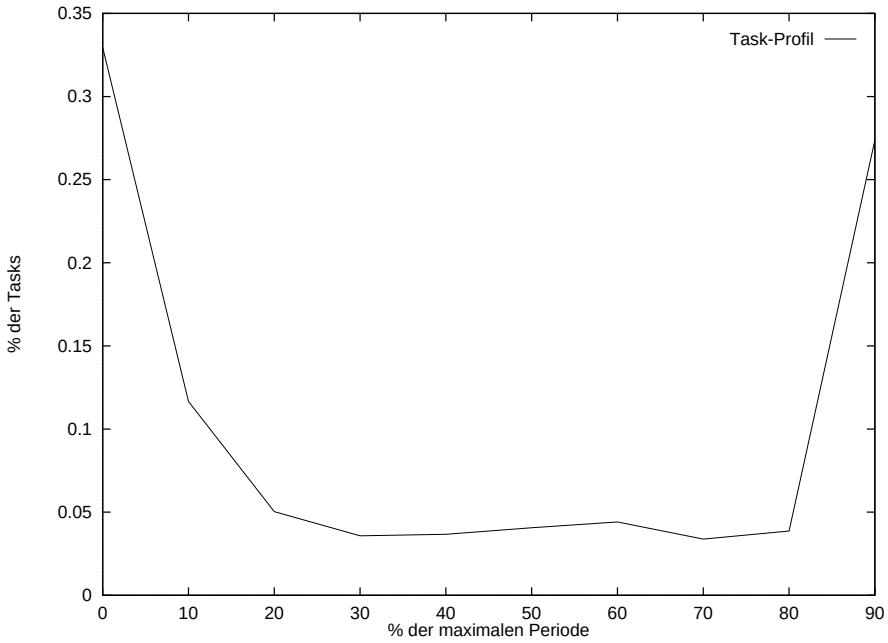


Abbildung 8.9: Task-Profil der synthetisierten Benchmark-Sätze. In diesem Fall gibt es jeweils eine Anhäufung von Tasks an den minimalen und maximalen Intervallgrenzen der Periode, dazwischen existiert eine Gleichverteilung.

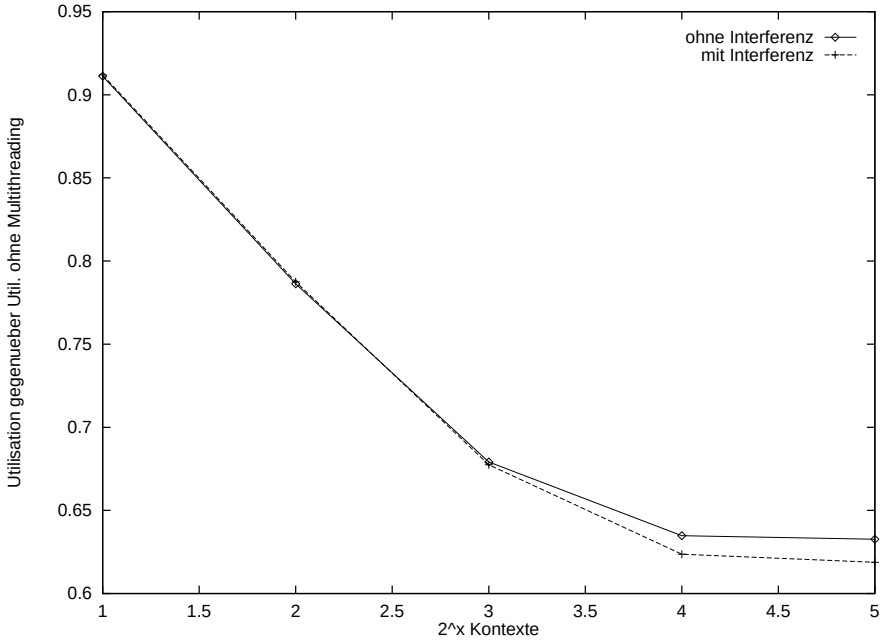


Abbildung 8.10: Performanzgewinn durch die Einführung von Multithreading mit RTOS OSEK[208] (Task-Profil aus Abbildung 8.9).

36% und 76% und übertrifft damit die des geraden Task-Profils deutlich. Dies wird in erster Linie durch die große Anzahl an hochfrequenten Tasks verursacht, die aufgrund ihres häufigen Auftretens einen deutlich größeren Anteil des Betriebssystems an der Prozessorauslastung nachsichziehen. Die Anwendung von Algorithmus 7 hat wiederum nur einen geringen Effekt.

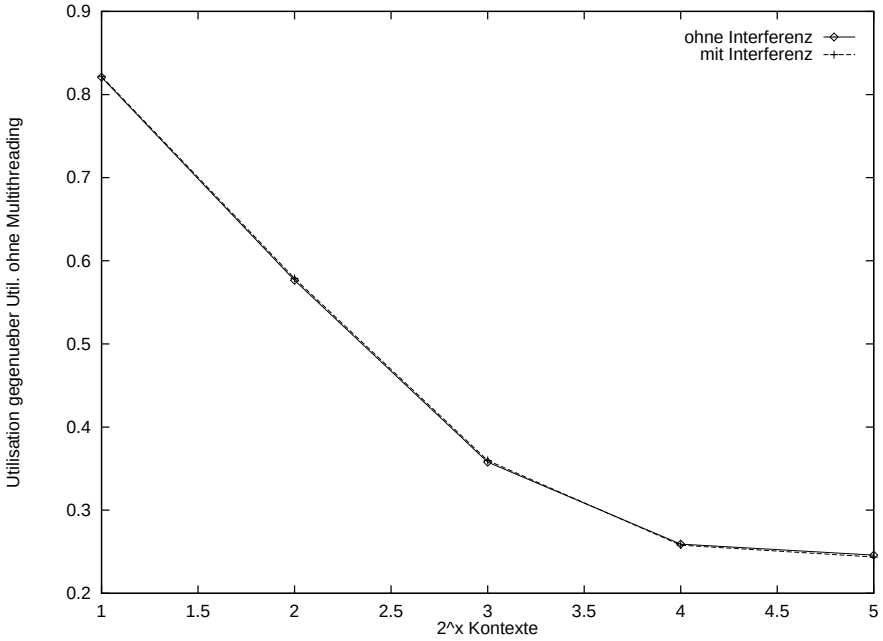


Abbildung 8.11: Performanzgewinn durch die Einführung von Multithreading mit Standard-RTOS (Task-Profil aus Abbildung 8.9).

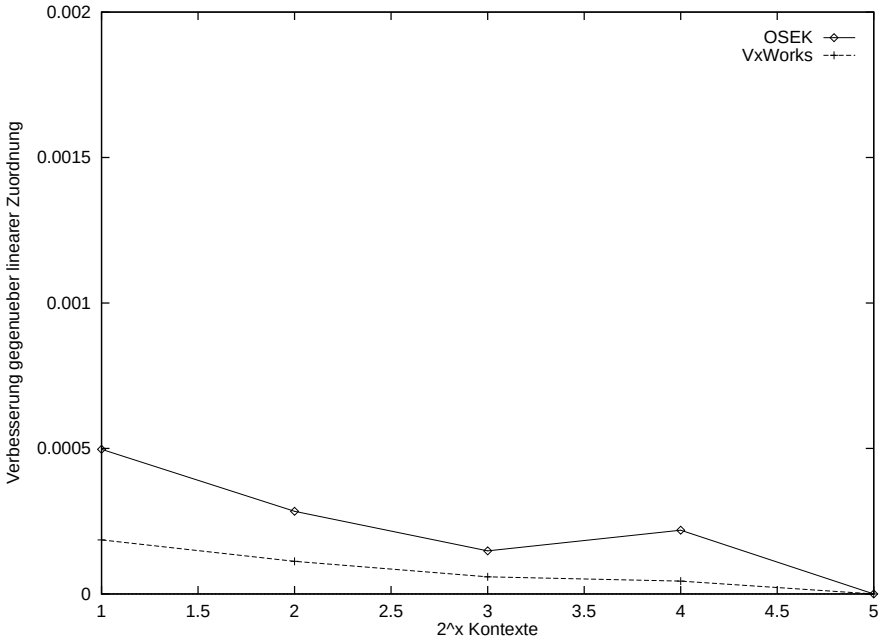


Abbildung 8.12: Performanzgewinn durch die Anwendung des Algorithmus 7.

Als letztes soll ein Task-Profil betrachtet werden, bei dem nur eine größere Anzahl hochfrequenter Tasks auftritt und alle übrigen Tasks gleichverteilt über dem Intervall der Perioden liegen (siehe Abbildung 8.13 auf Seite 283).

Die Reduzierung der Utilisation ist in diesem Fall noch etwas größer und beträgt zwischen 41% für OSEK-basierte Applikationen und 78% für solche, die auf dem Standard-Betriebssystem basieren (siehe die Abbildungen 8.14, 8.15 und 8.16 auf

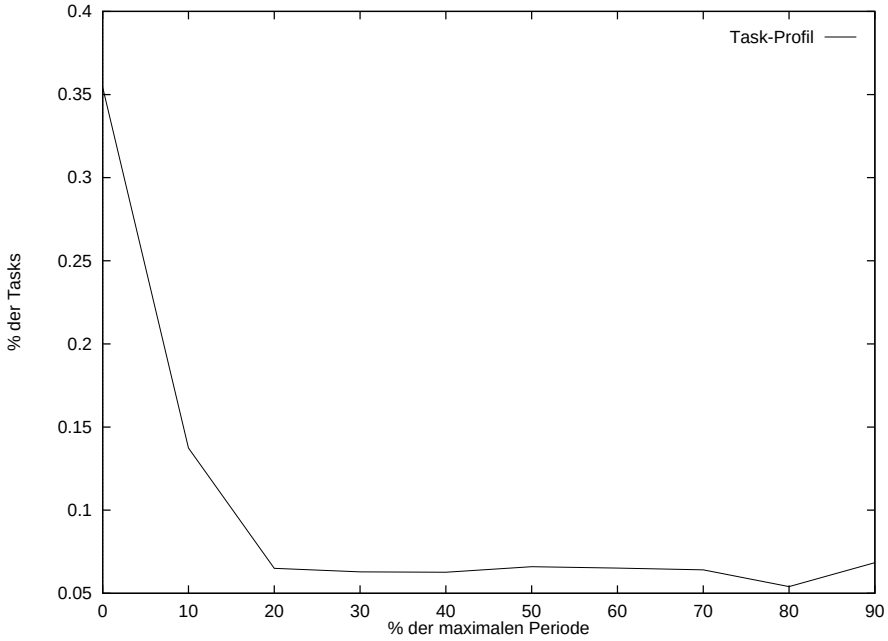


Abbildung 8.13: Task-Profil der synthetisierten Benchmark-Sätze. In diesem Fall existiert eine Anhäufung von Tasks mit niedrigen Perioden.

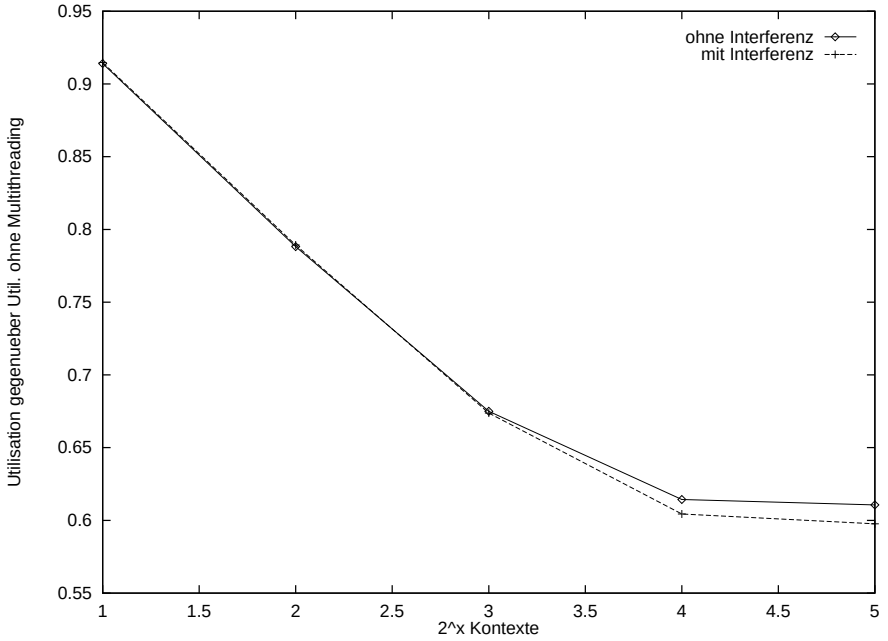


Abbildung 8.14: Performanzgewinn durch die Einführung von Multithreading mit RTOS OSEK[208] (Task-Profil aus Abbildung 8.13).

Seite 283 und folgende).

Insgesamt zeigt sich, welch ein großes Potential in der Verwendung von Multithreading steckt. Dies gilt gerade für eine wachsende Anzahl an Kontexten, obwohl die Erhöhung der Kontextzahl eine Verkleinerung des Caches bedingt. Die Ergebnisse zeigen auch, dass Auslastungsreduzierungen in der Größenordnung, wie sie für das Beispiel in Kapitel 7.3 vorhergesagt wurden, durchaus auftreten können. Es erweist

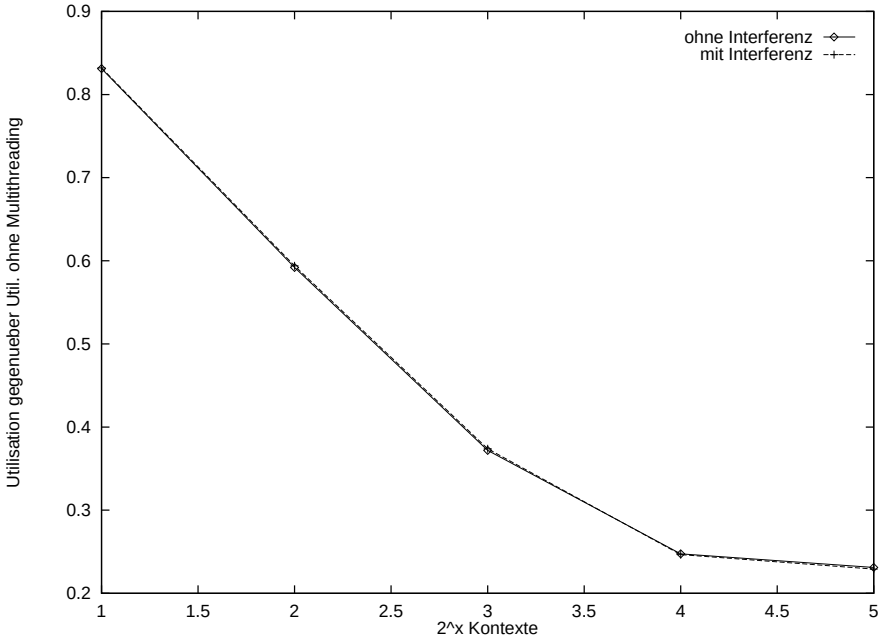


Abbildung 8.15: Performanzgewinn durch die Einführung von Multithreading mit Standard-RTOS (Task-Profil aus Abbildung 8.13).

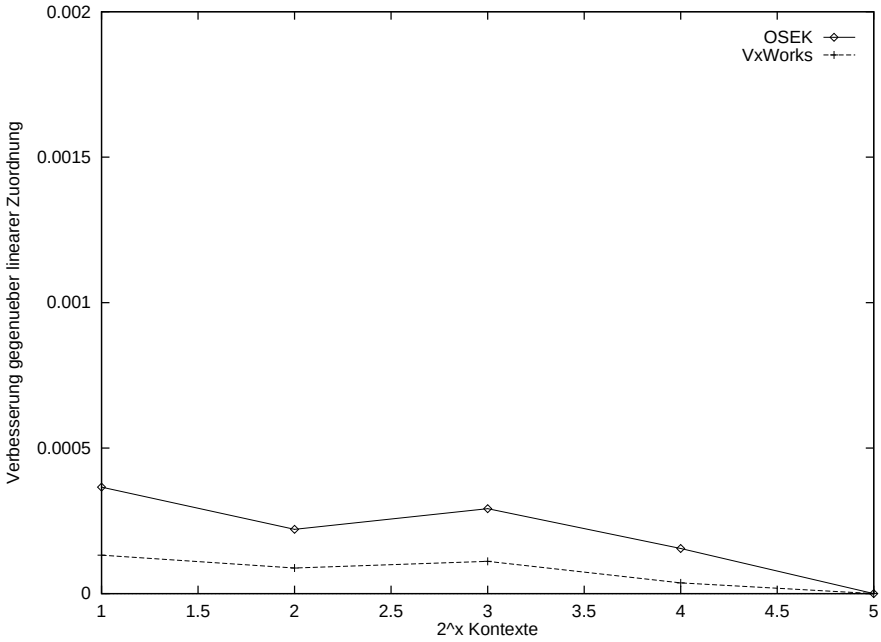


Abbildung 8.16: Performanzgewinn durch die Anwendung des Algorithmus 7.

sich allerdings als abhängig vom typischen Task-Profil einer Applikation, mit welchen Gewinnen gerechnet werden kann. Zudem zeigt sich, dass die Ergebnisse der linearen Zuordnung zumindest mit den hier betrachteten allgemeinen Systemparametern nur minimal durch den erweiterten Algorithmus verbessert werden können.

8.6 Integration in die automatische Platzierung

Abschließend soll hier noch gezeigt werden, in welcher Art und Weise die Kostenminimierung durch Verwendung von Multithreading in die — in Teil I dieser Arbeit vorgestellte — automatische Platzierung integriert werden kann. Wir werden uns dabei lediglich auf die Betrachtung der Taskwechsel-Kosten des Echtzeit-Betriebssystems beschränken und durch Cache-Interferenzen verursachte Kosten vernachlässigen. Die Grundannahme bei der Zuordnung von Tasks zu Kontexten ist hierbei die Verwendung des Algorithmus 6 auf Seite 266[1], d.h. die höchst priorisierten Tasks werden einzeln den Kontexten zugeordnet, während ein Sammelkontext alle übrigen verbliebenen Tasks auf dem jeweiligen Knoten aufnimmt und mittels eines Echtzeit-Betriebssystems scheduled. Der Einfachheit halber wird im folgenden davon ausgegangen, dass alle Prozessorknoten über Multithreading verfügen, alle über die gleiche Anzahl an Kontexten m verfügen und sowohl die Kontextwechselzeiten als auch die Taskwechselzeiten des Betriebssystem gleich sind. Eine Adaptierung an eine heterogene Architektur mit unterschiedlichen Prozessorknoten ist leicht durch eine differenziertere Bildung der Ungleichungen möglich. Prozessorknoten, die über kein Multithreading verfügen, können benutzt werden, indem die Kontextanzahl auf 1 gesetzt wird. In diesem Fall liefert die nun folgende Modellierung standardmäßig die Kosten des normalen Scheduling eines Echtzeit-Betriebssystems.

Die Grundlagen der Einbindung von Schedulingkosten in die Antwortzeitanalyse gemäß Kapitel 4.2 wurden bereits in Kapitel 8.3 formalisiert und ersetzen den Faktor $\Omega_i(\Pi(\tau_i))$ in der Antwortzeitberechnung aus Gleichung 4.3 auf Seite 49. Es muss folglich zunächst die Modellierung der Antwortzeit als arithmetisches Ungleichungssystem in Gleichung 5.10 auf Seite 133 angepasst werden. Hierzu werden sowohl die Kosten des Dispatchens $\texttt{dispatch}_i$ einer Task τ_i als auch die Kosten des Scheduling höher und niedriger priorisierter Tasks $\texttt{sched}_i^j$ hinzugefügt:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \texttt{r}_i = \texttt{dispatch}_i + \texttt{wcet}_i + \sum_{\forall \tau_j \in \mathcal{T}} \left(\texttt{interf}_i^j + \texttt{sched}_i^j \right) \tag{8.9}$$

Schedulingkosten durch höher oder niedriger priorisierte Tasks treten nur dann auf, wenn diese Tasks auf dem gleichen Prozessorknoten allokiert worden sind und hängen von der jeweiligen Priorität der Task ab. Dies kann analog zur $\texttt{interf}_i^j$ in Gleichung 5.11 auf Seite 134 erfolgen:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\texttt{place}_i \neq \texttt{place}_j) \rightarrow \texttt{sched}_i^j = 0 \tag{8.10}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\texttt{place}_i = \texttt{place}_j) \rightarrow \texttt{sched}_i^j = \texttt{I}_i^j \cdot \texttt{scost}_i^j, \tag{8.11}$$

wobei $\texttt{I}_i^j$ wiederum die maximale Anzahl an Auftreten der Task τ_j während des Ausführungsintervalls von τ_i ist und bereits in Gleichung 5.18 auf Seite 136 definiert wurde. Man beachte hier, dass dies nun auch für niedriger priorisierte Tasks

[1] Dies könnte natürlich durch eine Angabe von zusätzlichen Gleichungen auch ohne Probleme unter die Kontrolle des Optimierungsverfahrens gestellt werden. Da aber der Satz zur Optimalität der linearen Zuordnung unter Interferenzfreiheit nachweist, dass die lineare Zuordnung bereits optimal ist, ist dies hier nicht notwendig.

gilt: Während für diese Tasks die `interf`-Variable auf 0 gezwungen wird, können sie trotzdem Schedulingkosten `sched` hervorrufen.
Die Schedulingkosten `scost` selbst hängen davon ab, ob die potentiell unterbrechenden Tasks eine niedriger oder höhere Priorität haben und ob sie in einem Einzelkontext oder dem Sammelkontext angesiedelt sind. Dementsprechend ist zunächst auszuwerten, in welcher Art von Kontext sich eine Task gemäß der linearen Zuordnung befindet. Dazu führen wir zunächst eine Variable $\mathtt{pp}_i^j$ ein, die den Wert der Prioritätsvariablen $\mathtt{p}_i^j$ der beiden Tasks annimmt, wenn beide auf dem gleichen Prozessorknoten angesiedelt sind, andernfalls 0.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{place}_i = \mathtt{place}_j) \rightarrow \mathtt{pp}_i^j = \mathtt{p}_i^j \tag{8.12}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{place}_i \neq \mathtt{place}_j) \rightarrow \mathtt{pp}_i^j = 0 \tag{8.13}$$

Diese Information kann nun genutzt werden, zu ermitteln, wieviele Tasks auf dem gleichen Knoten höher priorisiert sind, indem eine Aufsummierung dieser Prioritätsvariablen durchgeführt wird:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{plev}_i = \sum_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \mathtt{pp}_i^j \tag{8.14}$$

Das Ergebnis in der Variablen $\mathtt{plev}_i$ drückt den Prioritätslevel einer Task aus, indem es sagt, dass $\mathtt{plev}_i$ Tasks eine höhere Priorität besitzen und führt damit implizit eine Ordnung der Tasks auf einem Prozessorknoten nach ihrer Priorität durch. Diese Information ist notwendig, um innerhalb der linearen Zuordnung von Tasks zu Kontexten anhand der Anzahl der Kontexte entscheiden zu können, welche Task einem Einzel- und welche einem Sammelkontext zugewiesen wurde. Dies wird nun sowohl zur Berechnung der Dispatch-Kosten `dispatch` einer Task genutzt als auch zur Quantifizierung der Kosten des Scheduling `scost` einer Task durch andere Tasks. Zunächst ist bekannt, dass die Dispatch-Kosten einer Task in einem Einzelkontext den Kosten eines Kontextwechsels c_{switch} entsprechen:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} (\mathtt{plev}_i < m - 1) \rightarrow \mathtt{dispatch}_i = c_{switch} \tag{8.15}$$

Befindet sich die Task hingegen in einer Sammelkontext, so muss das zu ihr gehörende Event zunächst mittels einer Interrupt-Service-Routine angenommen und anschließend der Scheduler zum Dispatchen gestartet werden. Demzufolge ergeben sich die Dispatchkosten aus der Summation dieser Zeiten.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} (\mathtt{plev}_i \geq m - 1) \rightarrow \mathtt{dispatch}_i = c_{switch} + c_{ISR} + c_{Sched} \tag{8.16}$$

Damit sind die Dispatchkosten determiniert, und wir werden nun die durch das Scheduling des Echtzeit-Betriebssystems induzierten Kosten quantifizieren. Hierbei ist zunächst nach der Prioritätsbeziehung des betrachteten Taskpaares zu differenzieren. Besitzt die potentiell unterbrechende Task eine höhere Priorität, so werden

abhängig vom Kontext entweder nur Kontextwechselzeiten oder aber Kontextwechselzeiten plus Taskwechselzeiten hierfür veranschlagt. Die Frage nach Einzelkontext oder Sammelkontext kann anhand des Prioritätslevel $\texttt{plev}_j$ der unterbrechenden Task beantwortet werden:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \left(\texttt{p}_i^j \wedge (\texttt{plev}_j < m-1)\right) \rightarrow \texttt{scost}_i^j = 2 \cdot c_{switch} \tag{8.17}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \left(\texttt{p}_i^j \wedge (\texttt{plev}_j \geq m-1)\right) \rightarrow \texttt{scost}_i^j = 2 \cdot (c_{switch} + c_{Sched}) + c_{ISR} \tag{8.18}$$

Ist die potentiell unterbrechende Task hingegen niedriger priorisiert, so kann höchstens das Aktivierungsereignis dieser Task angenommen und der Scheduler aufgerufen werden, ohne dass die Task dispatched wird. Dies ist aber ebenfalls abhängig von der Zuordnung der Tasks zu Kontexten, jedoch ist in diesem Fall entscheidend, ob die Task, deren Antwortzeit analysiert wird, einem Einzel- oder einem Sammelkontext zugewiesen ist. Da die Tasks nach der linearen Zuordnung auf die Kontexte verteilt sind, ist klar, dass ein niedriger priorisierter Task entweder im Sammelkontext angesiedelt ist und damit nur Kosten bei Tasks verursachen kann, die ebenfalls im Sammelkontext liegen, oder aber er liegt in einem Einzelkontext mit niedrigerer Priorität und verursacht damit keine Schedulingkosten (vgl. Implementierung des Hardware-Schedulers in Kapitel 8.2.3 auf Seite 258). Dies führt hier, anders als in Gleichung 8.17 und 8.18, zu einer Betrachtung von $\texttt{plev}_i$ anstatt $\texttt{plev}_j$.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \left(\neg\texttt{p}_i^j \wedge (\texttt{plev}_i < m-1)\right) \rightarrow \texttt{scost}_i^j = 0 \tag{8.19}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \left(\neg\texttt{p}_i^j \wedge (\texttt{plev}_i \geq m-1)\right) \rightarrow \texttt{scost}_i^j = c_{Sched} + c_{ISR} \tag{8.20}$$

Abschließend soll die Anwendbarkeit des hier aufgestellten Ungleichungs-Schemas an einem Beispiel gezeigt werden. Hierzu verwenden wir das in Kapitel 5 schon intensiv evaluierte Beispiel aus [183]. Zur Quantifizierung der Unterschiede wird in einem Optimierungslauf für alle Prozessorknoten das Multithreading ausgeschaltet ($m = 0$). Anschließend wird die gefundene Platzierung in die Probleminstanz übernommen, so dass in einem zweiten Durchlauf keine andere Platzierung gewählt werden kann. Vergleichend dazu folgt eine Optimierung dieser so erweiterten Probleminstanz auf einer Architektur, in der alle Prozessorknoten mit Multithreading ausgestattet sind. Die Anzahl der Kontexte auf jedem Prozessor wird mit $m = 4$, dem Beispiel der MSparc folgend (vgl. Kapitel 8.2), angenommen. Die Differenzen der Antwortzeiten der beiden Läufe in Prozent sind in Tabelle 8.1 zu finden, wobei ein Wert kleiner als 100% angibt, um wieviel die Platzierung mit Multithreading in Bezug auf die Antwortzeit von Tasks performanter ist.

Die Kosten für das Scheduling durch ein RTOS wurden hier mit $50\mu s$ für Interrupt Service Routine und Scheduling angenommen und mit $1\mu s$ für Kontextwechsel

τ_0	96%	τ_1	95%	τ_2	97%	τ_3	98%	τ_4	92%
τ_5	60%	τ_6	74%	τ_7	97%	τ_8	89%	τ_9	96%
τ_{10}	98%	τ_{11}	97%	τ_{12}	96%	τ_{13}	86%	τ_{14}	76%
τ_{15}	95%	τ_{16}	80%	τ_{17}	95%	τ_{18}	95%	τ_{19}	98%
τ_{20}	90%	τ_{21}	94%	τ_{22}	90%	τ_{23}	85%	τ_{24}	81%
τ_{25}	71%	τ_{26}	92%	τ_{27}	83%	τ_{28}	78%	τ_{29}	66%
τ_{30}	86%	τ_{31}	84%	τ_{32}	74%	τ_{33}	90%	τ_{34}	89%
τ_{35}	77%	τ_{36}	94%	τ_{37}	86%	τ_{38}	90%	τ_{39}	72%
τ_{40}	96%	τ_{41}	95%	τ_{42}	94%				

Tabelle 8.1: Verbesserung der Antwortzeiten der Tasks durch den Einsatz von Multithreading in % gegenüber einer Platzierung ohne Multithreading-Technik aber mit — durch das RTOS induziertem — Overhead.

des Multithreaded Prozessors. Die Architektur aus [183] musste mit einer höheren Busgeschwindigkeit ausgestattet werden, da mit der ursprünglichen Übertragungsrate keine valide Lösung mit Berücksichtigung der Schedulingkosten erzielt werden konnte. Tabelle 8.1 zeigt, dass es durchweg Verbesserungen in den Antwortzeiten von Tasks gibt. Dies gilt offensichtlich für niedriger priorisierte Tasks auf den einzelnen Prozessorknoten, aber auch die höher priorisierten Tasks profitieren von einer deutlich geringeren Dispatch-Zeit und weisen Verbesserungen der Antwortzeiten um wenige Prozent auf. Sehr hohe prozentuale Reduzierungen der Antwortzeiten treten bei Tasks mit kurzer WCET, aber hoher Interferenz mit anderen Tasks auf, so dass durch den Einsatz der Multithreading-Technik ein wesentlicher Anteil der Antwortzeit subsumiert werden kann.
Insgesamt ergibt sich im Mittel ein Wert von 87%, d.h. Multihreading verbessert die Antwortzeit in diesem Beispiel um etwa 13%. Diese im Vergleich zu der im letzten Kapitel gemessenen relativ geringen Verbesserung liegt an der mangelnden Diversifizität der Perioden des Tasksystems: Da sehr viele Tasks sehr ähnliche Perioden haben, ist der Einfluss der Schedulingkosten auf die einzelnen Antwortzeiten der Tasks als eher gering anzusehen. Dementsprechend zahlen sich weitere Einsparungen in dieser konkreten Beispielinstanz nur eingeschränkt aus. Es ist hier aber abschließend noch zu bemerken, dass diese Messung keine Interferenzen bzgl. der Cacheinhalte aller Tasks berücksichtigen und daher, bei unveränderter WCET einer jeden Tasks, nur einen Teilaspekt vom Einsatz von Multithreading-Techniken abdeckt. Im Falle einer Integration der Effekte von preemptions-induzierter Kosten sind weitere Steigerungen der Performanz für multithreading-basierte Prozessorknoten in dieser Beispielinstanz zu erwarten.

Kapitel 9

Dynamische Caches

9.1 Sinkende Cachegrößen und ihr Einfluss

Der Einsatz von Multithreading bedingt, wie im letzten Kapitel schon erläutert und am Beispiel der MSparc dargestellt, eine Verkleinerung des Caches. Die statische Aufteilung von Cachefragmenten für jeden Kontext ist dabei gerade im Hinblick auf die Vorhersagbarkeit ein wichtiger Faktor: Könnten sich die Caches der Kontexte überlappen (also eine Aufteilung, die dynamisch zur Laufzeit die Cachegröße bestimmt), wäre die Vorhersagbarkeit der Interferenzfreiheit von Tasks auf unterschiedlichen Kontexten nicht mehr gegeben. Dies würde die Performanz des Einsatzes von Multithreading bzgl. Echtzeit verringern und hat zugleich einen wesentlich erhöhten Aufwand in der Hardware zur Folge.

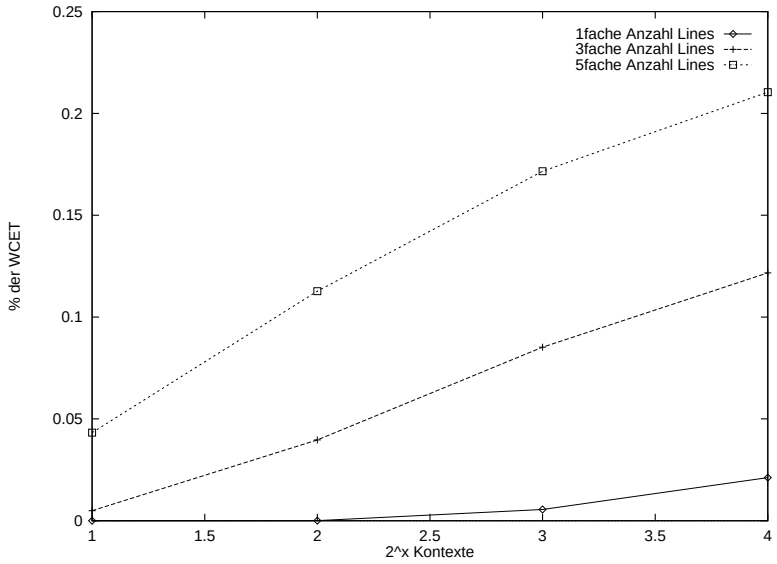


Abbildung 9.1: Anteil der durch verkleinerte Caches induzierten zusätzlichen Cachemisses an der WCET im Mittel.

Andererseits bewirken verkleinerte Caches unter Umständen die Erhöhung der WCET von Tasks, wenn der Workingset der jeweiligen Task nicht mehr in den verkleinerten Cache hineinpasst. In Abbildung 9.1 ist dieser Effekt anhand der im letzten Kapitel

gewählten Benchmarks dargestellt. Die Tasksätze der Benchmarks wurden hierfür um zusätzlich benutzte Lines pro Task in 2 Stufen erhöht, da in den ursprünglichen Benchmarks die Workingsets aufgrund ihrer kleinen Größe in vielen Fällen auch in den verkleinerten Caches unterzubringen waren (die untere Kurve in Abbildung 9.1 verdeutlicht diesen Effekt). Insgesamt zeigt sich die zu erwartende Steigerung des Anteiles von Cache Misses an der WCET einer Task mit steigender Anzahl an Kontexten. Die WCET selbst wurde in der Betrachtung um die zusätzliche Miss-Penalty angehoben, so dass bei einer großen Anzahl an Kontexten die Performanz einzelner Tasks aufgrund der kleinen Instruction Caches deutlich reduziert wird. Diese Effekte sind zwar in Kapitel 8 berücksichtigt, ihre Auswirkungen sind dort aber nicht differenziert betrachtet worden.
Eine weitere Beobachtung hängt direkt mit der Periode einer Task zusammen: Hochfrequente Tasks besitzen im Allgemeinen eine niedrige WCET, um ihre jeweilige Deadline einhalten zu können. Kleinere Laufzeiten wiederum kommen meist mit deutlich kleineren Codegrößen einher und benutzen mithin eine deutlich kleinere Anzahl an Cache Lines für den Instruktionsraum. D.h. der Verlust an Performanz bei verkleinerten Instruktions Caches tritt nicht auf. Vielmehr könnten für Tasks mit diesen Eigenschaften durchaus noch kleinere Caches verwendet werden, ohne die jeweilige WCET signifikant zu vergrößern. Dies führt zu dem Ansatz, die Fragmente des Instruktions Caches nicht gleichmäßig über die Kontexte zu verteilen, sondern gestaffelt nach der Bedürftigkeit eines jeden Kontextes.

9.2 Dynamische Cachegrößen

Eine ungleichmäßige Verteilung von Instruktions Cache Fragmenten auf die einzelnen Kontexte kann jedoch nicht zur Entwicklungszeit eines multithreaded Prozessors festgelegt werden, da die Bedürftigkeit eines Kontextes nach Cacheplatz inhärent durch die auf diesem Kontext platzierten Applikationstasks festgelegt wird. Durch diese Applikationsabhängigkeit verbietet sich eine statische Aufteilung, sondern diese muss sich dynamisch der jeweiligen Applikationsbedürfnisse anpassen können. Da sich rein dynamische Ansätze aus den oben dargelegten Gründen nicht anbieten, ist ein rekonfigurierbarer Ansatz für die Cacheaufteilung zu wählen: Jede Applikation benötigt die Möglichkeit, zur Initial-Phase des Systems die benötigte Cachekonfiguration zu wählen (beispielsweise als Teil eines parametrisierbaren Echtzeitbetriebssystems wie OSEK). Ziel dieser Konfiguration sollte es sein, jedem Kontext soviel Cachefragmente zuzuteilen, dass die Gesamtperformanz, also beispielsweise bezogen auf die Prozessorauslastung, minimal wird. Die Größen der Cachepartitionen können jedoch nicht frei gewählt werden, da in der Varianz der Größen beliebige Werte die Komplexität der Organsisation und Implementierung eines Caches drastisch erhöhen. Dieses kann einen gravierenden Einfluss auf die letztlich zu erreichende Taktfrequenz des Prozessors haben.
Um diesem Dilemma zu entgehen, werden im Kontext dieser Arbeit nur Cachefragmente zugelassen, die in ihrer Größe einer Zweierpotenz entsprechen. Dies ermöglicht eine nahezu äquivalente Organisationsform zu üblichen Caches und vereinfacht auch die Zugriffstrukturen mit den damit einhergehenden Vorteilen bzgl. der Takfrequenz.

Ein wesentlicher Parameter ist die Fragmentierung des Caches, also in wieviele kleine Fragmente der Cache zerteilt wird, die dann in bestimmten Portionen den jeweiligen Kontexten zugeordnet werden. Ein Erhöhung der Fragmente kann prinzipiell die Flexibilität der verschiedenen Cachegrößen erhöhen, hat aber den grundsätzlichen Nachteil, dass mit einer steigenden Anzahl an Fragmenten eine tiefere Hierarchie beim Selektionsprozess des Fragmentes notwendig ist. Tiefere Selektionshierarchien werden in der technischen Realisierung i.a. durch Multiplexerhierarchien gelöst, die wiederum einen Einfluss auf die mögliche Taktfrequenz des Prozessors haben können[1].

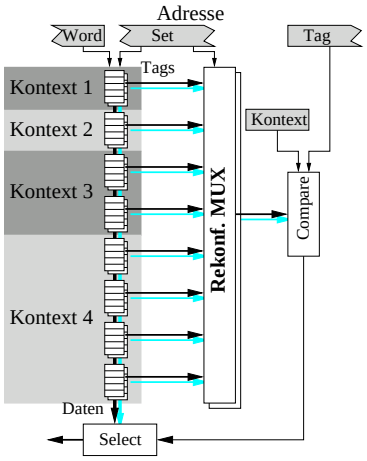


Abbildung 9.2: Architektur mit einer beispielhaften Fragmentierung für eine rekonfigurierbare zweifach assoziative Cache-Architektur mit 8 Fragmenten und 4 Kontexten.

Die Orientierung an Zweierpotenzen für die Größe der Fragmente erlaubt die Verwendung einfacher Multiplexer für die Selektion. In Abbildung 9.2 ist eine Architektur nach diesem Schema dargestellt. Eine mögliche Partitionierung ist ebenfalls mit angegeben. Sowohl die Tag-Einträge der Adresse als auch die gelieferten Daten der Cachefragmente (abhängig von den restlichen Anteilen an der Adresse) müssen entsprechend dem aktiven Kontext selektiert werden. Für applikationsspezifische Konfigurationen ist der Multiplexer der Tag-Einträge beim Systemstart konfigurierbar. Entsprechend der Konfiguration des Caches sind neben dem aktiven Kontext zusätzlich noch Teile der Adresse notwendig, um das korrekte Fragment an den Tag-Vergleicher durchzuleiten. Variationen in der Fragment-Anzahl pro Kontext erfordern ein variables Verschieben der Tag- und Set-Anteile innerhalb der Adresse bei verschiedenen Kontexten (siehe Abbildung 9.3).

Die Anzahl der Fragmente wird hierbei beim Entwurf der Architektur fest gewählt, ebenso wie die Anzahl der Kontexte statisch zur Entwurfszeit des Prozessors festzulegen ist. Dadurch wird eindeutig bestimmt, welche Bits der Adresse neben dem

[1]Sollte der Zugriff auf den Cache nicht im kritischen Pfad des Prozessors liegen, wie es z.B. bei der MSparc der Fall ist, so ist die Taktfrequenz des Prozessors nicht beeinträchtigt.

Abbildung 9.3: Eine größere Anzahl an Fragmenten in einem Kontext vergrößert den Set-Teil der Adresse und verkleinert damit den Tag-Teil.

aktuellen Kontext zur Selektion eines Fragmentes innerhalb des rekonfigurierbaren Multiplexers beachtet werden müssen. Die Art der Rekonfiguration sollte SRAM-basiert (wie etwa in heutiger FPGA-Technologie[193] üblich) gewählt werden und dabei auf eine schnelle Durchleitung der Tag-Daten optimiert sein. Da Kontexte selten umgestellt werden, ist eine schnelle Rekonfiguration nicht notwendig, solange sie den Takt des Prozessors nicht verlängernd beeinflusst.
Eine Konsequenz aus der variierenden Länge des Tag-Teiles einer Adresse ist, dass der Tag-Speicher der Kontexte in seiner Größe differieren kann. Dem kann durch eine Verbreiterung des Tag-Speichers entgegengewirkt werden, wobei bei größeren Caches (Kontexte mit mehreren Fragmenten als Cache) die nichtbenutzten Bits des Tagspeichers durch die rekonfigurierbaren Anteile herausgefiltert werden können, beispielsweise durch ein Konfigurationsregister, dass bei der Konfiguration der Caches ebenfalls initialisiert wird. Diese Lösung bedeutet jedoch einen erhöhten Flächenbedarf in den Tag-Speichern gegenüber der statischen Cache-Aufteilung, insbesondere wenn die Anzahl der Fragmente hoch und gleichzeitig die Größe der Fragmente klein ist.

9.3 Partitionierung

In Abbildung 9.2 ist eine beispielhafte Fragmentierung des Caches von $\{1, 1, 2, 4\}$ gewählt worden, die deutlich macht, welchen Randbedingungen die Wahl der Fragmentierung unterliegt. Zum einen ist dafür zu sorgen, dass keine Fragmente ungenutzt bleiben, zum anderen muss — aufgrund der Bindung an eine Zweierpotenz in der Cachegröße — bei einer Vergrößerung eines Kontext-Caches dieser jeweils die doppelte Größe haben. Diese dafür zusätzlich benötigten Fragmente müssen anderen Kontexten abgezogen werden. Diese induzierte Verkleinerung muss aber für alle Caches eine Größe einer Zweierpotenz sicherstellen[1]. Es ist also ein Optimierungsproblem zu lösen, dass diejenige Fragmentierung der Kontextcaches berechnet, die die minimale Utilisation des Prozessors für die Applikation liefert. Gleichzeitig kann es auch sinnvoll sein, Tasks von einem in einen anderen Kontext zu tauschen, ähnlich den Vorgängen in Kapitel 8. Anders als dort dargestellt, gelingt es im Fall der dynamischen Caches nicht mehr, einen einfachen Algorithmus zur Berechnung der optimalen Verteilung zu finden. Dies liegt an den vielen sich wechselseitig beeinflussenden Faktoren: Varianzen in der Cachegröße bedeuten unterschiedliche WCETs und unterschiedliche Interferenzkosten. Diese wiederum können besser ausgenutzt werden, wenn die Tasks auf andere Kontexte umsortiert werden, etc. Hier ist also

[1] Andere Größen erlauben keine einfache Selektion durch Multiplexer und verkomplizieren die Organisation des Caches dramatisch.

mit einem der bekannten Optimierungsverfahren eine Lösung zu berechnen (vergleiche auch die Aufstellung bekannter Verfahren in Kapitel 5.1).

Für die Taskverteilung gilt die Formalisierung aus Kapitel 8, wobei diese jedoch die unterschiedlichen Cachegrößen und deren Auswirkungen beachten muss. Die Fragmentierung der Caches kann wie folgt formalisiert werden:
Sei n die Anzahl der Kontexte und F die Anzahl der Cachefragmente. Dann ist $c = \{k_1, \ldots, k_n\}$ eine gültige Konfiguration, wenn gilt:

$$c = \left\{k_1, \ldots, k_n \mid k_i \in \{0, \ldots, l_{max}\}\right\} \wedge \sum_{i=1}^{n} 2^{k_i} = F \tag{9.1}$$

l_{max} bezeichnet die maximale Zweierpotenz, die einem Kontext an Fragmenten zugewiesen werden kann. Da die minimale Anzahl an Fragmenten eines Kontextes 1 (also 2^0) ist, errechnet sich l_{max} durch

$$l_{max} = \left\lfloor \log_2(F - n) \right\rfloor$$

Es bietet sich an, für dieses Optimierungsproblem ein ILP-Verfahren[21] zu verwenden, da diese Verfahren eine optimale Lösung berechnen können und das Optimierungsproblem sowohl hinsichtlich der Cachefragmentierung als auch der Taskverteilung unter Berücksichtigung der Einflüsse unterschiedlich großer Caches als Ungleichungssystem formuliert werden kann. Allerdings erwies sich die Komplexität des Problemes schon bei kleinen Taskmengen und wenigen Fragmenten als zu groß für frei verfügbare ILP-Solver, wie beispielsweise der eingesetzte `lp_solve`[106].
Stattdessen wurde ein Simulated Annealing Verfahren[1] benutzt, eine gültige und möglichst optimale Konfiguration zu berechnen. Wesentlicher Faktor bei diesem stochastischem Verfahren (siehe auch die Ausführungen in Teil I dieser Arbeit) ist die Auswahl einer sogenannten Nachbarschaftslösung, d.h. einer Lösung, die nur geringe Differenzen zur aktuell betrachteten Lösung aufweist. Dies erweist sich im Fall der dynamischen Cachefragmentierung als ausgesprochen schwierig, da die Bedingung 9.1 nur einen Bruchteil der möglichen Kombinationen von $c = \{k_i\}$ als gültig akzeptiert und somit Ähnlichkeiten zwischen Konfigurationen kaum auszumachen sind bzw. weit über den Lösungsraum gestreut liegen können. Als Heuristik für die Evaluation im folgenden Kapitel wurde deshalb der Ansatz gewählt, c als Zahl zur Basis l_{max} anzusehen und eine Nachbarschaftskombination durch einen Additionsalgorithmus zu berechnen. Dabei wird auf die Zahl c solange 1 addiert, bis eine neue gültige Lösung erzeugt wird. Trotz der Sprünge bei Überläufen im Zahlraum von l^*_{max} erweist sich diese Lösung als recht brauchbar.

9.4 Evaluation

Zur Evaluation des Ansatzes wurde eine Reihe von Analysen mit den in Kapitel 8 erzeugten Benchmarks durchgeführt. Dabei werden die Architekturparameter aus Kapitel 8 übernommen und für die Fragmentierung jeweils Messungen mit 16, 64 und 128 Fragmenten vorgenommen.

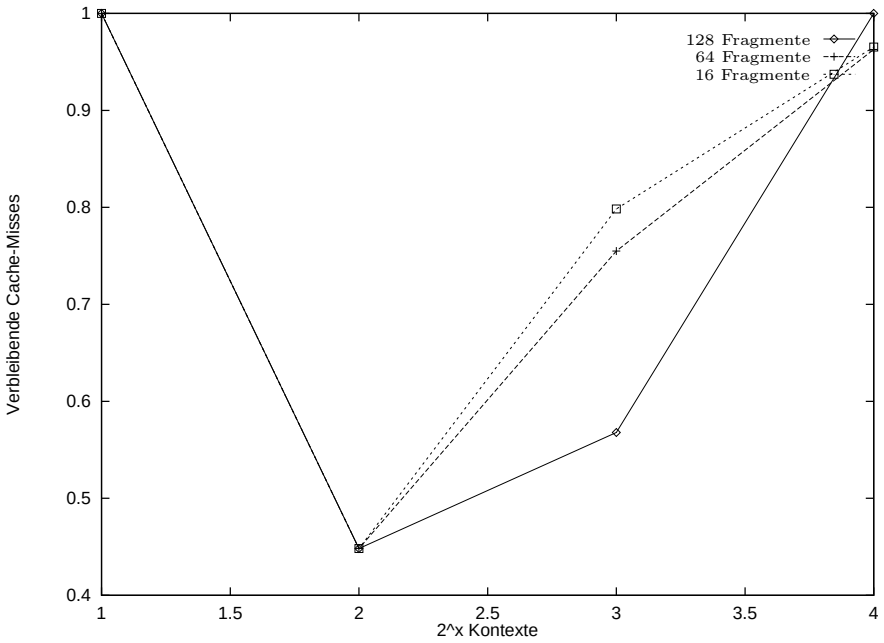


Abbildung 9.4: Eingesparte Instruktions Cache Misses durch den Einsatz dynamischer Caches mit Fragmenten von 16, 64 und 128.

Abbildung 9.4 illustriert beispielhaft den Anteil der eingesparten Cache Misses an den durch Cacheverkleinerung in der statischen Partitionierung — gemäß Kapitel 8 hinzugekommenden — Misses. Als Evaluationsbasis wurde das gerade Task-Profil mit 5facher Cacheline-Benutzung untersucht. Messungen an den anderen Benchmarksätzen mit variierender Anzahl an Cachelines führten zu sehr ähnlichen Ergebnissen.
Obwohl die Reduktion der Cachemisses im Vergleich zur statischen Partitionierung des Instruktions Caches deutlich ist (bis zu 54% der Cache Misses können eingespart werden), macht sich dieser Faktor bei der Betrachtung der Utilisation kaum bemerkbar. Im Mittel ist bei dem in Abbildung 9.4 verwendeten Benchmarksatz lediglich eine Utilisationsreduzierung von etw 2% zu verzeichnen. Dies ist im wesentlichen auf die Utilisation der einzelnen Tasks zurückzuführen. Ersparniss-Effekte treten vorwiegend bei niederfrequenten Tasks aufgrund deren hohen Speicherbedarfes auf. Da die große Anzahl an Tasks der Benchmarksätze eine geringe Utilisation der einzelnen Tasks vorraussetzen und damit die Feasibility des Gesamtsystems sicherstellen, ist der durch die Reduzierung der Missrate erfolgte Performanzgewinn aufgrund der langen Perioden der betroffenen Tasks in der Prozessorauslastung kaum sichtbar. Es bleibt dennoch festzuhalten, dass bis zu 10% der WCET durch diese Technik wieder eingespart werden können. Daher ist bei Tasksystemen mit einer kleineren Anzahl an Tasks — und demzufolge einem höheren Utilisationsanteil der einzelnen Tasks an der Gesamtutilisation — ein deutlicherer Gewinn zu erwarten. Ein weiterer zu berücksichtigender Faktor ist, dass der Spielraum der Cachefragmentierung durch die Bedingung 9.1 stark eingeschränkt ist. Dies erklärt auch das vergleichsweise schlechtere Ergebnis bei einer höheren Fragmentzahl für eine hohe Kontextanzahl, da das verwendete stochastische Optimierungsverfahren hier nicht in der Lage war,

eine bessere Lösung zu finden[1].
Insgesamt scheint die rekonfigurierbare dynamische Cachefragmentierung nur in speziellen Applikationsprofilen geeignet, die Performanz der Appliaktion bzgl. der Prozessorauslastung deutlich zu erhöhen. Die an den Benchmarksätzen durchgeführten Versuche ergeben keinen nennenswerten Gewinn. Es bleibt abzuwarten, ob reale Applikationen eher die Eigenschaften besitzten, die diese Technik und den damit verbundenen zusätzlichen Hardwareaufwand rechtfertigen.
Bei der Taskzuordnung zu den Kontexten ergaben sich nahezu beliebige Kombinationen in den betrachteten Tasksystemen, die allesamt in der Performanz bzgl. des Echtzeit-Betriebssystemeinflusses den Lösungen in Kapitel 8 gleichen. Eine Ersetzung der freien Taskverteilung in der Optimierung durch die lineare Verteilung aus Kapitel 8 ergab keine signifikante Verschlechterung der Prozessorauslastung, so dass auch in diesem Fall die lineare Zuordnung als sehr effektive Heuristik angesehen werden kann[2].

[1]Diese bessere Lösung hätte es gegeben: da die Anzahl der Fragmente in Zweierpotenzen erhöht wurde, ist eine Lösung bei hoher Fragmentzahl, die Fragmente so anzulegen, wie es eine Architektur mit weniger Fragmenten vorgibt. In diesem Sinne ist eine Architektur mit mehr Fragmenten immer mindestens so performant, wie eine mit wenigen (Vorrausgesetzt die Anzahl der Fragmente sind Zweierpotenzen).

[2]Der Optimierungsalgorithmus berücksichtigt diese Vorgaben und erzeugt daran angepasste Cachefragmentierungen.

Kapitel 10

Globale Zusammenfassung

Basierend auf den Beobachtungen der Probleme in der (industriellen) Entwicklung verteilter eingebetteter Echtzeitsysteme wurden in dieser Arbeit Beiträge zur Verwirklichung einer Vision eines durchgängigen Entwurfsprozesses unter spezieller Berücksichtigung von Echtzeiteigenschaften vorgestellt. Dabei galt das Hauptinteresse auf der einen Seite der Steigerung der Effizienz von Entwicklungsprozessen durch die Einführung komponentenbasierter Entwurfsstile und auf der anderen Seite der Steigerung der Effizienz der entworfenen Systeme mit dem Ziel, sowohl die Kosten der Entwicklung als auch der Produktion zu reduzieren und gleichzeitig die Qualität der Implementierung zu erhöhen. Ersteres wurde mittels eines komponentenbasierten Prozesses erreicht, der bezüglich der Echtzeiteigenschaften klar definierte Schnittstellen anbietet, mit deren Hilfe Systeme auf einfache Art und Weise integriert werden können. Zudem wurde ein auf die hier betrachtete Anwendungsdomäne spezialisiertes Integrationsverfahren, die inkrementelle Integration, vorgestellt, die dem Vorkommen von massiven Parallelentwicklungen in einer von Zulieferern geprägten Integration Rechnung trägt. Das Verfahren zur automatischen Platzierung von Tasks und Nachrichten auf eine verteilte Architektur unterstützt diesen Prozess und kann maßgeblich zur Verkürzung von Entwicklungszeiten beitragen. Die Optimalität der durch dieses spezielle Verfahren gelieferten Lösung erhöht dabei die Effizienz der Implementierung und trägt gleichzeitig dazu bei, den nahezu linearen Zusammenhang zwischen Anzahl der Softwarefunktionen und Anzahl der Steuergeräte zu durchbrechen. Spezielle, auf Multithreading basierende, Prozessorarchitekturen können die Effizienz durch Reduktion der durch ein Betriebssystem bedingten Kosten deutlich reduzieren und bieten gleichzeitig eine bessere Vorhersagbarkeit des zeitlichen Verhaltens der in dieser Arbeit betrachteten Echtzeitsysteme an. Dies steigert die Qualität der Implementierung eines verteilten eingebetteten Echtzeitsystems, da hiermit erstmals die Einflüsse eines Echtzeitbetriebssystems quantitativ in einer Zeitanalyse erfasst werden. Dies wird durch die formale Analyse des Zeitverhaltens von Tasks und Nachrichten auf unterschiedlichsten Kommunikationsmedien sowie in nahezu beliebigen hierarchischen Topologien des Steuergerätenetzwerkes unterstützt. Erstmals ist hier auch eine akkurate Analyse von gemischten ereignisbasierten und zeitbasierten Systemen gelungen. Die formalen Zeitanalysen, die die Basis für das automatische Platzierungsverfahren darstellen, ermöglichen eine hohe Qualität der Implementierung und können dazu beitragen, Entwicklungszeiten zu reduzieren,

indem auf komplexe Messungen und Simulationen verzichtet werden kann, die insbesondere in der Phase der Exploration des Entwurfsraumes die intensive Betrachtung von Implementierungsalternativen zumeist verhindern.

Neben dem durch die in dieser Arbeit dargestellten monetären und zeitlichen Einsparpotentiale in der Entwurfsphase komplexer Steuergerätenetzwerke ergeben sich zudem auch (durch eine gesteigerte Effizienz der Systeme erreichte) potentielle monetäre Vorteile in Systemen, die für die Massenproduktion bestimmt sind, wie es etwa auf die Automobilindustrie zutrifft: Die vorgestellte optimale Platzierungsmethodik für Softwarefunktionen kann helfen, die Anzahl der Steuergeräte pro produziertes System zu erniedrigen. Für Unternehmen, die Systeme in millionenfacher Stückzahl herstellen müssen, bedeutet die Einsparung einiger Steuergeräte pro System eine ganz erhebliche Kostenersparnis, zumal im Entwicklungsprozess die Kosten der Softwareentwicklung identisch zu einer Entwicklung für proprietäre Steuergeräte sind und sich somit bei der Reduzierung der Hardware eines Systems auch echte Kostenreduzierungen erreichen lassen. Vorraussetzung hierfür ist, dass die verbliebenen Steuergeräte über die Kapazität verfügen, zusätzliche Funktionen aufzunehmen. An dieser Stelle führt die exaktere und formale Vorhersage des zeitlichen Verhaltens von Echtzeitsystemen zu einer höheren Auslastung, da die herkömmlichen, auf fehleranfälligen Messwerten basierenden, Zeitabschätzungen dazu führen, dass große Sicherheitsmargen geschaffen werden müssen und damit Auslastungen für Prozessoren akzeptiert werden, die unter formaler Betrachtung durchaus noch freie Kapazitäten haben. Unterstützt werden kann dieser Prozess noch durch die Verwendung der in Teil II dieser Arbeit dargestellten Architekturerweiterungen, die eine weitergehende Reduzierung der Auslastung auf Steuergeräten in ganz erheblichen Maße anbieten. Zwar ist der Einsatz derartiger Spezialprozessoren im industriellen Umfeld im Moment nicht zu verzeichnen, allerdings kann sich dies in naher Zukunft durchaus ändern, da sich durch den zunehmenden Preisverfall seitens der Halbleiterindustrie Themen wie System-on-Chip und programmierbare Systeme, wie etwa FPGAs sowie die Kombination dieser beiden Techniken, in absehbarer Zeit auch im Massenmarkt der eingebetteten Systeme durchsetzen werden. Dieser Trend kann letztlich als Vehikel dienen, speziell für Echtzeitsysteme ausgestattete Hardware auch in den hier betrachteten Domänen, etwa der Automobilindustrie, im Massenmarkt zu etablieren.

Trotz der vielen erreichten Verbesserungen des Entwurfsprozesses und der Effizienz verteilter eingebetteter Echtzeitsysteme durch die hier dargestellten Methodiken verbleibt an vielen Stellen noch Bedarf nach weitergehenden Forschungen. Wir haben im Zuge der einzelnen Themenstellung in dieser Arbeit bereits jeweils die wichtigsten möglichen Erweiterungen detailliert erörtert, so dass dies an dieser Stelle nicht nochmals wiederholend erfolgen soll. Stattdessen sollen hier als Ausblick noch diejenigen offenen Fragen aufgezeigt werden, die der Vision eines durchgängigen, effizienten Entwurfsprozesses noch fehlen und deren Bearbeitung substantielle Fortschritte im Bereich der Entwicklung eingebetteter Echtzeitsysteme versprechen. So ist zunächst als übergeordneter Prozess die Integration der Methoden in einen ganzheitlichen Entwurfsprozess notwendig. Als Framework können hierbei sicherlich

die bereits angesprochenen Rich Component Models eine ganz ausgezeichnete Rolle übernehmen. Bei der Exploration des Entwurfsraumes und insbesondere der damit zusammenhängenden virtuellen Integration sind aber neben den hier vorgestellten Techniken, die im wesentlichen den Implementierungsprozess unterstützen, weitergehende abschätzende Techniken erforderlich. Wir haben dies bereits in Kapitel 6 detaillierter diskutiert. Zudem sollte eine Vereinigung mit Fragen des klassischen Hardware-Software-Codesigns erfolgen.
Ein weiterer wesentlicher Faktor der Güte einer Durchmusterung des Entwurfsraumes ist die Frage der Abbildung funktionaler Spezifikationen auf Tasknetzwerke. Hier reicht, je nach Abstraktionsgrad der funktionalen Spezifikation, oftmals eine angereicherte automatische Codegenerierung aus solchen Spezifikationen nicht aus, da gerade in verteilten Systemen das Kommunikationsverhalten der dadurch entstehenden Tasknetzwerke einen ganz entscheidenden Einfluss auf die Güte einer Lösung hat. Dies wiederum wird ganz entscheidend durch die jeweilige Ausprägung von Steuergerätearchitektur und benutzter Bussysteme sowie deren topologische Anordnung bestimmt, so dass hierbei die wechselseitige Beeinflussung dieser beider Faktoren mit berücksichtigt werden muss. Letztlich ist dies wiederum auch abhängig von der gewählten Platzierung und damit von Verfahren wie dem in dieser Arbeit vorgestellten.
Als letzter Punkt in diesem Umfeld soll hier die Komposition von Komponenten bzgl. ihrer Echtzeitschnittstellen genannt werden, die entweder synthetisierend oder überprüfend den komponentenbasierten Entwurfsprozess unterstützen können und sollen, wie dies bereits in Kapitel 4 erörtert wurde.

Seitens der Analyse von Echtzeitsystemen sind insbesondere weitere Arbeiten im Bereich der Preemptionsfolgekosten-Abschätzung unerlässlich. Zwar können diese Kosten durch Techniken, wie wir sie in Teil II vorgestellt haben, abgemildert werden, aber ihre Integration in Systemen, die über keine derartigen Architekturerweiterungen verfügen, fehlt bislang vollkommen. Unter den formalen Gesichtspunkten der Echtzeitanalysen ist dieser Zustand selbstverständlich nicht hinnehmbar. Hier ist die vordringliche Aufgabe, Methodiken zu entwickeln, die es erlauben die durch traditionelle WCET-Analysen gewonnen Werte der Schedulinganalyse zugänglicher zu machen. Beispielsweise könnten in Form von sogenannten Kooperativen Schedulingmethoden WCETs nur bis zu bestimmten Punkten (Invalidierungspunkten) ungestört vorhergesagt werden und dem Scheduling auf der anderen Seite untersagt werden, außerhalb dieser Invalidierungspunkte Tasks zu suspendieren. Dies macht aber spezialisiertere WCET-Analysen zur Vorbedingung und verschlechtert das Zeitverhalten durch zusätzliche Blockierungen in der Echtzeitanalyse. Hier ist ein geeignetes Maß bei der Aufteilung zu finden: Zu wenige Invalidierungspunkte erniedrigen die vorhergesagte WCET einer Task, erhöhen aber die Antwortzeit anderer Tasks durch längere Blockierungen. Andererseits führt eine Steigerung der Anzahl der Invalidierungspunkte zu höheren WCETs (Stichwort Cache- und Pipelineinvalidierung), kann aber die Blockierungszeit in der Antwortzeit andere Tasks reduzieren (bei gleichmäßiger Ausbalancierung der Invalidierungspunkte, was ein zusätzliches großes Problem aufwirft).
Diese Betrachtung führt insgesamt zu der Streitfrage, ob zeitbasierte oder ereignis-

basierte oder aber kombinierte Systeme die Entwürfe der Wahl sind. Der Trend der letzte Jahre zeigt klar in Richtung zeitbasierter Systeme, wobei dies wohl eher an der besseren Vorhersagbarkeit denn an der besseren Performanz dieser Systeme liegt (bei den Bussystemen mag dies anders liegen, da zeitbasierte Systeme elektrisch bedingt deutlich höhere Übertragungsraten aufweisen können). Ein besseres Verständnis mit damit einhergehenden besseren Vorhersagemöglichkeiten sowie ggf. spezialisierter Hardwareunterstützung könnte eine Wiederbelebung der ereignisbasierten Systeme bewirken und sie zumindest in den Bereichen wieder beliebter machen, in denen ihre Flexibilität notwendig ist. Letztlich sollte diejenige Implementierung gewählt werden, die die kostengünstigste ist und dies wird aller Vorraussicht nach auf eine gemischte Variante von zeitbasierten und ereignisbasierten Systemen hinauslaufen, für deren Analyse wiederum diese Arbeit einen wesentlichen Beitrag geliefert hat.

Es zeigt sich letzlich, dass selbst Arbeiten mit einem Umfang wie die hier vorliegende nur ein kleines Mosaiksteinchen in einem weit größeren Bild darstellen. Die Vervollständigung dieses Bildes ist eine der spannendsten und interessantesten Fragestellungen der Informatik und beim Niederschreiben der Zusammenhänge zu dem großen Ganzen ist der Autor geneigt, diesem Thema weitere 380 Seiten zu widmen. Aus naheliegenden Gründen werden wir diesen Vorsatz jedoch zunächst nicht umsetzen und schließen diese Arbeit mit der erwartungsfrohen Neugier auf die noch fehlenden Steinchen in eben jenem Mosaik.

Teil III

Anhang

Anhang A

Abkürzungsverzeichnis

ATM	Asynchronous Transfer Mode, ein spezielles Übertragungsprotokoll im Bereich der Breitbandübertragung
AUTOSAR	AUTomotive Open System ARchitecture, Konsortium von Automobilherstellern und Zulieferern zur Etablierung von mehr Austauschbarkeit im Entwurf elektronischer Steuergeräte
BCET	Best Case Execution Time, Untergrenze der reinen Laufzeit eines Programms einer Task auf einem Prozessor ohne Unterbrechungen oder Betriebssystemkosten
BDD	Binary Decision Diagrams, Darstellungsform für boolesche Aussagenlogik
CAN	Controller Area Network, ein serielles Busprotokoll, welches speziell für den Einsatz in der Automobilindustrie entworfen wurde
CRC	Cyclic Redundancy Check, ein spezielles Verfahren zur Fehlererkennung in Nachrichtenübertragungen
DM	Deadline Monotonic, ein Prioritätenzuweisungsverfahren für Tasksysteme
DSP	Digital Signal Processor, speziell auf die digitale Signalverarbeitung zugeschnittene Prozessoren
E2E	End-to-End-Deadlines, eine Zeitanforderung, die sich über die Ausführung mehrerer Tasks erstreckt
ECM	Embedded Control Mode, spezieller Modus des MSparc-Prozessors zur Kopplung externer Schedulerimplementierungen
ECU	Embedded oder Electronic Control Unit, zentrale Recheneinheit in eingebetteten Systemen, die die Softwarefunktionen aufnehmen
EDF	Earliest Deadline First, ein spezielles Schedulingverfahren, das Prioritäten zur Laufzeit bestimmt
EMI	ElektroMagnetische Interferenzen, die Fehler in den Übertragungen auf einem Bussystem erzeugen können
EOF	End Of Frame, eine spezielle Kennung für das Ende eines Frames im CAN-Bus Protokoll

FIFO	First In First Out, Warteschlangen, in denen die Werte zuerst herausgenommen werden, die am längsten in der Warteschlange eingetragen sind
ILP	Integer Linear Programming, siehe MILP
IPET	Implict Path Enumeration Technique, ein Verfahren zur Traversierung von Ablaufgraphen mittels MILP-Techniken
ISR	Interrupt Service Routinen, Programmfragmente, die die Annahme eines Interrupts auf einem Prozessor bearbeiten
LPOS	Last Point Of Sending, legt in gemischten ereignis- und zeitbasierten Systemen im dynamischen Anteil den Zeitpunkt fest, ab dem die letztmögliche Übertragung gestartet werden kann
MEDL	MEssage Descriptor List, verwaltet die Zuordnung von Nachrichten zu Slots im TTP-Protokoll
MILP	Mixed Integer Linear Programming, eine Variante eines numerischen Entscheidungsverfahrens, dass neben reelwertigen Variablen auch Integer-Variablen zulässt
MSA	Multiple Slot Access, eine TDMA-Variante, in der es innerhalb einer TDMA-Runde den Prozessorknoten erlaubt ist, mehrere Slots zu besitzen
MSPARC	Multithreaded SPARC, eine um multithreading Techniken erweiterte Implementierung eines SPARC-Prozessors
OEM	Original Equipment Manufacturer, im Kontext dieser Arbeit bezeichnet dieser Begriff die Hersteller von komplexen Systemen (Automobil, Flugzeug, etc.)
OSEK	Offene Systeme für die Elektronik im Kraftfahrzeug, ein Standard für ein Echtzeitbetriebssystem, speziell für den Anwendungskontext Automobiltechnik entworfen
RCM	Rich Component Models, ein modellbasierter Entwurfsprozess, der gemäß dem Assume-Guarantee Paradigma einen durchgängigen Entwurf ermöglichen soll
RM	Rate Monotonic, ein Prioritätenzuweisungsverfahren für Tasksysteme
RTOS	Real-Time Operating System, Betriebssysteme, die den Umgang mit Echtzeiteigenschaften unterstützen
RTR	Remote Transmit Request, eine spezielle Kennung innerhalb eines CAN-Frames, die für Remote-Zugriffe genutzt wird
SAT	SATisfiability-Checking, ein Entscheidungsverfahren für boolesche Aussagenlogik
SoC	System On Chip, integrieren ganze Systeme auf einem einzigen Die
SOF	Start Of Frame, eine spezielle Kennung in einem CAN-Frame, welches den Start des Frames anzeigt
SRU	Smallest Replaceable Unit, ermöglicht virtuelle Slots innerhalb des TTP-Protokolls
TDMA	Time Devision Multiplex Access, ein Zeitscheibenverfahren für den Zugriff von Prozessorknoten auf ein Bussystem

THT	Token Hold Time, Zeitspanne, die jeder Knoten innerhalb eines Tokenring-Protokolls zum Senden auf dem Bus bekommt
TRT	Token Rotation Time, Zeit die das Token eines Tokenring-Busses maximal benötigt, einen Zyklus zu durchlaufen
TTCAN	Time-Triggered CAN, eine Protokollvariante des CAN-Busses, die um zeitbasierte Anteile erweitert wurde
TTP	Time-Triggered Protocol, ein serielles Busprotokoll, welches speziell hinsichtlich minimaler Ausfallrate entwickelt wurde
WCET	Worst Case Execution Time, Obergrenze der reinen Laufzeit eines Programms einer Task auf einem Prozessor ohne Unterbrechungen oder Betriebssystemkosten

Anhang B

Symbolverzeichnis

B_i	Blockierungszeit einer Task τ_i
B_{g_i}	Blockierungszeit einer Nachricht g_i
$\breve{C}_k$	Menge der Nachrichten, die auf ein Kommunikationsmedium k platziert sind
$\breve{C}_k^p$	Menge der Nachrichten, die einem Prozessor p bzgl. eines Kommunikationsmediums k zugeordnet sind
c_i	WCET einer Task τ_i
c_{ISR}	WCET einer ISR
c_{Sched}	WCET der Scheduling-Routine eines RTOS
$c_{g_i}^k$	Anzahl der Pakete einer Nachricht g_i auf einem Kommunikationsmedium k
$\tilde{c}_{g_i}^k$	Übertragungsdauer eines letzten, kürzeren Paketes einer Nachricht g_i auf einem Kommunikationsmedium k
d_i	End-to-End Deadline einer Taskkette, die mit Task τ_i beginnt
F	Anzahl der Cachefragmente in einer dynamisch rekonfigurierbaren Cache-Architektur
g_j	NAchricht zu Task τ_j aus der Menge der Nachrichten einer Task
h_k	Anzahl der Headerbits eines Frames im Kommunikationsmedium k
J_i	Jitter einer Task τ_i
J_{g_i}	Jitter einer Nachricht g_i
K	Menge der Kommunikationsmedien
MP	Menge der maximalen Pfade in einer hieraechischen Topologie
$MP\downarrow_{g_i}$	durch Nachricht g_i eingeschränkte Menge der maximalen Pfade
m_j	Kontext j eines multithreaded Prozessors
N_{Slot}^k	Obergrenze für Slotlängen in TDMA-Verfahren, ausgedrückt in Anzahl der maximalen Frames auf dem Bussystem k
O_j^i	Offset einer Task τ_j bzgl. des Startzeitpunktes einer Task τ_i
P	Menge der Prozessoren (ECUs)
PH	Menge der Pfadhüllen
$PH\downarrow_{g_i}$	Durch Nachricht g_i eingeschränkte Pfadhüllenmenge
pos	Position eines Kommunikationsmediums in einem Wort über der Sprache der Menge der Kommunikationsmedien
pr_i	Projektion, die den i-ten Slot aus einer TDMA-Runde liefert

pre	Menge aller Präfixe eines Wortes über die Menge der Kommunikationsmedien
r_i	Antwortzeit einer Task τ_i
r_{g_j}	Antwortzeit einer Nachricht g_j
S	Menge an Slots auf einem zeitbasierten Kommunikationsmedium
S^k	maximal erlaubte Anzahl an Slots in einem TDMA-basiertem Bussystem
t_i	Periode oder minimale Zwischenankunftzeit einer Task τ_i
tr_i	Aktivierungsart einer Task τ_i
U_p	Auslastung auf einem Prozessorknoten p
$\bar{U}$	mittlere Auslastung des Systems
X^p_{TDMA}	Menge aller Indizees der Slots innerhalb einer TDMA-Round, die einem Prozessor p zugeordnet sind
y	Varianz der WCET
$\mathcal{A}$	Definition einer Architektur
$\mathcal{C}^k_j$	Menge der Tasks die die k-te Taskkette bilden deren Starttask τ_j ist
$\mathcal{G}$	Menge der Nachrichten eines Taskssystems
$\mathcal{I}$	Zusätzliche Warteplätze für Interrupts in einem HW-Scheduler
$\mathcal{P}_i$	Leistungsaufnahme einer Task τ_i
$\mathcal{S}_{\tau_i}$	Anteil der Ausführungszeit einer Task τ_i an einer Taskkette
$\mathcal{S}_{g_i}$	Anteil der Ausführungszeit einer Nachricht g_i an einer Taskkette
$\mathcal{T}$	Tasksystem mit einer Menge von Tasks
$\mathcal{Z}$	Abbildung zur Berechnung der Bitzahl einer Variable im Zweierkomplement
α_τ	Gewichtungsfaktoren für den Anteil der Tasks an der End-to-End-Deadline
α_κ	Gewichtungsfaktoren für den Anteil der Nachrichten an der End-to-End-Deadline
β_k	Bitstuffing-Distanz der physikalischen Übertragung auf einem Kommunikationsmedium k
Γ	Platzierungsabbildung für Nachrichten
γ_i	Menge von Kommunikationen einer Task τ_i
γ_{ij}	Interferenzkosten hervorgerufen durch Preemptionen
Δ_τ	synthetische Deadline einer Task
Δ_γ	synthetische Deadline einer Nachricht
δ_i	Redundanztask einer Task τ_i
δ_{SW}	Differenz der Ausführungszeiten des Scheduling zwischen normaler und multithreaded Prozessorarchitektur
ε	Genauigkeit des binären Suchverfahrens innerhalb der Optimierung
ζ_{TDMA}	Ordnet jedem Slot einer TDMA-Round einen Prozessorknoten zu
η_k	Anzahl der Nutzdatenbytes eines Frames im Kommunikationsmedium k

θ_i	Menge der Tasks, die auf Kontext m_i eines multithreaded Prozessors platziert sind
κ	spezifische Leistungsdaten eines Bussystems
Λ	Länge einer TDMA-Round
λ	Größe des Zeitintervalls für eine End-to-End-Deadline
$\mu^{\mathcal{A}}$	Größe des Hauptspeichers eines Prozessorknotens
$\mu_i^{\mathcal{T}}$	Größe des Speicherbedarfs einer Task τ_i
Ξ	Maximaler Fehler einer kontinuierlichen Form der Antwortzeitberechnung
ξ	Übertragungszeit des Tokens in einem Tokenring-Bus
Π	Platzierungsabbildung, die angibt, auf welchem Prozessorknoten der angegebene Task platziert wird
π_i	Erlaubte Platzierungen einer Task τ_i
ρ_k	Übertragungszeit für ein Frame auf dem Kommunikationsmedium k
σ	Synchronisationsoverhead in Taskketten
τ	ein Task eines Tasksystems
υ_k	die Übertragungsrate des Mediums k in Bit/Zeiteinheit
ϕ	Abbildung zur Bestimmung der Prioritätenrelation zwischen zwei Tasks oder zwei Nachrichten
χ_k	Übertragungsoverhead auf Kommunikationsmedium k durch physikalische Übertragungsprotokolle
ψ_i^k	Gesamtantwortzeit einer Taskkette inkl. Tasks, Nachrichten und Synchronisation
ψ_{TDMA}	Länge eines Slots in einer TDMA-Round
Ω_i	Blockierungen durch das RTOS für Task τ_i

Anhang C

Deadlines gleicher Größe im DM-Verfahren

Beweis von Lemma 5.3.1 auf Seite 135.

LEMMA 5.3.1 (Optimalität bei Deadlinegleichheit). *Gegeben sei ein Tasksystem* $\mathcal{T} = \{\tau_1, \ldots, \tau_i, \ldots, \tau_j, \ldots, \tau_n\}$ *mit* $\forall \tau_k : d_i \leq t_i$. *Sei* $d_i = d_j$ *und für alle anderen Taskpaare* τ_k, τ_l *aus* $\mathcal{T}$ *gelte* $d_k \neq d_l$. *Es gelte gemäß der Deadline Monotonic Strategie* $\forall \tau_k, \tau_l \in \mathcal{T} : \phi_k < \phi_l$ *gdw.* $d_k > d_l$. *Für* τ_i *und* τ_j *gelte* $\phi_i < \phi_j$. *Sei ferner das Tasksystem* $\mathcal{T}$ *feasible. Dann ist es auch feasible, wenn* $\phi_i > \phi_j$ *gilt.*

Beweis:

Da alle außer den beiden betrachteten Prioritäten der Tasks von der Vertauschung unbeeinflusst bleiben, ergeben sich aufgrund eben dieser Vertauschung keine Einflüsse auf alle übrigen Tasks, seien sie nun höher oder niedriger priorisiert. Es genügt also, die beiden betroffenen Tasks τ_i und τ_j zu betrachten.

Für das Ursprungssystem mit $\phi_i < \phi_j$ gilt folglich:

$$r_i^* = c_i + \sum_{\tau_k \in \mathcal{T} \setminus \{\tau_j\} : \phi_k > \phi_i} \left\lceil \frac{r_i^*}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{r_i^*}{t_j} \right\rceil \cdot c_j \leq d_i$$

Wegen der Feasibility des Systems und mit $d_i \leq p_i \wedge d_i = d_j$ folgt hieraus, dass Task τ_j Task τ_i nur maximal genau einmal unterbrechen kann:

$$r_i^* = c_i + \sum_{\tau_k \in \mathcal{T} \setminus \{\tau_j\} : \phi_k > \phi_i} \left\lceil \frac{r_i^*}{t_k} \right\rceil \cdot c_k + c_j$$

Für Task τ_j gilt folglich:

$$r_j^* = c_j + \sum_{\tau_k : \phi_k > \phi_i} \left\lceil \frac{r_j^*}{t_k} \right\rceil \cdot c_k \leq d_j$$

Gilt hingegen $\phi_i > \phi_j$, so ergeben sich neue Antortzeiten für die beiden Tasks, die anschließend auf Feasibility hin untersucht werden sollen. Für Task τ_i ergibt sich

damit:

$$r_i = c_i + \sum_{\tau_k:\phi_k>\phi_i} \left\lceil \frac{r_j^*}{t_k} \right\rceil \cdot c_k$$

Dies ist offensichtlich kleiner als die Antwortzeit r_i^* im ursprünglichen Tasksystem. Somit erfüllt τ_i auch innerhalb der neuen Priorisierung ihren Anteil an der Feasibility des Gesamtsystems.

Für τ_j ergibt sich nach der Vertauschung der Prioritäten die neue Antwortzeit von:

$$r_j = c_j + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_i\}:\phi_k>\phi_i} \left\lceil \frac{r_j}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{r_j}{t_i} \right\rceil$$

Hier ist zunächst nicht offensichtlich, dass $r_j \leq d_j$ gilt. Wir werden dies nun mittels eines Widerspruchsbeweises zeigen:
Annahme: τ_j verletzt seine Deadline, es gilt also $r_j > d_j$.
Da für den Fixpunkt von r_i^* galt $r_i^* \leq d_i$ und weiterhin $d_i = d_j$, muss der Fixpunkt der rekursiven Gleichung von r_j in jedem Fall größer sein, als der von r_i^*. Sei W der Fixpunkt von r_i^*. Dann gilt folglich:

$$c_i + c_j + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_j,\tau_i\}:\phi_k>\phi_i} \left\lceil \frac{W}{t_k} \right\rceil \cdot c_k < c_j + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_j,\tau_i\}:\phi_k>\phi_i} \left\lceil \frac{W}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{W}{t_i} \right\rceil \cdot c_i$$

Dies kann vereinfacht werden zu:

$$c_i < \left\lceil \frac{W}{t_i} \right\rceil \cdot c_i$$

Da $W \leq d_i \leq p_i$ gilt, folgt daraus $\left\lceil \frac{W}{t_i} \right\rceil = 1$ und damit:

$$c_i < c_i$$

Dies ist offensichtlich ein Widerspruch und damit wird die oben gemachte Annahme, τ_j könne unter der neuen Priorisierung seine Deadline verletzen, ungültig.
Es gilt folglich $r_j \leq d_j$. □

Bemerkung: Das oben gezeigte Prinzip gilt nicht mehr, wenn Jitter mitbetrachtet wird. Dies werden wir an der gleichen Situation deutlich machen. Es gelten also im folgenden die Vorraussetzungen von Lemma 5.3.1, jedoch verfügt jede Task τ_k zusätzlich über einen Jitter J_k. Die einzige Aussage, die wir über den jeweiligen Jitter hier machen können, ist, dass gilt $J_k < p_k$. Dies folgt aus der Konstruktion der Jitter-Werte, wie sie in Kapitel 4.4 dargelegt wurde.
Es gilt folglich für die Antwortzeiten der beiden betrachteten Tasks vor der Vertauschung der Prioritäten:

$$r_i^* = c_i + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_j\}:\phi_k>\phi_i} \left\lceil \frac{r_i^* + J_k}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{r_i^* + J_j}{t_j} \right\rceil \cdot c_j \leq d_i$$

Zwar ist dieses System gemäß Vorraussetzung feasible, jedoch kann hier aufgrund des Jitters von Task τ_j keine weitere Vereinfachung der Gleichung gefunden werden. Für τ_j gilt gleichermaßen analog zu obiger Gleichung:

$$r_j^* = c_j + \sum_{\tau_k:\phi_k>\phi_i} \left\lceil \frac{r_j^* + J_k}{t_k} \right\rceil \cdot c_k \leq d_j$$

Werden nun die Prioritäten vertauscht, können wir mit der gleichen Argumentation versuchen, einen Widerspruchsbeweis zu führen. Es gelte also die Annahme, dass bei Vertauschung der Prioritäten τ_j seine Deadline verletzt. Wiederum wird als Ausgangspunkt der Fixpunkt W aus dem Ursprungssystem für r_i^* betrachtet. Dann ergibt sich analog dem oben geführten Beweis:

$$\begin{aligned} & c_i + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_j,\tau_i\}:\phi_k>\phi_i} \left\lceil \frac{W + J_k}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{W + J_j}{t_j} \right\rceil \cdot c_j \\ < \quad & c_j + \sum_{\tau_k\in\mathcal{T}\setminus\{\tau_j,\tau_i\}:\phi_k>\phi_i} \left\lceil \frac{W + J_k}{t_k} \right\rceil \cdot c_k + \left\lceil \frac{W + J_i}{t_i} \right\rceil \cdot c_i \end{aligned}$$

Dies kann vereinfacht und umgeformt werden zu:

$$\left(c_i - \left\lceil \frac{W + J_i}{t_i} \right\rceil \cdot c_i \right) + \left(\left\lceil \frac{W + J_j}{t_j} \right\rceil \cdot c_j - c_j \right) < 0 \tag{C.1}$$

Hier kann offensichtlich kein Widerspruch konstruiert werden. Zur Verdeutlichung können wir die möglichen Werte der beiden Ceiling-Funktionen weiter einschränken. Da $J_k < d_k$ gilt und zudem $W \leq d_i \leq t_i$ sowie $W \leq d_i = d_j \leq p_j$ nach Vorraussetzung gilt, können folgende obere Schranken für die betrachtete Iteration der rekursiven Gleichungen festgelegt werden:

$$\left\lceil \frac{W + J_i}{t_i} \right\rceil \leq 2 \quad \wedge \quad \left\lceil \frac{W + J_j}{t_j} \right\rceil \leq 2$$

Damit kann dann aber Gleichung C.1 erfüllt werden, wenn beispielsweise $\left\lceil \frac{W+J_i}{t_i} \right\rceil = 2$ und $\left\lceil \frac{W+J_j}{t_j} \right\rceil = 1$ gewählt werden. Dann folgt aus Gleichung C.1:

$$\begin{aligned} (c_i - 2c_i) + (c_j - c_j) \quad & < \quad 0 \\ -c_i \quad & < \quad 0 \end{aligned}$$

Offensichtlich können also für Jitter-behaftete Systeme Beispiele konstruiert werden, in denen Lemma 5.3.1 nicht mehr gilt.

Generell kann dieses Ergebnis auf die Optimalität des Deadline Montonic Verfahren übertragen werden. Schon Tindell hat in [186] gezeigt, dass dieses Verfahren unter Berücksichtigung von Jitter nicht mehr notwendigerweise optimal ist. Es existiert allerdings zur Zeit kein Prioritätenverfahren, welches für Tasksysteme mit Jitter optimal ist. Hier können also nur Heuristiken angewendet werden und die Deadline Monotonic Heuristik erfreut sich weiterhin einer großen Beliebtheit, da sie auch für

Tasksysteme mit Jitter gute Prioritätenordnungen liefert. Wir werden daher innerhalb dieser Arbeit diese Heuristik beibehalten. Zwar schlägt Tindell in [186] eine weitere Heuristik vor, die den Jitter mit einbezieht, aber dieses Verfahren ist hier aufgrund der unterschiedlichen Vorraussetzungen[1] im Vergleich zu [186] nichteinsetzbar.

[1]Tindell berechnet in [186] die Antwortzeit einer Task jeweils vom Startzeitpunkt der ersten Task der Taskkette an, d.h. er führt Antwortzeiten von Vorgängertasks und -Nachrichten mit. Daraus folgt letztlich, dass der Jitter immer grundsätzlich kleiner als die Antwortzeit sein muss und dies begründet auch die Prioritätenvergabe der Tasks. Dies ist im hier betrachteten Fall anders: Da die Tasks grundsätzlich über eigene, lokale Deadlines verfügen und also die Antwortzeiten nur lokal betrachtet werden, sind durchaus Jitter-Werte über der Antwortzeit der Task möglich; sie haben mithin nichts miteinander zu tun und sind vollkommen getrennt zu betrachten.

Anhang D

TATONO — Software-Struktur

D.1 Struktur der Software

Die automatische Platzierung, wie sie in Kapitel 5 beschrieben wurde, basiert bzgl. des Optimierungsverfahrens wesentlich auf dem HySAT-Kernel, der jedoch für eine iterative Suche inklusive Paralleler Ausführungen erweitert wurde und in dem Platzierungswerkzeug TATONO aufgegangen ist. Hier werden gleichzeitig die jeweiligen Gleichungssysteme gemäß der Vorschriften aus Kapitel 5.3 erzeugt, die jeweils mit einer ganzen Reihe an Optimierungen versehen sind. Ebenfalls angeschlossen ist die Synthese der Deadlines aus End-to-End-Deadlines, wie sie in Kapitel 6 vorgestellt wurde. Einen Gesamtüberblick über die Softwarearchitektur zeigt die Abbildung D.1.

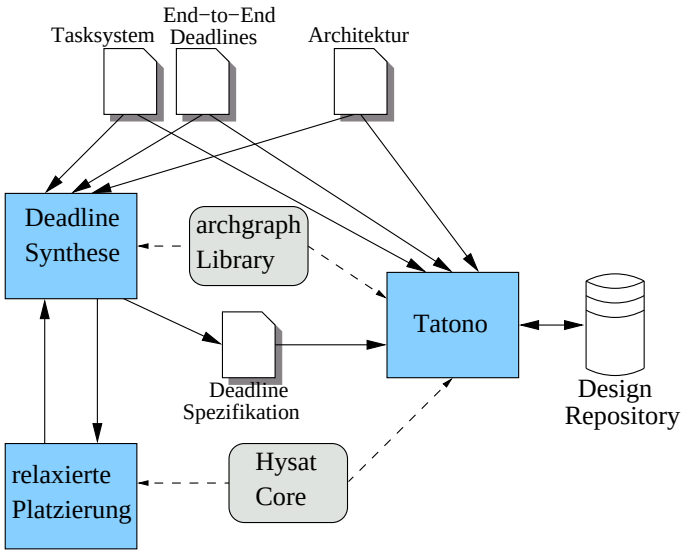


Abbildung D.1: Übersicht über die Software-Struktur des Platzierungswerkzeuges Tatono und seiner Umgebung.

Eingabe für sowohl die Deadline-Synthese als auch die automatische Platzierung sind Spezifikationen des Tasksystems, der Architektur, sowie der End-to-End-Deadlines. Aus diesen Daten wird zunächst ein Modell des Systems innerhalb der Deadline-

Synthese geschaffen. Anschließend werden die einzelnen Pfade der End-to-End-Deadlines im Tasksystem innerhalb der Deadline-Synthese extrahiert. Je nach Synthese-Verfahren (vgl. Kapitel 6.2.3) wird entweder direkt aus diesen Daten die Deadline für jede Nachricht und jede Task berechnet, oder aber es findet eine relaxierte Platzierung statt. Die relaxierte Platzierung wird hier wiederum mittels eines iterativen, auf binärer Suche basierenden, Aufrufens des HySST-Kerns durchgeführt. Dazu ist eine entsprechende, speziell für diesen Zweck erweiterte, Solverversion geschaffen worden. Als Ergebnis wird eine Platzierung zurückgegeben, auf deren Basis die Deadline-Synthese die Antwortzeiten von Tasks berechnet und diese für eine anteilsbasierte Intervallzuteilung nutzen kann (vgl. Kapitel 6.2.3). Unter Ausnutzung der Architektureigenschaften über die archgraph-Library können dann die erzeugten lokalen Deadlines spezifiziert und an das Platzierungswerkzeug TATONO weitergereicht werden.
TATONO nutzt ebenfalls den HySAT-Kern zur Optimierung, erstellt aber auch den Satz von Gleichungen. Hierbei ist insbesondere die Wahl des Pfades innerhalb der Architektur für Nachrichten zu berücksichtigen, wie sie in Kapitel 5.3.5 vorgestellt wurde. Die zentrale archgraph-Library bietet hierfür entsprechende Funktionen an, die mittels Graphenoperationen auf der Topologie des Systems durchgeführt werden. Zahlreiche Optimierungen, wie preemptionsfrei Fenster, das Erkennen von garantiert verschiedenen Prioritäten, Das Erkennen überflüssiger Gleichungsanteile aufgrund garantiert unterschiedlicher Platzierungen von Tasks, etc. sorgen dafür, dass die erzeugten Gleichungssysteme möglichst kompakt generiert werden. Optimierungskriterien werden über Kommandozeilenparameter (ebenso wie weitergehende Optionen, wie etwa variable WCETs, Multithreading-Einsatz oder auch Anzahl der parallelen Prozesse, etc.) aus einem Satz vorgegebener Funktionen ausgewählt. Das Ergebnis wird anschließend in einem Repository abgespeichert und kann zur weiteren Verwendung, insbesondere auch zur inkrementellen Integration, hieraus abgegriffen werden.

D.2 Designflow — Ein Beispiel

Der oben dargestellte generelle Ablauf bei einer automatischen Platzierungsberechnung soll nun anhand eines sehr kleinen Beispiels exemplarisch vorgeführt werden. Dabei werden jeweils die notwendigen Schnittstellen in Form ihrer tatsächlichen Datenformate abgedruckt. Im weiteren werden wir auf eine detaillierte Darstellung der Syntax dieser Zwischenformate allerdings verzichten, da sie zumeist selbsterklärend sind und anhand der hier abgedruckten Beispiele studiert werden können.

Das betrachtete Beispielsystem habe eine Struktur wie in Abbildung D.2 gegeben, d.h. es besteht aus 5 Tasks, die in zwei Taskgraphen angeordnet sind und einer Architektur aus 2 Prozessorknoten sowie einem zentralen Tokenring-Bussystem.

Die Charakterisierung der einzelnen Tasks sind im folgenden Listing erkennbar, dass das zentrale Eingabeformat für Tasksysteme darstellt. Um die Platzierungsbeschränkungen zu motivieren, sind in Abbildung D.2 zusätzlich noch Sensoren und Aktuatoren eingezeichnet.
Das Eingabeformat für Tasksysteme legt sämtliche Parameter einer Task, wie etwa

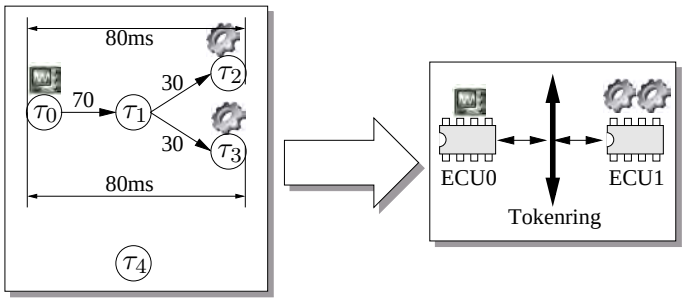


Abbildung D.2: Tasksystem und Architektur des Beispielsystems.

WCET, BCET, Speicherverbrauch, Periode, etc. fest, wobei hier auf eine Darstellung unterschiedlicher WCETs und BCETs aufgrund unterschiedlicher Prozessoren verzichtet wurde. Die Triggerbedingung zeigt an, ob eine lose oder eine feste Kopplung einer Task an eingehende Nachrichten erfolgt. Nachrichten werden hier ebenfalls spezifiziert in Form von Zieltask und Anzahl der Nutzdatenbytes. Die Flags `mediumsused` und `mediumsnotused` sind zunächst nicht von Bedeutung; Sie werden für die inkrementelle Integration genutzt, um für schon platzierte Nachrichten die benutzten Bussysteme mitzuteilen. Platzierungsrestriktionen werden invertiert zum π_i-Parameter einer Task dargestellt, drücken also aus, auf welchen Knoten diese Task nicht platziert werden darf.
Mit diesen Informationen kann nun das Eingabeformat für Tasksysteme für obiges Beispiel niedergeschrieben werden.

```
taskspec: T0
wcet: 8
bcet: 8
period: 80
memory: 800
trigger: period
communications: 1
target: T1 70
mediumsused: 0
mediumsnotused: 0
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec

taskspec: T1
wcet: 10
bcet: 10
period: 80
memory: 1000
trigger: comm
communications: 2
target: T2 30
mediumsused: 0
mediumsnotused: 0
```

```
target: T3 30
mediumsused: 0
mediumsnotused: 0
forbidden: 0
duplicate: _no_
endtaskspec

taskspec: T2
wcet: 5
bcet: 5
period: 80
memory: 1200
trigger: comm
communications: 0
forbidden: 1
node: ECU0
duplicate: _no_
endtaskspec

taskspec: T3
wcet: 10
bcet: 10
period: 80
memory: 700
trigger: comm
communications: 0
forbidden: 1
node: ECU0
duplicate: _no_
endtaskspec

taskspec: T4
wcet: 6
bcet: 6
period: 50
memory: 2500
trigger: period
communications: 0
forbidden: 0
duplicate: _no_
endtaskspec
```

Diese Spezifikation der Eigenschaften eines Tasksystems enthält allerdings noch keine Angaben über etwaige Deadlines. Stattdessen erfolgt die Spezifikation der End-to-End-Deadlines in einem weiteren Zwischenformat. Hier wird für jeden Pfad in den verschiedenen Taskgraphen End-to-End-Deadlines festgelegt. Tasks, die isoliert stehen und trotzdem eine Deadline einhalten müssen, werden als Taskketten der Länge 1 definiert. Für obiges Beispiel existieren folglich 3 End-to-End-Deadlines, zwei innerhalb der Taskkette von 80ms Länge und eine für den Einzeltasks von

50ms Länge.

```
e2e_spec
deadline: 80
tasks: 3
T0
T1
T2
end_e2espec

e2e_spec
deadline: 80
tasks: 3
T0
T1
T3
end_e2espec

e2e_spec
deadline: 50
tasks: 1
T4
end_e2espec
```

Anschließend muss noch die Architektur charakterisiert werden. Hierzu besteht ebenfalls ein textuelles Eingabeformat, in dem die Prozessorknoten, die Bussysteme und die verwendete Topologie festgelegt werden können. Die einzige zur Zeit verwertbare Information über Prozessoren besteht hierbei aus der Größe des Hauptspeichers. Für Bussysteme werden neben dem Übertragungsparadigma (in diesem Beispiel Tokenring, hier können aber auch CAN, TTP oder FlexRay angegeben werden) noch die notwendigen Informationen zu den Headerbits eines Frame mitgegeben. Die Topologie des Systems wird über die sogenannten "connections" definiert, in der für jeden Prozessorknoten die direkte Verbindung zu anderen Prozessorknoten mittels einer Liste der an diesen Knoten angeschlossenen Bussysteme angegeben ist. Transitive Verbindungen, etwa über hierarchische Strukturen, werden dabei automatisch vom System errechnet und berücksichtigt.

```
-- Prozessor-Spezifikationen
--
node_declaration:
ECU0 ( processor = MSPARC;
       memory = 10000 bytes; )
ECU1 ( processor = MSPARC;
       memory = 10000 bytes; )

-- Bus-Spezifikationen
--
comm_declaration:
Tok_0 ( network = Tokenring;
        rate = 100 MBsec;
```

```
        header = 0;
        databytes = 0; )

-- Verbindungs-Spezifikation
--
connection:
ECU0 (
ECU1: {Tok_0};
)
ECU1 (
ECU0: {Tok_0};
)
```

Diese drei Eingabeformate definieren das zu verteilende System vollständig. Bevor die eigentliche Platzierung erfolgen kann, müssen nun zunächst die Deadlines für Tasks und Nachrichten ermittelt werden. Dies kann, wie bereits beschrieben, auf unterschiedlichste Weite geschehen. Hier wollen wir die relaxierte Verteilung betrachten, die zur Generierung von Deadlines genutzt wird. Diese benötigt einen Optimierungslauf auf einem abgeschwächten System, aber auch schon unter Verwendung eines SAT-Checkers. Die für das oben dargestellte System erzeugten Gleichungen können im folgenden abgelesen werden (Man beachte auch das Eingabeformat des SAT-Checkers, welches eine sehr kompakte Form der Gleichungserzeugung liefert, die gut analysierbar ist). Wesentlicher Punkt ist hierbei die Abschwächung der Platzierungsinformation, die in diesem Beispiel anhand einer Utilisationsbetrachtung (im Gegensatz zur Betrachtung von Antwortzeiten) erfolgt. Abbruchkriterium für Platzierungen, die (potentiell) nicht feasible sind, erfolgt durch eine Auslastunsggrenze von 90%. Speicherverbrauch und die einfache Form der Kommunikationslatenzen mittels Berechnung der TRT wird hier beibehalten. Die Domänen der verschiedenen Variablen sind jeweils nach den Vorschriften aus Kapitel 5.6 errechnet worden. Zu beachten sind auch Skalierungen der Wertebereiche, um mit Zahlen umgehen zu können, die in der normalen Zeitauflösung nicht-ganzzahlig sind.

```
DECL

-- TRT
longint [0,13] trt;

-- Knoten Util
longint [0,52] u_0;
longint [0,52] u_1;

-- Taskzuweisung auf Knoten
longint [0,1] t0;
longint [0,1] t1;
longint [0,1] t2;
longint [0,1] t3;
longint [0,1] t4;

-- Speicher
longint [0,6200] m_0;
```

```
longint [0,6200] m_1;

EXPR

-- TRT berechnen
trt = 0
        + if (t0!=t1) then 7 else 0 fi
        + if (t1!=t2) then 3 else 0 fi
        + if (t1!=t3) then 3 else 0 fi;

-- Speicherauslastung berechnen
m_0 = 0
       + if (t0=0) then 800 else 0 fi
       + if (t1=0) then 1000 else 0 fi
       + if (t2=0) then 1200 else 0 fi
       + if (t3=0) then 700 else 0 fi
       + if (t4=0) then 2500 else 0 fi;

m_1 = 0
       + if (t0=1) then 800 else 0 fi
       + if (t1=1) then 1000 else 0 fi
       + if (t2=1) then 1200 else 0 fi
       + if (t3=1) then 700 else 0 fi
       + if (t4=1) then 2500 else 0 fi;

-- Speichergrenze
m_0 <= 10000;
m_1 <= 10000;

-- Verbotene Knoten
t0 != 1;
t2 != 0;
t3 != 0;

-- Util-Begrenzung
u_0 <= 90;
u_1 <= 90;

-- Util berechnen
u_0 = 0
        + if (t0 = 0) then 10 else 0 fi
        + if (t1 = 0) then 12 else 0 fi
        + if (t2 = 0) then 6 else 0 fi
        + if (t3 = 0) then 12 else 0 fi
        + if (t4 = 0) then 11 else 0 fi;
u_1 = 0
        + if (t0 = 1) then 10 else 0 fi
        + if (t1 = 1) then 12 else 0 fi
        + if (t2 = 1) then 6 else 0 fi
```

```
        + if (t3 = 1) then 12 else 0 fi
        + if (t4 = 1) then 11 else 0 fi;

-- TRT-Begrenzung
trt <= 405;
```

Die Deadline-Synthese erzeugt nun fixierte Echtzeitschnittstellen, die jeweils den Aktivierungsjitter und die Deadline einer Task festlegen. Dies wird wiederum in einem textuellen Zwischenformat abgelegt und dient als Eingabe in die automatische Platzierung als zusätzliche Information und anstelle der End-to-End-Deadlines, für das hier betrachtete Beispielszenario als die folgenden Werte (wobei bei Wertpaaren immer gilt: der erste Wert entspricht der Deadline, der zweite entspricht dem Jitter):

```
t0: 21 0
nr_of_communication: 1
t1 10 13
t1: 22 16
nr_of_communication: 2
t2 8 28
t3 6 28
t2: 16 33
nr_of_communication: 0
t3: 18 31
nr_of_communication: 0
t4: 50 0
nr_of_communication: 0
```

Mittels dieser Informationen (Tasksystem, Deadlines für Tasks und Nachrichten und Architektuspezifikation) kann nun die automatische Platzierung mittels TATONO erfolgen, dass die jeweils notwendige Gleichungsstruktur erzuegt und dem HySAT-Kern zur Lösung übergibt. Für das oben angegebene Beispiel ergibt sich der folgende Satz an Gleichungen für HySAT. Welchem Zweck die unterschiedlichen Variablen dienen, ist jeweils bei ihrer Deklaration kommentiert. Die `Pf`-Variablen dienen hierbei übrigens der Pfadhüllendefinition für das Routing von Nachrichten auf der Topologie. Sie sind in der Erstellungsphase gemäß den Vorschriften aus Kapitel 5.3.5 eindeutig charakterisiert worden. `serv`-Variablen dienen der Berücksichtigung der Kosten von Gateway-Servern auf dem Weg einer Nachricht über mehrere Bussysteme. Hier sind diese Variablen mit 0 belegt, da es erstens nur ein Bussysteme gibt und damit keine Server benötigt werden, und zweitens innerhalb der Architekturspezifikation die Kosten hierfür unberücksichtigt blieben. Im Prinzip finden sich hier also alle Gleichungen aus Kapitel 5.3 wieder, wenn auch in leicht veränderter und an das Eingabeformat des Solvers angepassten Versionen.

```
-----------------------------------------------
--
-- HYSAT input file
--
-- Author: tatono--
-----------------------------------------------
DECL
```

```
-- num of nodes: 2
-- num of tasks: 5
boole TRUE;
boole FALSE;

-- mi : Speicherverbrauch aller Tasks die auf Knoten i platziert sind
longint[0,6200] m0;
longint[0,6200] m1;

-- di : Deadline von Task i
longint[0,800] d0;
longint[0,800] d1;
longint[0,800] d2;
longint[0,800] d3;
longint[0,500] d4;

-- ti : Platzierungsvariable
longint[0,1] t0;
longint[0,1] t1;
longint[0,1] t2;
longint[0,1] t3;
longint[0,1] t4;

-- wi : Response Time von Task i
longint[0,800] w0;
longint[0,800] w1;
longint[0,800] w2;
longint[0,800] w3;
longint[0,500] w4;

-- K_k_i_j gibt an, ob das Medium k von Kommunikation von Task i zu Task j
--         benutzt wird
boole KTok_0_0_1;
boole KTok_0_1_2;
boole KTok_0_1_3;

-- lat_k_i_j gibt die Latenzzeit der Kommunikation von Task i zu Task j ueber
--           das Medium k an
longint[0, 100000] latTok_0_0_1;
longint[0, 80000] latTok_0_1_2;
longint[0, 60000] latTok_0_1_3;

-- locD_k_i_j fuer sub-lokale Jitter und Deadlinewerte der Kommunikation von
--            Task i zu Task j ueber Medium k
longint[0, 100000] locDTok_0_0_1;
longint[0, 80000] locDTok_0_1_2;
longint[0, 60000] locDTok_0_1_3;

-- wmk_i_j gibt die response-time der Kommunikation von Task i zu Task j ueber
```

```
  --         das Medium k an
  longint [0, 100000] wmTok_0_0_1;
  longint [0, 80000] wmTok_0_1_2;
  longint [0, 60000] wmTok_0_1_3;

  -- Jk_i_j gibt den jitter der Kommunikation von Task i zu Task j ueber das
  --        Medium k an
  longint [0, 230000] JTok_0_0_1;
  longint [0, 360000] JTok_0_1_2;
  longint [0, 340000] JTok_0_1_3;

  -- Pf_i_j Pfadvariable der Kommunikation von Task i zu Task j
  longint [0, 1] Pf0_1;
  longint [0, 1] Pf1_2;
  longint [0, 1] Pf1_3;

  -- servi_j der Kommunikation von Task i zu Task j
  longint [0, 1] serv0_1;
  longint [0, 1] serv1_2;
  longint [0, 1] serv1_3;


  -- hi_j : Anzahl der Preemptionen von Task i durch Task j
  longint[0,2] h0_4;
  longint[0,2] h1_4;
  longint[0,1] h2_3;
  longint[0,2] h2_4;
  longint[0,1] h3_2;
  longint[0,2] h3_4;
  longint[0,1] h4_0;
  longint[0,1] h4_1;
  longint[0,1] h4_2;
  longint[0,1] h4_3;

  -- Token Rotation Time für jeden Tokenring
  longint[0,3000] trt_Tok_0;
  longint[0,3000] tokentrt;

EXPR

  TRUE;
  !FALSE;

  (t0 = t1) -> Pf0_1 = 0;
  (t1 = t2) -> Pf1_2 = 0;
  (t1 = t3) -> Pf1_3 = 0;

  KTok_0_0_1 -> (locDTok_0_0_1 >= 70000);
  KTok_0_1_2 -> (locDTok_0_1_2 >= 30000);
```

```
KTok_0_1_3 -> (locDTok_0_1_3 >= 30000);

(!KTok_0_0_1) -> (locDTok_0_0_1 = 0);
(!KTok_0_1_2) -> (locDTok_0_1_2 = 0);
(!KTok_0_1_3) -> (locDTok_0_1_3 = 0);

KTok_0_0_1 -> (latTok_0_0_1 = wmTok_0_0_1);
KTok_0_1_2 -> (latTok_0_1_2 = wmTok_0_1_2);
KTok_0_1_3 -> (latTok_0_1_3 = wmTok_0_1_3);

(!KTok_0_0_1) -> (latTok_0_0_1 = 0);
(!KTok_0_1_2) -> (latTok_0_1_2 = 0);
(!KTok_0_1_3) -> (latTok_0_1_3 = 0);

latTok_0_0_1 <= locDTok_0_0_1;
latTok_0_1_2 <= locDTok_0_1_2;
latTok_0_1_3 <= locDTok_0_1_3;

Pf0_1 != 0 -> (locDTok_0_0_1 + serv0_1 = 100000);
Pf1_2 != 0 -> (locDTok_0_1_2 + serv1_2 = 80000);
Pf1_3 != 0 -> (locDTok_0_1_3 + serv1_3 = 60000);

(KTok_0_0_1) -> (serv0_1 = 0
     + if (KTok_0_0_1) then 0 else 0 fi
     - 0);
(KTok_0_1_2) -> (serv1_2 = 0
     + if (KTok_0_1_2) then 0 else 0 fi
     - 0);
(KTok_0_1_3) -> (serv1_3 = 0
     + if (KTok_0_1_3) then 0 else 0 fi
     - 0);

(Pf0_1 = 0) -> ( (!KTok_0_0_1) and (t0 = t1));
(Pf0_1 = 1) -> (((KTok_0_0_1)
               and (t0 = 0 or t0 = 1)
               and (t1 = 0 or t1 = 1)))
               and (JTok_0_0_1 = 130);

(Pf1_2 = 0) -> ( (!KTok_0_1_2) and (t1 = t2));
(Pf1_2 = 1) -> (((KTok_0_1_2)
               and (t1 = 0 or t1 = 1)
               and (t2 = 0 or t2 = 1)))
               and (JTok_0_1_2 = 280);

(Pf1_3 = 0) -> ( (!KTok_0_1_3) and (t1 = t3));
(Pf1_3 = 1) -> (((KTok_0_1_3)
               and (t1 = 0 or t1 = 1)
               and (t3 = 0 or t3 = 1)))
               and (JTok_0_1_3 = 280);
```

```
w0 = 80
     + if((t0 = t4) and (d0 > d4)) then h0_4 * 60 else 0 fi;
w1 = 100
     + if((t1 = t4) and (d1 > d4)) then h1_4 * 60 else 0 fi;
w2 = 50
     + if((t2 = t3) and (d2 > d3)) then h2_3 * 100 else 0 fi
     + if((t2 = t4) and (d2 > d4)) then h2_4 * 60 else 0 fi;
w3 = 100
     + if((t3 = t2) and (d3 > d2)) then h3_2 * 50 else 0 fi
     + if((t3 = t4) and (d3 > d4)) then h3_4 * 60 else 0 fi;
w4 = 60
     + if((t4 = t0) and (d4 > d0)) then h4_0 * 80 else 0 fi
     + if((t4 = t1) and (d4 > d1)) then h4_1 * 100 else 0 fi
     + if((t4 = t2) and (d4 > d2)) then h4_2 * 50 else 0 fi
     + if((t4 = t3) and (d4 > d3)) then h4_3 * 100 else 0 fi;

(500 * h0_4 - 499) <= w0;
(500 * h0_4) >= (w0 + 0);
(500 * h1_4 - 499) <= w1;
(500 * h1_4) >= (w1 + 0);
(800 * h2_3 - 1109) <= w2;
(800 * h2_3) >= (w2 + 310);
(500 * h2_4 - 499) <= w2;
(500 * h2_4) >= (w2 + 0);
(800 * h3_2 - 1129) <= w3;
(800 * h3_2) >= (w3 + 330);
(500 * h3_4 - 499) <= w3;
(500 * h3_4) >= (w3 + 0);
(800 * h4_0 - 799) <= w4;
(800 * h4_0) >= (w4 + 0);
(800 * h4_1 - 959) <= w4;
(800 * h4_1) >= (w4 + 160);
(800 * h4_2 - 1129) <= w4;
(800 * h4_2) >= (w4 + 330);
(800 * h4_3 - 1109) <= w4;
(800 * h4_3) >= (w4 + 310);

w0 <= d0;
d0 = (210 + if ( (t0=t1) ) then 100 else 0 fi);
w1 <= d1;
d1 = (220 + if ( (t1=t2)  and  (t1=t3) ) then 80 else 0 fi);
w2 <= d2;
d2 = (160);
w3 <= d3;
d3 = (180);
w4 <= d4;
d4 = (500);
```

```
t0!=1;
t2!=0;
t3!=0;

m0 = 0
     +800
     +if (t1=0) then 1000 else 0 fi
     +if (t4=0) then 2500 else 0 fi;

m1 = 0
     +if (t1=1) then 1000 else 0 fi
     +1200
     +700
     +if (t4=1) then 2500 else 0 fi;

m0 <= 10000;
m1 <= 10000;

wmTok_0_0_1 = trt_Tok_0*100;
wmTok_0_1_2 = trt_Tok_0*100;
wmTok_0_1_3 = trt_Tok_0*100;

trt_Tok_0 = 0
     + if (!KTok_0_0_1) then 0 else 700 fi
     + if (!KTok_0_1_2) then 0 else 300 fi
     + if (!KTok_0_1_3) then 0 else 300 fi;

tokentrt = 0 + trt_Tok_0;
```

Letztlich wird das iterative Optimierungsverfahren eine Lösung der Platzierungsanfrage liefern und diese im Design Repository abspeichern. Letzteres ist exakt das Eingabeformat für Tasksysteme, in dem aber nur noch eine mögliche Platzierung für die jeweilige Task verbleibt. Pfade der Nachrichten werden durch die oben bereits angedeuteten Flags `mediumsused` und `mediumsnotused` festgelegt, wie im hier betrachteten Beispiel die Nachrichten von Task τ_1. Für das hier betrachtete Beispielsystem ergibt sich bei einer Optimierung nach minimaler Token Rotation Time des Tokenring-Bussystems das folgende Ergebnis:

```
taskspec: T0
wcet: 8
bcet: 8
period: 80
memory: 800
trigger: period
communications: 1
target: T1 70
mediumsused: 0
mediumsnotused: 1
Tok_0
```

```
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec

taskspec: T1
wcet: 10
bcet: 10
period: 80
memory: 1000
trigger: period
communications: 2
target: T2 30
mediumsused: 1
Tok_0
mediumsnotused: 0
target: T3 30
mediumsused: 1
Tok_0
mediumsnotused: 0
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec

taskspec: T2
wcet: 5
bcet: 5
period: 80
memory: 1200
trigger: period
communications: 0
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec

taskspec: T3
wcet: 10
bcet: 10
period: 80
memory: 700
trigger: period
communications: 0
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec
```

```
taskspec: T4
wcet: 6
bcet: 6
period: 50
memory: 2500
trigger: period
communications: 0
forbidden: 1
node: ECU1
duplicate: _no_
endtaskspec
```

Anhang E

Beispielsysteme

E.1 Beispielsystem für End-to-End Deadlines

Die Daten für das Beispielsystem zur Integration von End-to-End-Deadlines können Tabelle E.1 entnommen werden. Das System besteht aus 30 Tasks, die grob in 2 Gruppen mit ähnlichen Platzierungsrestriktionen unterteilt werden können und damit ein typisches Beispiel aus der betrachteten Anwendungsdomäne darstellen.

Tabelle E.1: Synthetischer Benchmark mit End-to-End-Deadlines.

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
0	50	3.5	854	$121 \rightarrow 11, 133 \rightarrow 7$	0,1,2,3
1	10	1.3	4084	$61 \rightarrow 2$	0,1,2,3,4
2	10	1.3	3799	$102 \rightarrow 16$	0,1,2,3
3	30	6	3330		
4	10	0.7	4735	$143 \rightarrow 8$	0,1,2,3
5	80	7.9	2497	$155 \rightarrow 6, 81 \rightarrow 10$	0,1,2,3,4
6	80	7.9	2098		0,1,2,3
7	50	3.5	4179	$165 \rightarrow 12, 151 \rightarrow 14$	0,1,2,3
8	10	0.7	646	$118 \rightarrow 13, 123 \rightarrow 9$	0,1,2,3
9	10	0.7	2011		0,1,2,3
10	80	7.9	2411		0,1,2,3,5,6
11	50	3.5	1636	$41 \rightarrow 15$	0,1,2,3,7
12	50	3.5	1349	$184 \rightarrow 17$	0,1,2,3
13	10	0.7	2179		
14	50	3.5	1104		0,1,2,3,4
15	50	3.5	3927		0,1,2,3,7
16	10	1.3	3417		0,1,2,3
17	50	3.5	1912		
18	35	5.8	4979	$117 \rightarrow 19, 195 \rightarrow 23$	4,5,6,7
19	35	5.8	2149		2,4,5,6,7
20	15	1.2	1852	$85 \rightarrow 21, 147 \rightarrow 26$	4,5,6,7
21	15	1.2	2964	$95 \rightarrow 25$	4,5,6,7

Tabelle E.1: Synthetischer Benchmark mit End-to-End-Deadlines.

22	35	5.8	1117		4,5,6,7
23	35	5.8	4047		4,5,6,7
24	80	7.9	4283	163 → 29	4,5,6,7
25	15	1.2	1209		2,3,4,5,6,7
26	15	1.2	3426	32 → 28	4,5,6,7
27	55	27.5	2252		4,5,6,7
28	15	1.2	2780		0,4,5,6,7
29	80	7.9	1456		1,4,5,6,7

End-to-End-Deadlines wurden jeweils über alle möglichen Pfade der Taskgraphen gelegt und mit einem Wert gleich der Periode der jeweiligen Taskkette belegt.
Als Zielarchitektur wurde die in [183] bereits eingeführte Architektur verwendet (siehe auch Seite 171 und folgende), allerdings wurde für Knoten 4 und 5 der Speicher auf 12000 Bytes erhöht. Näheres zur Architektur ist Tabelle 5.4 auf Seite 174 zu entnehmen.

E.2 Fallstudie zur inkrementelle Integration

Die Fallstudie zur inkrementellen Integration, die auf Seite 223 und folgende verwendet wurde, besteht aus 4 Tasksets, die nacheinander platziert wurden. Für jedes der Tasksets *Comp A1*, *Comp A2*, *Comp B*, sowie *Comp Ctrl* ist in den Tabellen E.2 bis E.5 die vollständige Definition zu finden.

Tabelle E.2: Komponente *Comp A1* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.5 auf Seite 225).

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
0	100	10	2000		0,1,2,3,4,5
1	2000	30	1000	10 → 2	0
2	2000	60	2000	10 → 3	0,1,2,3,4,5
3	2000	40	1000	20 → 4, 10 → 5	0,1,2,3,4,5
4	2000	20	1000		1
5	2000	80	3000		2
6	800	20	1000	20 → 7	2
7	800	20	5000	20 → 8, 10 → 9	0,1,2,3,4,5
8	800	40	2000		3
9	800	30	1000		2
10	1000	20	2000	30 → 11	0,1,2,3,4,5,8,9
11	1000	40	3000	10 → 12	0,1,2,3,4,5,8,9
12	1000	30	1400		9
13	1000	10	1000	10 → 14	4
14	1000	20	1000	20 → 15	0,1,2,3,4,5,8,9

Tabelle E.2: Komponente *Comp A1* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.5 auf Seite 225).

15	1000	60	4000	$20 \rightarrow 51$	0,1,2,3,4,5,8,9
16	500	30	2000	$20 \rightarrow 17$	8
17	500	20	1500		5,6,7,8,9,10
18	200	10	1500	$15 \rightarrow 19$	4
19	200	10	2000	$40 \rightarrow 53$	0,1,2,3,4,5

Tabelle E.3: Komponente *Comp A2* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.6 auf Seite 226).

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
20	100	10	2000		10,11,12,13,14,15
21	2000	30	1000	$10 \rightarrow 22$	15
22	2000	60	2000	$10 \rightarrow 23$	10,11,12,13,14,15
23	2000	40	1000	$20 \rightarrow 24, 10 \rightarrow 25$	10,11,12,13,14,15
24	2000	20	1000		11
25	2000	80	3000		12
26	800	20	1000	$20 \rightarrow 27$	12
27	800	20	5000	$20 \rightarrow 28, 10 \rightarrow 29$	10,11,12,13,14,15
28	800	40	2000		13
29	800	30	1000		12
30	1000	20	2000	$30 \rightarrow 31$	6,7,10,11,12,13,14,15
31	1000	40	3000	$10 \rightarrow 32$	6,7,10,11,12,13,14,15
32	1000	30	1400		7
33	1000	10	1000	$10 \rightarrow 34$	14
34	1000	20	1000	$20 \rightarrow 35$	6,7,10,11,12,13,14,15
35	1000	60	4000	$20 \rightarrow 56$	6,7,10,11,12,13,14,15
36	500	30	2000	$20 \rightarrow 37$	6
37	500	20	1500		5,6,7,8,9,10
38	200	10	1500	$15 \rightarrow 39$	14
39	200	10	2000	$40 \rightarrow 54$	10,11,12,13,14,15

Tabelle E.4: Komponente *Comp B* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.7 auf Seite 226).

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
40	1000	20	2000	$30 \rightarrow 41$	2,3,4,5,6,7,8,9
41	1000	40	3000	$10 \rightarrow 42$	2,3,4,5,6,7,8,9
42	1000	30	1400		9
43	1000	10	1000	$10 \rightarrow 44$	6
44	1000	20	1000	$20 \rightarrow 45$	8,9,10,11,12,13,14,15
45	1000	60	4000	$20 \rightarrow 58$	6,7,10,11,12,13,14,15

Tabelle E.4: Komponente *Comp B* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.7 auf Seite 226).

46	500	30	2000	20 → 47	8
47	500	20	1500		5,6,7,8,9,10
48	200	10	1500	15 → 49	8
49	200	10	2000	40 → 55	5,6,7,8,9

Tabelle E.5: Komponente *Comp Ctrl* der Fallstudie zur inkrementellen Integration (vgl. Abbildung 6.8 auf Seite 227).

Task	Periode	WCET	Speicher	Nachrichten	Platzierung
50	200	20	1000		3,4,5,6,7,8,9,10,11,12
51	1000	40	4000	50 → 52	
52	1000	30	1000	20 → 10	
53	200	20	2000	50 → 56	
54	200	20	2000	50 → 58	
55	200	20	2000	50 → 51	
56	1000	50	2000	40 → 57	
57	1000	30	1000	20 → 30	
58	200	10	2000	50 → 59	
59	1000	30	1000	20 → 40	

Neben den Taskset-Spezifikationen fehlt nun noch die Festlegung der End-to-End-Deadlines über die einzelnen Pfade in den Teilgraphen des Systems. Diese kann der folgenden textuellen Spezifikation im original Format entnommen werden:

```
e2e_spec
deadline: 20000
tasks: 4
T1
T2
T3
T4
end_e2espec

e2e_spec
deadline: 20000
tasks: 4
T1
T2
T3
T5
end_e2espec

e2e_spec
deadline: 8000
```

```
tasks: 3
T6
T7
T8
end_e2espec

e2e_spec
deadline: 8000
tasks: 3
T6
T7
T9
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T10
T11
T12
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T13
T14
T15
end_e2espec

e2e_spec
deadline: 5000
tasks: 2
T16
T17
end_e2espec

e2e_spec
deadline: 2000
tasks: 2
T18
T19
end_e2espec

e2e_spec
deadline: 1000
tasks: 1
T0
end_e2espec
```

```
e2e_spec
deadline: 20000
tasks: 4
T21
T22
T23
T24
end_e2espec

e2e_spec
deadline: 20000
tasks: 4
T21
T22
T23
T25
end_e2espec

e2e_spec
deadline: 8000
tasks: 3
T26
T27
T28
end_e2espec

e2e_spec
deadline: 8000
tasks: 3
T26
T27
T29
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T30
T31
T32
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T33
T34
T35
```

```
end_e2espec

e2e_spec
deadline: 5000
tasks: 2
T36
T37
end_e2espec

e2e_spec
deadline: 2000
tasks: 2
T38
T39
end_e2espec

e2e_spec
deadline: 1000
tasks: 1
T20
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T40
T41
T42
end_e2espec

e2e_spec
deadline: 10000
tasks: 3
T43
T44
T45
end_e2espec

e2e_spec
deadline: 5000
tasks: 2
T46
T47
end_e2espec

e2e_spec
deadline: 2000
tasks: 2
T48
```

```
T49
end_e2espec

e2e_spec
deadline: 10000
tasks: 2
T51
T52
end_e2espec

e2e_spec
deadline: 10000
tasks: 2
T56
T57
end_e2espec

e2e_spec
deadline: 10000
tasks: 2
T58
T59
end_e2espec

e2e_spec
deadline: 2000
tasks: 1
T50
end_e2espec

e2e_spec
deadline: 2000
tasks: 1
T53
end_e2espec

e2e_spec
deadline: 2000
tasks: 1
T54
end_e2espec

e2e_spec
deadline: 2000
tasks: 1
T55
end_e2espec
```

Die dafür vorgesehene Architektur bestehend aus 16 Prozessorknoten, die in einer hier-

archischen Topologie mit 3 Tokenring-Bussen angeordnet sind, kann Abbildung 6.9 auf Seite 227 entnommen werden. Die vollständige Architektur ist durch die folgende textuelle Spezifikation im Architekturformat definiert:

```
node_declaration:
ECU0 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU1 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU2 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU3 ( processor = MSPARC;
       memory = 22000 bytes; )
ECU4 ( processor = MSPARC;
       memory =  27000 bytes; )
ECU5 ( processor = MSPARC;
       memory =  27000 bytes; )
ECU6 ( processor = MSPARC;
       memory = 22000 bytes; )
ECU7 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU8 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU9 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU10 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU11 ( processor = MSPARC;
       memory = 20000 bytes; )
ECU12 ( processor = MSPARC;
       memory = 22000 bytes; )
ECU13 ( processor = MSPARC;
       memory =  27000 bytes; )
ECU14 ( processor = MSPARC;
       memory =  27000 bytes; )
ECU15 ( processor = MSPARC;
       memory = 22000 bytes; )

comm_declaration:
Tok_1 ( network = Tokenring;
        rate = 50 MBsec;
        header = 0;
        databytes = 0; )

Tok_2 ( network = Tokenring;
        rate = 100 MBsec;
        header = 0;
        databytes = 0; )
```

```
Tok_3 ( network = Tokenring;
        rate = 50 MBsec;
        header = 0;
        databytes = 0; )


connection:
ECU0 ( ECU1: {Tok_1};
       ECU2: {Tok_1};
       ECU3: {Tok_1};
       ECU4: {Tok_1};
       ECU5: {Tok_1}; )

ECU1 ( ECU0: {Tok_1};
       ECU2: {Tok_1};
       ECU3: {Tok_1};
       ECU4: {Tok_1};
       ECU5: {Tok_1}; )

ECU2 ( ECU1: {Tok_1};
       ECU0: {Tok_1};
       ECU3: {Tok_1};
       ECU4: {Tok_1};
       ECU5: {Tok_1}; )

ECU3 ( ECU1: {Tok_1};
       ECU2: {Tok_1};
       ECU0: {Tok_1};
       ECU4: {Tok_1};
       ECU5: {Tok_1}; )

ECU4 ( ECU1: {Tok_1};
       ECU2: {Tok_1};
       ECU3: {Tok_1};
       ECU0: {Tok_1};
       ECU5: {Tok_1}; )

ECU5 ( ECU1: {Tok_1};
       ECU2: {Tok_1};
       ECU3: {Tok_1};
       ECU4: {Tok_1};
       ECU0: {Tok_1};
       ECU6: {Tok_2};
       ECU7: {Tok_2};
       ECU8: {Tok_2};
       ECU9: {Tok_2};
       ECU10: {Tok_2}; )

ECU6 ( ECU5: {Tok_2};
```

```
       ECU7: {Tok_2};
       ECU8: {Tok_2};
       ECU9: {Tok_2};
       ECU10: {Tok_2}; )

ECU7 ( ECU5: {Tok_2};
       ECU6: {Tok_2};
       ECU8: {Tok_2};
       ECU9: {Tok_2};
       ECU10: {Tok_2}; )

ECU8 ( ECU5: {Tok_2};
       ECU7: {Tok_2};
       ECU6: {Tok_2};
       ECU9: {Tok_2};
       ECU10: {Tok_2}; )

ECU9 ( ECU5: {Tok_2};
       ECU7: {Tok_2};
       ECU8: {Tok_2};
       ECU6: {Tok_2};
       ECU10: {Tok_2}; )

ECU10 ( ECU5: {Tok_2};
        ECU7: {Tok_2};
        ECU8: {Tok_2};
        ECU9: {Tok_2};
        ECU6: {Tok_2};
        ECU11: {Tok_3};
        ECU12: {Tok_3};
        ECU13: {Tok_3};
        ECU14: {Tok_3};
        ECU15: {Tok_3}; )

ECU11 ( ECU10: {Tok_3};
        ECU12: {Tok_3};
        ECU13: {Tok_3};
        ECU14: {Tok_3};
        ECU15: {Tok_3}; )

ECU12 ( ECU10: {Tok_3};
        ECU11: {Tok_3};
        ECU13: {Tok_3};
        ECU14: {Tok_3};
        ECU15: {Tok_3}; )

ECU13 ( ECU10: {Tok_3};
        ECU12: {Tok_3};
        ECU11: {Tok_3};
```

```
        ECU14: {Tok_3};
        ECU15: {Tok_3}; )

ECU14 ( ECU10: {Tok_3};
        ECU12: {Tok_3};
        ECU13: {Tok_3};
        ECU11: {Tok_3};
        ECU15: {Tok_3}; )

ECU15 ( ECU10: {Tok_3};
        ECU12: {Tok_3};
        ECU13: {Tok_3};
        ECU14: {Tok_3};
        ECU11: {Tok_3}; )
```

Literaturverzeichnis

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley and Sons, 1989.

[2] T. Abdelzaher and K. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 162–171, 1995.

[3] E. Abraham and C. Herde. Accelerated Bounded Model CHecking for Hybrid Systems. In *Proceedings of the 1st German Verification Day*, 2005.

[4] Alcatel Mietec. *Standard Cell Design Manual*, 1996.

[5] P. Altenbernd. On the false path problem in hard real-time programs. In *Proceedings of the Eight Euromicro Workshop on Real-Time Systems*, pages 102–107, 1996.

[6] P. Altenbernd. *Timing Analysis, Scheduling and Allocation of Periodic Hard Real-Time Tasks.* PhD thesis, Universität Paderborn, 1996.

[7] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The TERA Computer System. In *Proceedings of the International Conference on Supercomputing*, 1990.

[8] N. Audsley and A. Burns. On Fixed Priority Scheduling, Offsets and Co-Prime Task Periods. *Inf. Process. Lett.*, 67(2):65–69, 1998.

[9] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, 1991.

[10] F. Balarin, Y. Watanabe, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *IEEE Computer*, 36(4):45–52, 2003.

[11] P. Barth. A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1995.

[12] S. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. Technical report, Department of Computer Science, University of North Carolina, 2003.

[13] J. Beck and D. Siewiorek. Simulated Annealing Applied to Multicomputer Task Allocation and Processor Specification. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, pages 232–239, 1996.

[14] A. Bender. Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System. In *EDTC '96: Proceedings of the 1996 European conference on Design and Test*. IEEE Computer Society, 1996.

[15] A. Bender. MILP Based Task Mapping for Heteregeneous Multiprocessor Systems. In *Proceedings of the EURO-DAC with EURO-VHDL*, 1996.

[16] G. Bernat and A. Burns. New Results on Fixed Priority Aperiodic Servers. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 68–78, 1999.

[17] J. Berwanger and T. Schumm. Introducing FlexRay at BMW. In *ARTIST2 Workshop on Semantic Platforms and Merging ET with TT*, 2005.

[18] R. Bettati and J. Liu. End-to-end scheduling to meet deadlines in distributed systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 452–459, 1992.

[19] BMW AG. *byteflight Specification*, 1999.

[20] S. Bokhari. A Network Flow Model for Load Balancing in Circuit-Switched Multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):649–657, 1993.

[21] I. Bomze and W. Grossmann. *Optimierung - Theorie und Algorithmen.* BI-Wissenschafts-Verlag, 1993.

[22] J. Le Boudec. The Asynchronous Transfer Mode: Tutorial. *Computer Networks and ISDN Systems*, 24:279ff, 1992.

[23] W. Brauch, H. Dreyer, and W. Haacke. *Mathematik fü Ingenieure.* B.G. Teubner, 1995.

[24] I. Broster, A. Burns, and G. Rodriguez-Navas. Probabilistic Analysis of CAN with Faults. In *IEEE Real Time Systems Symposium*, 2002.

[25] M. Broy, M. Beeck, and I. Krüger. SOFTBED: Problemanalyse für ein Großverbundprojekt: Systemtechnik Automobil — Software für eingebettete Systeme. Technical report, Department of Computer Science, Technische Universität München, 1998.

[26] J. V. Busquets-Mataix and A. Wellings. Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems. In *Proceedings of the 2nd Real-Time Technology and Applications Symposium*, 1996.

[27] H. Cambazard, P. Hladik, A. Deplanche, N. Jussien, and Y. Trinquet. Decomposition and Learning for a Hard Real-Time Task Allocation Problem. In *Proceedings of the*, volume 3258 of *Lecture Notes in Computer Science*, pages 153–167, 2004.

[28] R. Chapman, A. Wellings, and A. Burns. Integrated Program Proof and Worst-Case Timing Analysis of SPARK Ada. In *Proceedings of the Workshop on Language, Compiler, and Tool Support for Real-Time Systems*, 1994.

[29] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[30] W. Chu and L. Lan. Task Allocation and Precedence Relations for Distributed Real-Time Systems. *IEEE Transaction on Computers*, 36(6):667–679, 1987.

[31] W. Chu and K. Leung. Module Replication and Assignment for Real-Time Distributed Processing Systems. *Proceedings of the IEEE*, 75(5):547–562, 1987.

[32] V. Claus and A. Schwill. *Duden Informatik*. Dudenverlag, 1988.

[33] J. Coffman. *Computer and Job-Shop Scheduling Theory*. John Wiley, 1976.

[34] A. Colin and I. Puaut. Worst Case Execution Time Analysis for a Processor with Branch Prediction. *Real-Time Systems*, 18(2-3):249–274, 2000.

[35] A. Colin and I. Puaut. A modular and retargetable framework for tree-based wcet analysis. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, 2001.

[36] Cypress Semiconductor Corporation. *CY7C006: Dual-Port Static RAM Datasheet*, 1996.

[37] W. Damm, E. Böde, A. Metzner, T. Peikenkamp, and A. Votintseva. Rich component models. Technical report, Department of Computer Science, Carl-von-Ossietzky University Oldenburg, 2005.

[38] W. Damm, A. Metzner, A. Mikschl, and J. Niehaus. Die EVENTS Architektur. *it + ti*, 42(2):40–44, 2000.

[39] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde. Boosting Re-use of Embedded Automotive Applications Through Rich Components. In *Proceedings of the Foundation of Interface Technology Workshop*, 2005.

[40] J. Dannenberg and C. Kleinhans. The Coming Age of Collaboration in the Automotive Industry. *Mercer Management Journal*, 17:88–94, 2004.

[41] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5:394–397, 1962.

[42] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5:394–397, 1962.

[43] S. Dhall and C. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26(1):127–140, 1978.

[44] G. Ehmen. Rekonfigurierbarer Hardware Real-Time Scheduler. Technical report, Department of Computer Science, Carl-von-Ossietzky Universität Oldenburg, 2004.

[45] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System-Level Hardware/Software Partitioning based on Simulated Annealing and Tabu Search. *Design Automation for Embedded Systems*, 4(4), 1997.

[46] J. Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, 2002.

[47] A. Ermedahl and J. Gustafsson. Deriving Annotations for Tight Calculation of Execution Time. In *Proceedings of the Third International Euro-Par Conference on Parallel Processing*, pages 1298–1307. Springer-Verlag, 1997.

[48] E. Erpenbach and P. Altenbernd. Worst-Case Execution Times and Schedulability Analysis of Statecharts Models. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 1999.

[49] Bundesministerium des Inneren EStdIT. V-Model, Development Standard for IT-Systems of the federal Republic of Germany, 1997.

[50] Bundesministerium des Inneren EStdIT. V-Model: Method Allocation, 1997.

[51] F. Balarin et al. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Publisher, 1997.

[52] W. Damm et al. Seamless Design Flow for Automotive Electronic Systems with Architecture Design Exploration Emphasis (SEA). Technical report, Department of Computer Science, Carl-von-Ossietzky University Oldenburg, 2004.

[53] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and Precise WCET Determination for a Real-Life Processor. In *Proceedings of the First International Workshop on Embedded Software*, pages 469–485. Springer-Verlag, 2001.

[54] C. Ferdinand, F. Martin, and R. Wilhelm. Applying Compiler Techniques to Cache Behavior Prediction. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*, pages 37–46, 1997.

[55] The FlexRay group. *FlexRay Requirement Specification, Version 0.1.*

[56] The FlexRay group. *FlexRay Communication Systems Protocol Specification, Version 2.0*, 2004.

[57] M. Fränzle and C. Herde. Efficient SAT engines for concise logics: Accelerating proof search for zero-one linear constraint systems. In M. Vardi and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2003)*, volume 2850 of *LNAI*. Springer, 2003.

[58] M. Fränzle and C. Herde. Efficient proof engines for bounded model checking of hybrid systems. In *Ninth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 04)*, Electronic Notes in Theoretical Computer Science (ENTCS). Elsevier, 2004.

[59] H. Frischkorn. Automotive Software — The Silent Revolution. In *Proceedings of the Automotive Software Workshop*, 2004.

[60] FSMLabs. *RTLinux Real-Time Operating System*, 2004.

[61] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on can (ttcan). In *Proceedings of the ICC'2000*, 2000.

[62] G. Garcia and G. Harbour. Optimal priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, pages 124–134, 1995.

[63] F. Glover and M. Laguna. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley Sons, 1993.

[64] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver, 2002.

[65] K. Gresser. An Event Model for Deadline Verification of Hard Real-Time Systems. In *Proceedings of the Euromicro Workshop on Real-Time Systems*, pages 118–123, 1993.

[66] P. Gutiérrez and M. Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *IEEE Real-Time Systems Symposium*, pages 328–339, 1999.

[67] R. Haniak, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis — the SymTA/S Approach. In *Proceedings of the IEE Computers and Digital Techniques*, 2005.

[68] C. Healy, R. Arnold, F. Mueller, M. Harmon, and D. Walley. Bounding Pipeline and Instruction Cache Performance. *IEEE Transaction on Computers*, 48(1):53–70, 1999.

[69] H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Maté, K. Nishikawa, and T. Scharnhorst. AUTomotive Open System ARchitecture — an industry-wide initiative to manage the complexity of emerging automotive E/E-architectures. In *Proceed. Convergence 2004, International Congress on Transportation Electronics, Detroit*, 2004.

[70] R. Henia and R. Ernst. Context-Aware Scheduling Analysis of Distributed Systems with Tree-Shaped Task-Dependencies. In *DATE*, pages 480–485. IEEE Computer Society, 2005.

[71] T. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society, 1996.

[72] C. Herde. Hysat-dokumentation. Technical report, Department of Computer Science, Carl-von-Ossietzky Universität Oldenburg, 2005.

[73] C. Herde. *Efficient Decision Procedures for Bounded Model Checking and Inductive Verification of Hybrid Systems*. PhD thesis, Carl-von-Ossietzky Universität Oldenburg, to appear 2006.

[74] C. Hopper and Y. Pan. Task Allocation in Distributed Computer Systems Through an AI Planner Solver. In *Proceedings of the National Aerospace and Electronics Conference*, 1995.

[75] SPARC International Inc., editor. *The SPARC Architectural Manual*, volume 8. Prentice Hall, 1992.

[76] The Institute of Electrical and Electronics Engineers. *IEEE Standard 1003.13-1998*, 1998.

[77] International Council on Systemes Engineering (INCOSE). *Systems Engineering Handbook*, 2000.

[78] The International Engineering Consortium. *Time Devision Multiple Access (TDMA)*, 2003.

[79] B. Ip. Performance Analysis of VxWorks and RTLinux. Technical report, Department of Computer Science, Columbia University, 2002.

[80] J. Jonsson and K. Shin. Robust adaptive metrics for deadline assignment in distributed hard real-time systems. *Real-Time Systems*, 23:239–271, 2002.

[81] P. Jorgenson and J. Madsen. Critical Path Driven Co-Synthesis for Heterogeneous Target Architectures. In *Proceedings of the International Workshop on Hardware-Software Co-design*, pages 15–19, 1997.

[82] M. Joseph, editor. *Real-time Systems Specification, Verification and Analysis*. Prentice Hall International, London, 1996.

[83] M. Joseph and P. Pandya. Finding Response Times in Real-Time Systems. *BCS Computer Journal*, 29(5):390–395, 1986.

[84] P. Wältermann K. Lamberg. Einsatz der HIL-Simulation beim Test von Mechatronik-Komponenten im Automobil. In *2. Tagung Mechatronik im Automobil*, 2000.

[85] H. Kasahara and S. Narita. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. *IEEE Transaction on Computers*, 33(11):1023–1029, 1984.

[86] D. Kästner and S. Thesing. Cache Sensitive Pre-Runtime Scheduling. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, volume 1474 of *LNCS*, pages 131–145, 1998.

[87] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[88] D.B. Kirk. SMART (strategic memory allocation for real-time) cache design. In *Proceedings of the 10th Real-Time Systems Symposium*, 1989.

[89] R. Kirner and P. Puschner. Transformation of path information for wcet analysis during compilation. In *Proc. 13th Euromicro International Conference on Real-Time Systems*, pages 29–36, 2001.

[90] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Harbour. *A Practitioners Handbook for Real-Time Analysis*. Kluwer Academic Publishers, 1993.

[91] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 2001.

[92] H. Kopetz and R. Nossal. The Cluster-Compiler — A Tool for the Design of Time Triggered Real-Time Systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.

[93] J. Kreuzinger, A. Schulz, M. Pfeffer, T. Ungerer, U. Brinkschulte, and C. Krakowski. Real-time scheduling on multithreaded processors. In *Real-Time Computing Systems and Applications*, 2000.

[94] J. Laudon, A. Gupta, and M. Horowitz. *Multithreaded Computer Architecture: A Summary of the State of the Art*. Kluwer Academic Publishers, 1994.

[95] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemtive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998.

[96] J. Lee, V. Mooney, A. Daleby, K. Ingström, T. Klevin, and L. Lindh. A comparison of the rtu hardware rtos with a hardware/software rtos. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 683–688, 2003.

[97] J. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real Time Systems Symposium*, pages 201–209, 1990.

[98] J. Lehoczky, L. Sha, and D. Ding. The Rate Monotonic Scheduling Algorithm: Exact characterization and Average Case Behaviour. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 261–270, 1989.

[99] J. Lehoczky and S. Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 110–123, 1992.

[100] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. In *Performance Evaluation*, volume 2, 1982.

[101] Y. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, & tools for real-time systems*, pages 88–98. ACM Press, 1995.

[102] Y. Li, E. Richards, Y. Jiang, and N. Azarmi. Localized Simulated Annealing in Constraint Satisfaction and Optimization. In *Modern Heuristic Search Methods*. John Wiley Sons, 1996.

[103] S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, S. Moon, and C. Kim. An Accurate Worst Case Timing Analysis for RISC Processors. *IEEE Transaction on Software Engineering*, 21(7):593–604, 1995.

[104] C. Liu and Layland J. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[105] F. Liu, T. Peikenkamp, and W. Damm. An extended gradient model for NUMA multiprocessors. In *Algorithms, Concurrency and Knowledge*, volume 1023 of *Lecture Notes in Computer Science*, pages 210–224, 1995.

[106] lpsolve. http://www.cs.sunysb.edu/∽algorith/implement/lpsolve.

[107] L. Lüth, A. Metzner, and J. Niehaus. A statemate-based rapid prototyping environment. In *6. Deutsches Anwenderforum für Statemate*, 1998.

[108] M. Harbour M. Rivas. Evaluation of New POSIX Real-Time Operating Systems Services for Small Embedded Platforms. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 2003.

[109] P. Ma. A Task Allocation Model for Distributed Computing Systems. *IEEE Transaction on Computers*, 31(1):41–47, 1982.

[110] J. Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In *Proceedings of the 9th Conference on Artificial Intelligence*, 1999.

[111] R. Mazur. Übersetzung integer-arithmetischer Gleichungen und Ungleichungen in SAT-Instanzen. Master's thesis, Carl-von-Ossietzky Universität Oldenburg, 2004.

[112] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[113] C. Meinel and T. Theobald. *Algorithmen und Datenstrukturen im VLSI-Design — OBDD-Grundlagen und Anwendungen*. Springer Verlag, 1998.

[114] A. Metzner. Dokumentation der MSparc. Technical report, Department of Computer Science, Carl-von-Ossietzky University Oldenburg, 1999.

[115] A. Metzner. Skriptum zur Vorlesung *Entwurf von Realzeit-Systemen*, 2002.

[116] A. Metzner. Incremental Task Allocation: Integrating Real-Time Software in Distributed Embedded Systems. In *Proceeding of the* 16^{th} *International Conference on Software and Systems Engineering and their Application*, 2003.

[117] A. Metzner. Why Model Checking Can Improve WCET Analysis. In *Proceeding of the* 16^{th} *Internation Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, 2004.

[118] A. Metzner and T. Bienmüller. Embedded Control Mode der MSparc: Spezifikation und Implementierung. Technical report, Department of Computer Science, Carl-von-Ossietzky University Oldenburg, 1997.

[119] A. Metzner and J. Niehaus. MSPARC: Multithreading in Real-Time Architectures. *Journal of Universal Computer Science*, 6(10):1034–1051, 2000.

[120] A. Mikschl and W. Damm. MSparc: A Multithreaded SPARC. *Lecture Notes on Computer Science*, 1996.

[121] A. Millsap. Automotive embedded systems issues, a GM perspective. In *ARTIST2 Workshop on Semantic Platforms and Merging ET with TT*, 2005.

[122] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[123] Motorola. *MPC555: The MPC500 Family of 32-Bit Embedded Controllers*, 2001.

[124] F. Mueller. Compiler Support for Software-Based Cache Partitioning. In *Workshop on Languages, Compilers, & Tools for Real-Time Systems*, pages 125–133, 1995.

[125] A. Müller. *Entwurfsmethodik und automatisierte Verteilung für Steuerungssoftware in einem verteilten Rechensystem der Automobilelektronik*. PhD thesis, Universität Tübingen, 1999.

[126] M. Naedele, L. Thiele, and M. Eisenring. Characterizing Variable Task Releases and Processor Capacities. Technical Report TIK-45, Computer Engineering and Networks Laboratory, ETH Zürich, 1999.

[127] M. Di Natale and J. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 216–227, 1994.

[128] N. Navet and Y. Song. Validation of In-Vehicle Real-Time Applications. *Computers in Industry*, 46(2):107–122, 2001.

[129] K. Neumann. *Partitionierung verteilter Echtzeitfunktionen in vernetzten Systemen.* PhD thesis, Universität Karlsruhe, 1998.

[130] M. Nicholson, J. McDermid, and A. Burns. Analysis and Design Synthesis for Hard Real-Time Safety Critical Systems. Technical Report YCS-94-237, Department of Computer Science, University of York, 1994.

[131] E. Norden. *A Multithreading Extension for Low Power, Low Cost Application.* Infinion, 2002.

[132] University Oldenburg, University Freiburg, and University Saarbrücken. Automatic Verification and Analysis of Complex Systems — SFB/Transregio 14. Technical report, http://www.avacs.org, 2004.

[133] The OSEK group. *OSEK/VDX: Operating System Version 2.2*, 2001.

[134] The OSEK group. *OSEK/VDX: Time Triggered Operating System Version 1.0*, 2001.

[135] The OSEK group. *OSEK/VDX: Communication Version 3.0*, 2002.

[136] The OSEK group. *OSEK/VDX: OSEK Implementation Language Version 2.4*, 2002.

[137] B. Pagel and H. Six. *Software Engineering.* Addison-Wesley, 1994.

[138] C. Park. Predicting program execution times by analyzing static and dynamic program paths. *Real-Time Systems*, 5(1):31–62, 1993.

[139] D. Peng, K. Shin, and T. Abdelzaher. Assignment and Scheduling Communiating Periodic Tasks in Distributed Real-Time Systems. *IEEE Transaction on Software Engineering*, 23(12):745–758, 1997.

[140] S. Petters and G. Färber. Making Worst Case Execution Time Analysis for Hard Real-Time Tasks on State of the Art Processors Feasible. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications.* IEEE Computer Society, 1999.

[141] C. Piguet, editor. *High-Level Power Extimation and Analysis*, 2005.

[142] T. Plaks, editor. *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms.* CSREA Press, 2003.

[143] P. Pop, P. Eles, and Z. Peng. Schedulability-driven frame packing for multi-cluster distributed embedded systems. In *ACM Transactions on Embedded Computing Systems*, 2004.

[144] P. Pop, P. Eles, T. Pop, and Z. Peng. An Approach to Incremental Design of Distributed Embedded Systems. In *Proceedings of the Conference on Design Automation.* IEEE Computer Society, 2001.

[145] T. Pop, P. Eles, and Z. Peng. Holistic Scheduling and Analysis of Mixed Time/Event Triggered Distributed Embedded Systems. In *Proceeding of the International Symposium on Hardware/Software Codesign*, 2002.

[146] R. Rajkumar, L. Sha, J. Lehoczky, and K. Ramamritham. *An Optimal Priority Inheritance Policy for Synchronization in Real-Time Systems*, chapter 11. Prentice Hall, 1995.

[147] R. Rajkumar, L. Sha, J.P. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.

[148] K. Ramramritham. Allocation and Scheduling of Precedence-Relates Periodic Tasks. *IEEE Transaction on Parallel and Distributed Systems*, 6(4):412–420, 1995.

[149] J. Real and A. Crespo. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems*, 26:161–197, 2004.

[150] I. Rechenberg. *Evolutionsstrategie*. Friedrich Frommann Verlag, 1972.

[151] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proceedings of the* 39^{th} *Design Automation Conference*. ACM Press, 2002.

[152] Robert Bosch GmbH. *CAN Specification, Version 2.0*, 2003.

[153] F. Sandnes. A Hybrid Genetic Algorithm Applied to Automatic Parallel Controller Code Generation. In *Proceedings of the IEEE Euromicro Workshop on Real-Time Systems*, 1996.

[154] K. Sandström and C. Norström. Frame packing in real-time communication. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, 2000.

[155] V. Sarkar and J. Hennesey. Compile-Time Partitioning and Scheduling of Parallel Programs. *Symposium on Compiler Construction*, 21(7):17–26, 1986.

[156] A. Schallenberg. Analyse der Ausführungszeiten von Realzeit Tasks auf abstrakten Prozessoren. Master's thesis, Carl-von-Ossietzky Universität Oldenburg, 2001.

[157] B. Schätz, J. Romberg, O. Slotosch, M. Strecker, A. Weisspeintner, T. Hain, W. Prenninger, M. Rappl, and K. Spies. Modeling Embedded Software: State of the Art and Beyond. In *Proceeding of the* 16^{th} *International Conference on Software and Systems Engineering and their Application*, 2003.

[158] J. Schneider. Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, 2000.

[159] A. Scholl, G. Krispin, R. Klein, and W. Domschke. Besser beschränkt — Clever optimieren mit Branch & Bound. *c't Magazin für Computer Technik*, 10(1):336–354, 1997.

[160] P. Scholz and E. Harbeck. Task Assignment for Distributed Computing. In *Proceedings of the Conference on Advances in Parallel and Distributed Systems*, pages 270–277, 1997.

[161] E. Schöneburg, F. Heinzmann, and S. Fedderson. *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, 1994.

[162] L. Sha, T. Abdelzaher, K. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2-3):101–155, 2004.

[163] L. Sha, J. Lehoczky, and R. Rajkumar. Task Scheduling in Distributed Real-Time Systems. In *Proceedings of the IEEE Industrial Electronics Conference*, 1987.

[164] C. Shen and W. Tsai. A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion. *IEEE Transaction on Computers*, 34(3):197–203, 1985.

[165] K. Shin and C. Hou. Evaluation of Load Sharing in HARTS with Consideration of Its Communication Activities. *IEEE Transaction on Parallel and Distributed Systems*, 7(7):724–739, 1996.

[166] G. Shmonin, A. Metzner, S. Winkel, and F. Eisenbrandt. Scheduling and Distribution of Real-Time Tasks by ILP. Technical report, University Oldenburg, University Saarbrücken, MPI Saarbrücken, 2005.

[167] J. Sinclair. Efficient Computation of Optimal Assignments for Distributed Tasks. *Journal on Parallel and Distributed Computing*, 4:342–362, 1987.

[168] F. Slomka, K. Albers, and R. Hofmann. A Multiobjective Tabu Search Algorithm for the Design Space Exploration of Embedded Systems. In *IFIP Working Conference on Distributed and Parallel Embedded Systems*, 2004.

[169] B. Sprunt. *Aperiodic Task Scheduling for Real-Time Systems*. PhD thesis, Carnegie Mellon University, 1990.

[170] J. A. Stankovic and K. Ramamritham. *Hard Real-Time Systems*. IEEE Computer Society Press, 1988.

[171] I. Stierand. *Ambrosia/MP — Ein Echtzeitbetriebsystem für eingebettete Mehrprozessorsysteme*. PhD thesis, Carl-von-Ossietzky Universität Oldenburg, 2001.

[172] I. Stierand and A. Metzner. Anforderungskatalog für die erstellung einer toolbasis zur scheduling-analyse. Technical report, Department of Computer Science, Carl-von-Ossietzky University Oldenburg, 2005.

[173] I. Stierand and A. Metzner. Mode change protocols for distributed real-time systems. to appear, 2005.

[174] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. In *IEEE Real-Time Systems Symposium*, 1996.

[175] H. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithm. *IEEE Transactions on Software Engineering*, 3(1):85–93, 1977.

[176] H. Stone and S. Bokhari. Control of Distributed Processes. *Computer*, 1:97–106, 1978.

[177] J. Strosnider, J. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsivness in Real-Time Environments. *IEEE Transaction on Computers*, 4(1):73–91, 1995.

[178] E. Thaden. Vergleich der Echtzeiteigenschaften von Time-triggered und Event-basierten Bussystemen. Master's thesis, Carl-von-Ossietzky Universität Oldenburg, 2003.

[179] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics. In *Proceedings of the International Performance and Dependability Symposium (IPDS)*, 2003.

[180] C. Thomas. Skriptum zur Vorlesung *Systems Engineering*, 2005.

[181] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, 1994.

[182] K. Tindell and A. Burns. Guaranteeing Message Latencies on Controller Area Network (CAN). In *Proceedings of the 1st International CAN Conference*, 1994.

[183] K. Tindell, A. Burns, and A. Wellings. Allocating hard real time tasks (an np-hard problem made easy). In *Real Time Systems Journal*, volume 4(2), 1992.

[184] K. Tindell, A. Burns, and A. Wellings. An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[185] K. Tindell, A. Burns, and A. G. Wellings. Mode Changes in Priority Pre-emptively Scheduled Systems. In *Proc. Real Time Systems Symposium*, pages 100–109, Phoenix, Arizona, Dec. 1992.

[186] K. Tindell and H. Hansson. Real-time systems and fixed priority scheduling. Technical report, Department of Computer Science, Uppsala University, 1995.

[187] B. Tsai and K. Shin. Assignment of Task Modules in Hypercube Multicomputers with Component Failures for Communication Efficiency. *IEEE Transaction on Computers*, 43(5):613–618, 1994.

[188] G. Tseitin. On the complexity of derivations in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, 1968.

[189] TTECH Computertechnik AG. *Specification of the TTP/C Protocol*, 1999.

[190] D. Tullsem, S. Eggers, and H. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *Proceedings of the Internation Symposium on Computer Architecture*, 1995.

[191] R. v. Hanxleden, A. Botorabi, and S. Kupczyk. A Co-Design Approach for Safety-Critical Automotive Applications. *IEEE Micro Special Issue on Embedded Fault-Tolerant Systems*, 18(5):66–79, Sep/Oct 1998.

[192] A. Votintseva, S. Gröning, A. Metzner, and I. Stierand. Real-Time Automata Specification for a Rich Component Model. Technical report, Department of Computer Science, Carl-von-Ossietzky Universität Oldenburg, 2005.

[193] M. Wannemacher. *Das FPGA Kochbuch.* iwt, 1998.

[194] W. Weber and A. Gupta. Exploring the benefits of multiple hardware contexts in a multiprocessor architecture: Preliminary results. In *Proceedings of the 16th International Symposium on Computer Architecture*, pages 273–280, 1989.

[195] K. Weiß, T. Steckstor, W. Rosenstiel, and C. Nietsch. Performance analysis of real-time operating systems by emulation of an embedded system. *Rapid System Prototyping*, 1999.

[196] E. Wells and C. Caroll. An Augmented Approach to Task Allocation: Combining Simulated Annealing with List-Based Heuristics. In *Proceedings of the Euromicro Workshop*, pages 508–515, 1993.

[197] R. White, F. Mueller, C. Healy, D. Whalley, and M. Harmon. Timing analysis for data caches and set-associative caches. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 192–202, 1997.

[198] Wind River. *VxWorks 5.X Real-Time Operating System*, 2003.

[199] G. Wittich. *Ein problem-orientierter Ansatz zum Nachweis von Realzeiteigenschaften eingebetteter Systeme*. PhD thesis, Carl-von-Ossietzky Universität Oldenburg, 1999.

[200] H. Wittke. *An Environment for Compositional Specification Verification of Complex Embedded Systems*. PhD thesis, Carl-von-Ossietzky Universität Oldenburg, 2005.

[201] F. Wolf and R. Ernst. Intervals in software execution cost analysis. In *Proceedings of the 13th international symposium on System synthesis*, pages 130–135. IEEE Computer Society, 2000.

[202] A. Wolfe. Software-based cache partitioning for real-time applications. In *Third International Workshop on Responsive Computer Systems*, 1993.

[203] Xilinx Inc. *SPARTAN FPGA Family: Datasheet*, 2003.

[204] J. Xu. Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence and Exclusion Relations. *IEEE Transaction on Software Engineering*, 19(2):139–154, 1993.

[205] S. Yoo and H. Youn. An Efficient Task Allocation Scheme for Two Dimensional Mesh-Connected Systems. In *Proceedings of the International Conference on Distributed Computer Systems*, pages 501–508, 1995.

[206] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proc. of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, November 2001.

[207] W. Zhen, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and Scalable Time Triggered Scheduling. In *Proceedings of the International Conference on Application of Concurrency to System Design*, 2005.

[208] K. M. Zuberi, P. Pillai, K. G. Shin, W. Nagaura, T. Imai, and S. Suzuki. EMERALDS-OSEK: A Small Real-Time Operating System for Automotive Control and Monitoring. In *Society of Automotive Engineers Congress and Exposition*, 1999.

Lebenslauf

Name:	Alexander Metzner
Geburtsdatum/-ort:	05.01.1969 in Bremen
Familienstand:	verheiratet, 3 Kinder
Wohnort:	Ahornweg 4, 26919 Brake

Schulischer Werdegang:	
1975-1979	Grundschule Kirchhammelwarden
1979-1981	Orientierungsstufe Brake Süd
1981-1989	Gymnasium Brake
Grundwehrdienst:	
1989-1990	FlaRakBtl 26, Stadland
Studium:	
1990-1996	Diplomstudiengang Informatik an der Universität Oldenburg Schwerpunkt: Technische Informatik Nebenfach: Physik (Schwerpunkt Optik) Abschluss als Diplom-Informatiker mit der Note Sehr Gut
Diplomarbeit:	
1995	VLSI-Implementierung einer multithreaded RISC-Architektur
Beruflicher Werdegang:	
1993-1995	Wissenschaftliche Hilfskraft für die Lehre
1996-2005	Wissenschaftlicher Mitarbeiter an der Universität Oldenburg, Abteilung SES
Seit 08/2005	OFFIS, Bereich SC

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel verfasst habe.
