Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik

Synthese von deadline-konformen Protokollkonvertern für heterogene verteilte Anwendungen

Studienarbeit

Studiengang Informatik

von

Dieter Averberg

Alte Brauerei 19

33098 Paderborn

betreut durch

Dipl.-Inform. Stefan Ihmor

vorgelegt bei

Prof. Dr. rer. nat. Franz Josef Rammig

im

März 2005

Dank und Erklärung

Diese Studienarbeit ist in der Arbeitsgruppe von Prof. Dr. rer. nat. Franz Josef Rammig (HNI) der Universität Paderborn entstanden. Durch die Anfertigung dieser Arbeit, habe ich gelernt Problemstellungen in der Informatik zielstrebig zu studieren.

An dieser Stelle möchte ich mich für das interessante Thema der Arbeit bei Prof. Dr. Franz J. Rammig bedanken. Mein Dank gilt allen, die mir bei der Erstellung dieser Arbeit zur Seite standen. Besonders möchte ich mich bei meinem Betreuer, Stefan Ihmor bedanken, der mich während des Erstellungszeitraums immer fachlich und engagiert betreut hat.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Paderborn, 8. März 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Der IP (Intellectual Property) basierte Systementwurf sowie Soc-Design (System-on-Chip Design) werden häufig dadurch erschwert, dass die zur Verfügung stehende Hardware unterschiedlicher Hersteller verschiedenste Schnittstellen besitzen oder unterschiedliche Protokolle zur Kommunikation voraussetzen. Für das zu entwickelnde System kann dann oft nur Hardware mit kompatiblen Schnittstellen ausgewählt werden oder man muss entsprechende Adapter als Protokollkonverter einbinden. Dadurch wird es erschwert, die für ein Projekt benötigten Komponenten zusammenzufügen.

Mit dem Ziel das Problem beherrschbar zu machen, wurde das Forschungsthema IFS (Interface Synthese) gegründet [2]. Die Interface Synthese befasst sich mit der automatisierten Erzeugung eines Adaptermoduls namens Interface Block (IFB). Mit dessen Hilfe können Schnittstellen inkompatibler Hardware verbunden werden. Dabei wird die Information der Protokolle für die entsprechende Gegenstelle übersetzt.

Der IFB basiert auf einer Makro-Struktur die bisher aus zwei Protocol Handlern, einem Sequence Handler und einer Steuerung, der Control Unit besteht. Die beiden Protocol Handler kommunizieren mit der fremden Hardware, den Tasks, während der Sequence Handler die Daten des einen Protokolls in die des andere Protokolls umwandelt.

Die bisher benutzte Architektur der Makro-Struktur verfügt über eine generische Anzahl an vordefinierten Leitungen und erlaubt keine Echtzeitkommunikation. Zudem konnten angeschlossene Tasks nicht duplex fähig sein, was die Menge der nutzbaren Hardwarekomponenten stark einschränkte. Weiterhin musste für bestimmte Teile der Makro-Struktur handgeschriebener VHDL-Programmcode eingefügt werden. Damit wurden Systemmodifikationen sehr aufwendig.

1.2 Aufgabenstellung

Im Rahmen dieser Studienarbeit soll der IFB zu einem deadline-konformen Protokollkonverter für heterogene verteilte Anwendungen erweitert werden. Die dazu notwendigen Architekturerweiterungen und Syntheseschritte sind in das IFS Konzept zu integrieren.

Eine Herausforderung stellt die Unterstützung verschiedener duplexfähiger Tasks dar.

Dazu muss der IFB als Multi-Task Interface mehrere heterogene Tasks unterstützen, die lesend und schreibend mit dem IFB kommunizieren können. Um diese Aufgabe effizient zu ermöglichen, ist die Makro-Struktur von einer generischen auf eine dedizierte Verbindungsstruktur umzustellen.

Die durch einen IFB verbundenen Tasks müssen sich dessen Ressourcen teilen. Um mögliche Konflikte zu beheben, soll ein echtzeitfähiger Scheduler als zentrale Steuerung in der Control Unit entwickelt werden. Die Integration des Schedulers erfordert eine systematische Erweiterung der Makro-Struktur.

Durch den Einsatz von Schedulern wird eine Ausführbarkeits- und Schedulinganalyse erforderlich. Die Analysen sollen statisch bereits zur Entwurfszeit feststellen, ob ein IFB den Echtzeitanforderungen gerecht wird, ohne dass das System synthetisiert werden muss.

Neben der Echtzeitfähigkeit soll die Control Unit die gepipelinte Ausführung von Kommunikationszyklen nach dem *Eingabe-Verarbeitung-Ausgabe* Prinzip ermöglichen. Das Design der Control Unit soll die bestehenden Konzepte und Ideen zur Rekonfiguration des IFBs zur Laufzeit unterstützen.

1.3 Aufbau der Arbeit

Die Studienarbeit stellt zunächst im Kapitel zwei vorhandene Technologien vor, auf die in späteren Kapiteln eingegangen wird. Dazu gehören insbesondere die Scheduling Verfahren.

Das dritte Kapitel stellt den IFB detailliert vor. Es beschreibt den modularen Aufbau des IFBs sowie das Zusammenspiel der einzelnen Module. Kapitel vier knüpft daran an und zeigt die, für diese Studienarbeitarbeit, notwendigen Erweiterungen auf. Des Weiteren werden die Erweiterungen der Architektur und Control Unit um dedizierte Kommunikation und Echtzeitfähigkeit vorgestellt. Insbesondere wird auf den Aufbau und die Funktionalität der Control Unit eingegangen.

Die im Kapitel fünf vorgestellte Ausführbarkeits- und Schedulinganalyse kann mit Hilfe der Erweiterungen in der Control Unit die Laufzeit der verschiedenen Protokolle untersuchen und eine Aussage über die Schedulability treffen.

2 Technologien

Ein IFB[1] ermöglicht es Komponenten mit inkompatiblen Protokollen zu verbinden. Da mehr als zwei Protokolle mit einem Multi-Task IFB verbunden werden können, muss ein Scheduler eingesetzt werden um die Ressourcen des IFB zu verwalten.

Dieses Kapitel erklärt die für diese Arbeit relevanten Aspekte von Protokollen und beschreibt die Grundbegriffe des Schedulings. Vorgestellt wird auch ein Protokoll, das häufig bei Realzeitanwendungen verwendet wird. Das Kapitel endet mit der Vorstellung verschiedener Schedulingverfahren.

2.1 Protokolle

Protokolle definieren die Kommunikation zwischen verschiedenen Kommunikationsteilnehmern. Dabei kann die Kommunikation direkt zwischen zwei Kommunikationspartnern ablaufen, oder über einen Bus, an dem mehr als zwei Kommunikationspartner angeschlossen werden können.

Sobald ein Bus verwendet wird, muss der Zugriff auf diesen geregelt werden. Das kann statisch oder dynamisch geschehen [11], [10].

- statische Zugriffsverfahren
  Der Zugriff auf den Bus wird mit statischen Zugriffsverfahren geregelt. Jedem Kommunikationsteilnehmer ist bekannt, wann er den Bus verwenden darf, und wann der Bus von anderen Kommunikationsteilnehmern verwendet wird. Ein Beispiel ist das Time Triggered Protocol, was später in diesem Kapitel vorgestellt wird.

- dynamische Zugriffsverfahren
  Bei dynamischen Zugriffsverfahren, wird der Bus benutzt, wenn er gebraucht wird. Man unterscheidet zwei Zugriffsstrategien anhand von Kollisionsvermeidungsstrategien. Systeme ohne Kollisionsvermeidung auf Protokollebene müssen Kollisionen im Anwendungsprogramm erkennen. Ein Beispiel dafür ist das ALOHA Protokoll. Systeme mit Kollisionserkennung verwenden den Bus nur schreibend, wenn Sie berechtigt sind. Diese Berechtigung, kann durch Prioritäten- oder Token-Verfahren vergeben werden. Beispiele sind Tokenring und FDDI Netzwerke.

[1] Das Modell des IFBs wird im Kapitel 3 vorgestellt

Alle mit einem Bus verbundenen Kommunikationspartner müssen das gleiche Protokoll verwenden.

Die Kommunikation zwischen Sender und Empfänger kann man drei verschiedene Protokolltypen anhand der durchgeführten Handshakes unterscheiden.

- fully interlocked

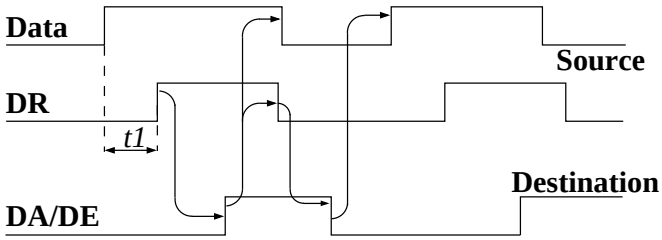

Abbildung 2.1: *Ablauf eines fully interlocked Protokolls*

Ein fully interlocked Protokoll zeichnet sich dadurch aus, dass Sender und Empfänger den Datenaustausch allein über Handshakes regeln. Die Abbildung 2.1 zeigt eine fully interlocked Kommunikation mit Sender (Source) und Empfänger (Destination).

Sobald der Sender Daten auf den Bus (`Data`) gelegt hat signalisiert er dies dem Empfänger indem er das Signal Data Ready auf 1 setzt (`DR<=1`). Der Empfänger registriert diese Signaländerung, liest die Daten und bestätigt das erfolgreiche Lesen des Signals indem er das Signal Data Acknowledge setzt (`DA<=1`). Der Sender nimmt die Daten vom Bus und setzt das Signal Data Ready auf 0 zurück, woraufhin der Empfänger Data Acknowledge zurücknimmt. Nun kann der Sender beginnen neue Daten zu übertragen.

- half interlocked

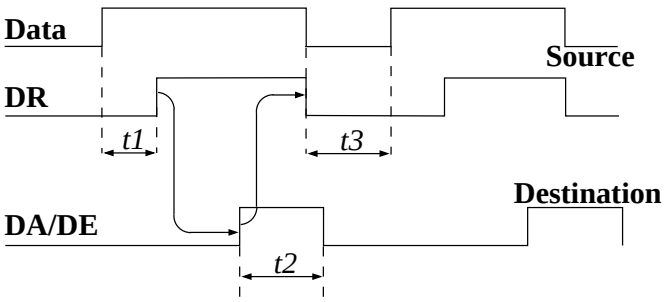

Abbildung 2.2: *Ablauf eines half interlocked Protokolls*

Wird ein half interlocked Protokoll verwendet, so wird ein Teil der Ereignisse per Laufzeitannahmen ausgelößt. In Abbildung 2.2 arbeitet der Empfänger mit Laufzeitannahmen, der Sender hingegen arbeitet interlocked. Der Sender nimmt also erst das Signal `DR` zurück, wenn der Empfänger das Signal `DA` gesetzt hat. Der Empfänger setzt aber voraus, dass der Sender innerhalb der Zeit t_2 das Signal DA gelesen hat.

- non interlocked

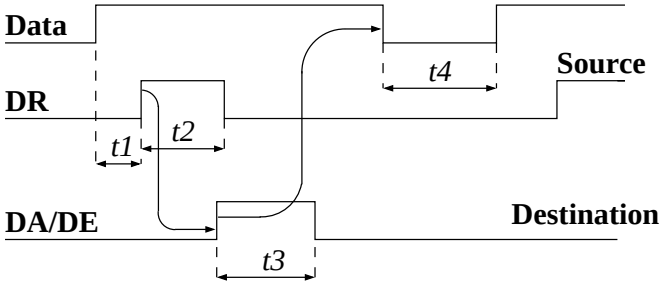

Abbildung 2.3: *Ablauf eines non interlocked Protokolls*

Wie in Abbildung 2.3 zu sehen ist werden bei einem non interlocked Protokoll alle Ereignisse durch Laufzeitannahmen ausgelößt. Die Kommunikationsteilnehmer müssen davon ausgehen, dass der jeweilige Kommunikationspartner gemäß der Zeitannahmen korrekt arbeitet.

2.1.1 Time Triggered Protocol (TTP)

Das Time Triggered Protokoll [9] wurde entwickelt, um den Bedürfnissen von verteilten Echtzeitsystemen gerecht zu werden. Da der IFB, in dieser Arbeit, um die Echtzeitfähigkeit erweitert wird, soll das TTP kurz vorgestellt werden.

Der Zugriff auf das Medium erfolgt via TDMA (Time Division Multiple Access). Das bedeutet, dass der Zugriff auf den Bus in Zeitschlitze (Slots) für Nachrichten eingeteilt wird. Eine Zuordnung zwischen Nachrichten und Slots wird fest definiert. Diese Zuordnung speichert man in einer MEDL (Message Descriptor List). Jeder am TTP/C Bus angeschlossende Kommunikationscontroller besitzt eine MEDL, damit die Lese- und Schreibzugriffe zum richtigen Zeitpunkt durchgeführt werden können. Ein TTP/C Controller wird in Abbildung 2.4 dargestellt.

Die Architektur eines TTA (Time Triggered Architecture) Knotens trennt das Kommunikationssystem und das Anwendungsprogramm vollständig voneinander. Eventuell vorhandene Kontrollsignale können das Interface nicht passieren. Die Kontrolle des Kommunikationssystems unterliegt dabei vollständig dem Kommunikationscontroller. Entsprechend der MEDL sendet das Kommunikationssystem die Daten “time triggered”.

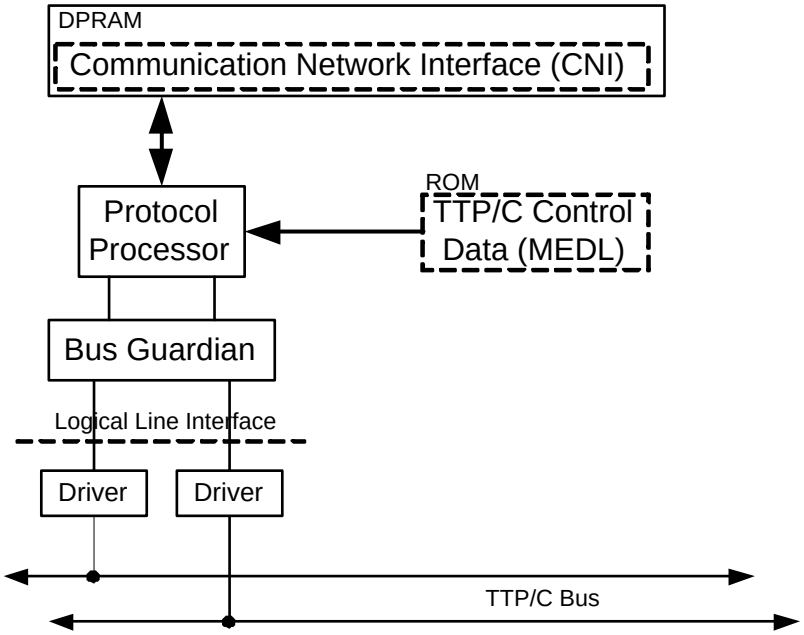

Abbildung 2.4: *Knoten der Time Triggered Architecture*

Das Anwendungsprogramm besitzt selbst keine „send“ Routine. Zeitliche Fehler des Anwendungsprogramms können das Interface nicht überwinden und andere Komponenten in Mitleidenschaft ziehen.

Der Protokollprozessor kann mit Hilfe der MEDL-Daten, die Lese- und Schreibzugriffe auf den TTP/C Bus und dem Kommunikationsinterface koordinieren. Das Anwendungsprogramm arbeitet auf dem Kommunikationsinterface und hat keinen Kontakt zum TTP/C Bus.

2.2 Scheduling

Wenn Aufgaben, die nicht gleichzeitig ausgeführt werden können, in eine zeitliche Abfolge gebracht werden müssen, werden Scheduler verwendet. Zunächst werden die Grundbegriffe des Schedulings vorgestellt, im Anschluss werden die verschiedene Schedulingverfahren klassifiziert und vorgestellt.

2.2.1 Grundbegriffe

Job

Ein Job ist die Liste von Befehlen, die unterbrechungsfrei durchgeführt werden müssen. Eine Menge von Jobs muss durch den Scheduler in eine Reihenfolge gebracht werden.

Dabei müssen die unterschiedlichen Eigenschaften eines Jobs berücksichtigt werden. So sollte zum Beispiel jeder Job vor seiner Deadline ausgeführt worden sein. Die Abbil-

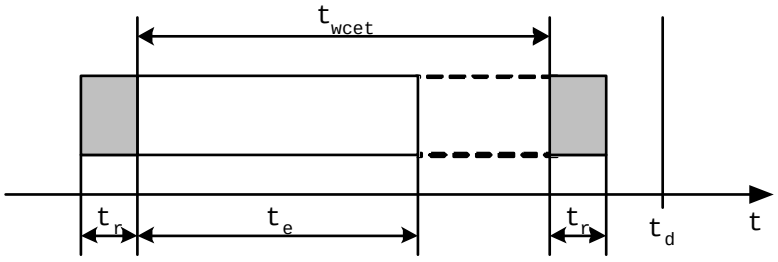

Abbildung 2.5: *Zeitlicher Ablauf eines Jobs*

Ausführungszeit	execution time	t_{ex}
maximale Ausführungszeit	worst case execution time	t_{wcet}
Freigabezeit, Bereitzeit	release time	t_r
Zeitschranke	deadline	t_D

Tabelle 2.1: *Zeitliche Charakteristiken eines Jobs*

dung 2.5 veranschaulicht die zeitlichen Eigenschaften eines Jobs. Die x-Achse bildet die Zeitachse. Alle Zeiten sind in Tabelle 2.1 zusammengefasst.

Task

Eine Task besteht aus $n, n \in \mathbb{N} \setminus 0$ Jobs, die zusammengehören. Bespielsweise kann eine Task aus 3 Jobs bestehen, wobei der erste Daten liest, der Zweite diese verarbeitet und der dritte Job die Daten in einen Speicher zurückschreibt.

Deadline

Zu jedem Job kann eine Zeit definiert werde, die angibt bis wann ein Job abgeschlossen werden muss. Diesen Zeitpunkt nennt man Deadline t_D. Wenn ein Job nicht innerhalb seiner Deadline abgeschlossen wurde, so werden die verarbeiteten Daten nutzlos. Bei einigen Systemen ist es zwingend Erforderlich, dass alle Jobs innerhalb ihrer Deadline abgeschlossen werden. Man spricht dann auch von „Harter Realzeit“ .

Ressource

Ressourcen sind alle Bestandteile eines Systems, die durch Jobs angefordert werden können. Die unterschiedlichen Ressourcen werden von einem Job belegt, und nach Beendigung des Jobs wieder freigegeben. Dazu benötigt ein Job die Freigabezeit (release time) t_r. Eine Ressource steht einem Job immer exklusiv zur Verfügung.

Schedule

Ein Schedule ordnet eine Menge von Jobs den verfügbaren Prozessoren in Form eines Ablaufplans zu. Wenn die Zuordnung keine der gegebenen Bedingungen verletzt, so ist der Schedule gültig (valid). Werden alle Zeitschranken der Jobs eingehalten, so ist der Schedule ausführbar (feasible).

Schedulability Test

Ein Schedulability Test prüft, ob es zu einem Scheduling Verfahren und einer Menge von Tasks einen gültigen und ausführbaren Schedule gibt. Dazu wird die Auslastung der verfügbaren Prozessoren bestimmt. Die **Utilisation** μ kann mit folgender Gleichung errechnet werden:

$$\mu = \sum_i \mu_i = \sum_i \frac{t_{ex_i}}{t_{P_i}} \leq n$$

Die für eine Task T_i benötigte Rechenzeit μ_i wird aus der Taskperiode t_{P_i} und der Taskausführungszeit t_{ex_i} berechnet. Der Wert n gibt die Anzahl der verfügbaren Prozessoren an und damit die maximal verfügbare Rechenzeit. Unterschreitet μ einen bestimmten, vom Scheduling Verfahren abhängigen Wert, so existiert ein gültiger Schedule.

Admission

Die Admission, auch Acceptance Test genannt, prüft, ob eine neue Task dem Ablaufplan hinzugefügt werden kann. Dazu wird ein Schedulability Test durchgeführt. Schlägt der Acceptance Test fehl, so wird die neue Task nicht zugelassen.

Preemptives Scheduling

Das preemptive Scheduling kann eine Task mit Priorität n abbrechen, wenn eine Task mit höherer Priorität $> n$ aktiviert wird. Wurde die höher priorisierte Task abgeschlossen, so wird die Abarbeitung der unterbrochenen Tasks fortgesetzt. Dies setzt voraus, dass jede Task unterbrochen werden kann. Alle Tasks müssen daher entsprechende Schnittstellen zur Verfügung stellen, um eine Unterbrechung und eine anschließende Wiederaufnahme der Berechnung zu ermöglichen.

Non Präemptives Scheduling

Im Gegensatz zum Präemptiven Scheduling lassen sich hier laufende Tasks nicht unterbrechen. Sind viele kurze Tasks für das Scheduling vorgesehen ist diese Methode besser geeignet als präemptives Scheduling, da keine Zeit für die Kontextwechsel benötigt wird.

Dynamisches Scheduling

Das Scheduling wird stets zur Laufzeit berechnet. Die Berechnung benötigt zusätzliche Ressourcen und stellt häufig ein $\mathbb{NP}$ Vollständiges Proplem dar. Man benötigt daher Verfahren, die in kurzer Zeit zu einem akzeptablen Ergebnis gelangen.

Statisches Scheduling

Beim statischen Scheduling wird der Ablaufplan vollständig zur Entwurfzeit berechnet. Zu diesem Zeitpunkt müssen daher alle Laufzeiten und Taskperioden bekannt sein. Die Schedulinginformationen werden dann in eine so genannte Dispatchingtabelle geschrieben, die zur Laufzeit ausgewertet wird.

Scheduling von Realzeitprogrammen

In Realzeitanwendungen müssen alle Zeitbeschränkungen eingehalten werden, da das System sonst nicht mehr zuverlässig arbeitet. Realzeitanwendungen sind daher besonders schwer zu schedulen. Man unterscheidet zwei Arten von Realzeitanwendungen:

- Harte Realzeit
  Harte Realzeit (engl. hard real-time) wird immer dann benötigt, wenn Anwendungen aufgrund verspäteter Nachrichten lebensnotwendige Systeme mit falschen Informationen ansteuern würden. Alle Nachrichten müssen innerhalb ihrer Deadline beim Empfänger angekommen sein. Herrschen in einem Auto keine Realzeitbedingungen, so kann beispielsweise das ABS nicht richtig angesteuert werden, wenn der Fahrer den elektrischen Außenspiegel einstellt. Die Datenpakete des Außenspiegels könnten die Datenpakete für das ABS verzögern, womit das ABS versagen würde.
- Weiche Realzeit
  Wenn die Folgen einer Verletzung der Realzeitbedingung die Stabilität des Gesamtsystems nicht beinträchtigen, so spricht man von weicher Realzeit (engl. soft real-time). Bei einem System zur Videowiedergabe, könnten sich zum Beispiel Videoinformationen verspäten. Die Wiedergabe ist dann zwar nicht mehr flüssig, das Gesamtsystem funktioniert aber weiter.

2.3 Scheduling Verfahren

2.3.1 Klassifikation

Scheduler für Realzeitanwendungen werden in verschiedene Klassen eingeteilt. Eine Übersicht ist in Abbildung (2.6) zu sehen. Zunächst unterscheidet man Scheduler für harte

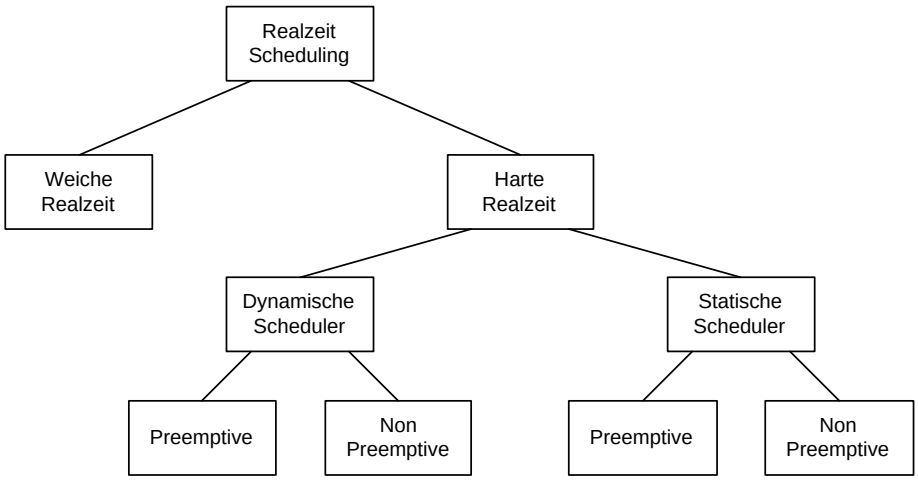

Abbildung 2.6: *Klassifikation von Schedulern*

und weiche Realzeit. Innerhalb der Klasse von Schedulern für harte Realzeit, werden statische und dynamische Verfahren unterschieden. Die Scheduling Verfahren teilen sich innerhalb dieser Klassen auf Verfahren für unterbrechbare (preemptive) und nicht unterbrechbare (non preemptive) Tasks auf. Das „zyklische Scheduling“ ist beispielsweise ein statischer Scheduler für Harte Realzeit mit nicht unterbrechbaren Tasks.

2.3.2 Zyklisches Scheduling

Eine Menge von Tasks sollen Daten empfangen und versenden. Folgende Voraussetzungen müssen erfüllt sein, damit das zyklische Scheduling angewendet werden kann:

- Die Anzahl der Tasks ist fest.
  Zur Laufzeit steht die Anzahl der Tasks fest. Es entstehen keine neuen Tasks und es werden keine Tasks entfernt.

- Jede Task ist periodisch und besitzt eine fest definierte Periode t_P.

- Die Tasks sind paarweise unabhängig.
  Es darf nicht vorkommen, dass Task A nur ausgeführt werden kann, wenn zuvor Task B ausgeführt wurde.

- Für die Umschaltung der Tasks wird keine Zeit benötigt.
  Beim Kontextwechsel vergeht keine Zeit. Sollte dennoch Zeit benötigt werden, so wird diese der entsprechenden Taskrechenzeit angerechnet.

- Jede Task hat eine feste worst-case Rechenzeit.

Dieses Modell ist sehr eingeschränkt. Das Scheduling lässt sich aber vollständig **vor** der Laufzeit berechnen, da von jedem der n Tasks die maximale Ausführungszeit und die Periode bekannt ist.

2.3.3 Schedulingverfahren für periodische unterbrechbare Tasks

Periodisch auftretende Tasks die Unterbrechbar sind, können mit Prioritätenbasierten Schedulingverfahren geschedult werden. Die Verfahren vergeben an jede Task eine Priorität. Die Task mit der höchsten Priorität wird ausgeführt. Die Prioritäten können statisch vergeben werden, oder zur Laufzeit bestimmt werden. Wird eine Task mit Priorität P ausgeführt, so kann sie durch eine Task mit höherer Priorität verdrängt werden. Beispiele für diese Verfahren sind:

- Rate Monotonic Priority assignment
  Die Priorität wird anhand der Periodenlänge vergeben. Die Task mit der kürzesten Periode erhält die höchste Priorität.
- Deadline Monotonic Priority Ordering
- Earliest Deadline First (EDF)
  Die Task, mit der kürzesten Zeit bis zu seiner nächsten Deadline, erhält die höchste Priorität.
- Shortest Job First

2.3.4 Schedulingverfahren für aperiodische Tasks

Treten Tasks nur aperiodisch auf, so müssen dynamische Scheduler eingesetzt werden. Diese Scheduler berechnen online einen möglichen Schedule. Dabei ist zu beachten, dass für die Berechnung eines Scheduls Zeit benötigt wird. Um diese Zeit zu minimieren werden oft heuristische Verfahren angewendet. In [1] werden unter anderem folgende Schedulingalgorithmen für aperiodische Tasks vorgestellt.

- Horn Algorithmus (EDF)
- Bratley Algorithmus
- Jackson Algorithmus (EDD)

2.4 Zielplattform

Der Interfaceblock IFB soll für die Verbindung verschiedener Tasks genutzt werden. Dazu muss der IFB als Hardware existieren. Zu diesem Zweck wird der IFB mit dem IFS-Editor in VHDL Code umgewandelt. Der entstandene Code kann dann mit geeigneten Softwarewerkzeugen in einen FPGA (Field programmable Gate Array) geladen werden.

Alle Teile des IFBs werden in einem FPGA parallel susgeführt. Für die sequenzielle Verarbeitung werden daher Automaten verwendet.

Ein FPGA ist eine synchrone Schaltung mit einer festen Taktfrequenz. Damit wird der synthetisierte IFB auch zu einer synchronen Schaltung. Die Taktfrequenz des FPGAs wird zum IFB Takt. Dieser Takt muss bei der Modifikation von zeitbehafteten Operationen berücksichtigt werden.

3 Das Modell des Interface Blocks

Ein Interface Block (IFB) ist ein System, welches uns ermöglicht Kommunikationskomponenten mit inkompatiblen Protokollen miteinander zu verbinden. Dabei kann der Interface Block das Protokoll eines Kommunikationspartners in das Protokoll des anderen Kommunikationspartners übersetzen. Aus Sicht der Kommunikationspartner ist der IFB als transparent anzusehen.

Das Blockschaltbild eines Interfaceblocks ist in Abbildung (3.1) zu sehen. Ein IFB ist modular aufgebaut und besteht im wesentlichen aus drei Grundbausteinen: dem Protocol Handler, dem Sequence Handler und der Control Unit. Zunächst werden die Funktionen des Sequence Handlers und Protocol Handlers erläutert. Im Anschluß werden die Aufgaben der Control Unit genauer spezifiziert.

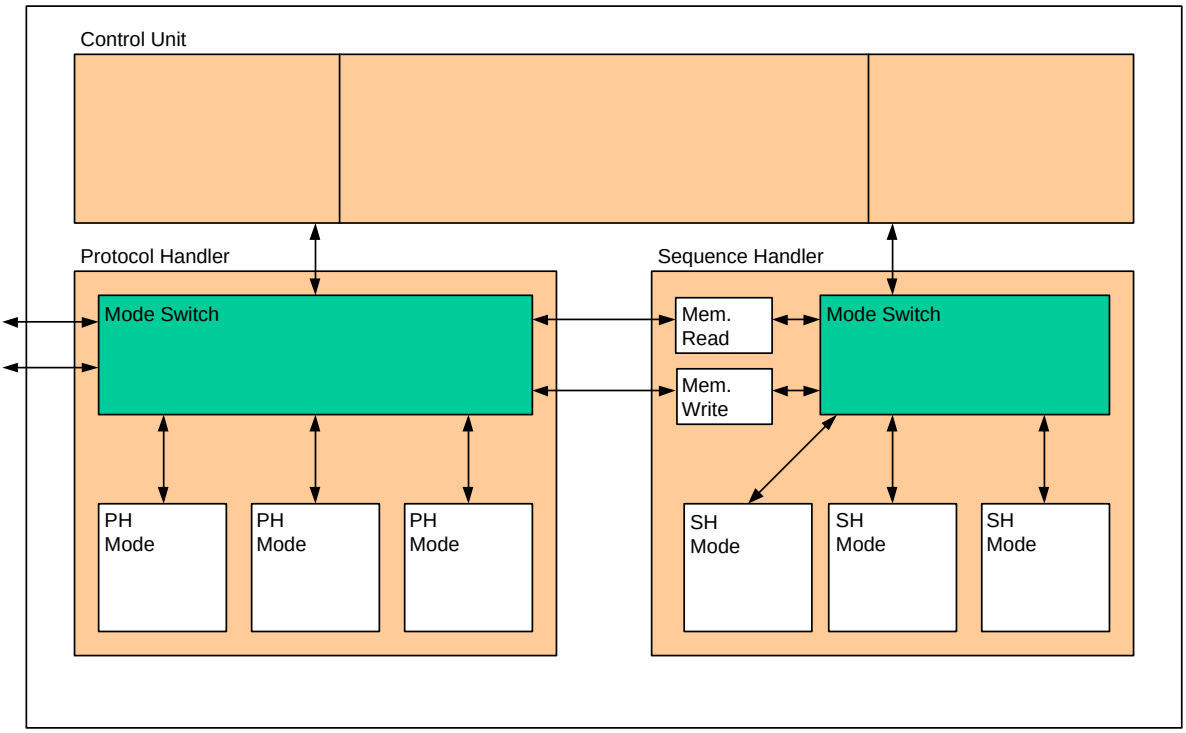

Abbildung 3.1: *Die Struktur des IFBs.*

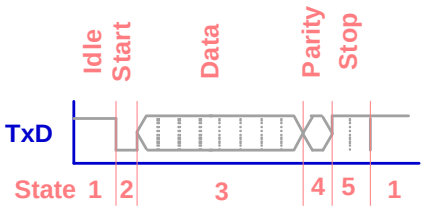

Abbildung 3.2: *Protokoll einer Task*

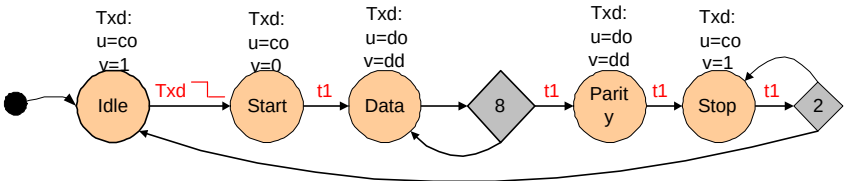

Abbildung 3.3: *Protokollautomat zu Protokoll in Abbildung (3.2)*

3.1 Die Makrostruktur

Die für den Datentransfer wichtigen Teile des IFBs sind der Protocol Hander und der Sequence Handler. Diese beiden Module sind wie der IFB modular aufgebaut. Aufbau und Kommunikation dieser Komponenten untereinander soll nun vorgestellt werden.

3.1.1 Der Protocol Handler (PH)

Der Protocol Handler verbindet den Interface Block mit allen angeschlossenen Kommunikationspartnern. Die Kommunikationspartner werden im Folgenden als Tasks bezeichnet. Kein anderer Teil des IFBs ist mit Tasks verbunden. Da jede Task ein anderes Protokoll zur Kommunikation voraussetzen kann, muss der Protocol Handler für jede Task ein geeignetes Pendant bereitstellen. Daraus folgt, dass für jede angeschlossene Task ein so genannter Protocol Handler Mode bereitgestellt werden muss.

Ein Protocol Handler Mode ist genau auf eine Task abgestimmt und simmuliert für diese einen realen Kommunikationspartner mit dem komplementären Protokoll. Um das komplementäre Protokoll zu implementieren, wird aus dem Kommunikationsprotokoll der Task der komplementäre Moore Automat abgeleitet. Der generierte Automat kommuniziert ständig mit der angeschlossenden Task. Sobald die Task Daten sendet, kann der Automat die Nutzdaten extrahieren und für die Weiterverarbeitung im IFB bereitstellen. Alle Modes im Protokollhandler arbeiten voll parallel.

Als Beispiel soll eine serielle Datenübermittlung dienen. In Abbildung (3.2) ist eine serielle Datenübermittlung auf der Leitung TxD zu sehen. Es gibt ein Startbit, 8 Datenbits, ein Paritybit und zwei Stoppbits. Der Protokollautomat, der für den Protocol Handler Mode abgeleitet wurde, ist in Abbildung (3.3) zu sehen. Da der Automat speziell

für dieses Protokoll geschaffen worden ist, ist auch bekannt, dass zunächst Steuerdaten **State 1** konsumiert werden müssen, bevor die Daten **State 3** und **State 4** empfangen werden können. Im Anschluss werden die Steuerdaten **State 5** empfangen. Der Automat im Protocol Handler Mode ist somit in der Lage zwischen Nutzdaten und Steuerdaten zu unterscheiden. Die Steuerdaten werden als redunanter Teil des Protokolls vom Protokollautomaten generiert bzw. konsumiert. Ausschließlich die Nutzdaten werden zur Weiterverarbeitung übertragen.

Ein weiterer Bestandteil des Protocol Handlers ist der Protocol Handler Switch. Der PH Switch verbindet jeden einzelnen Task mit dem dazugehörigen Protocol Handler Mode. Gleichzeitig kann der Switch einen Protocol Handler Mode mit dem Datenbus verbinden, damit Nutzdaten zur Weiterverarbeitung an den Sequence Handler gegeben werden können. Alle Verbindungen werden über den PH Switch geleitet. Der Einsatz des PH Switches ermöglicht auch die Rekonfiguration des Systems. Der Aspekt der Rekonfiguration wird im Kapitel 3.3.1 beschrieben.

Der Protocol Handler Switch besitzt 4 Interfaces. Das erste Interface verbindet den Switch mit allen am IFB angeschlossenden Tasks. Das zweite stellt eine Verbindung zwischen dem Switch und den Protocol Handler Modes her. Ein weiteres Interface verbindet die Control Unit mit dem Protocol Handler Switch und das vierte Interface bildet die Verbindung zum Sequence Handler.

Das Interface zwischen Task und Protocol Handler Switch enthält lediglich die Signale des Tasks. Zusätzliche Kontrollsignale sind nicht notwendig.

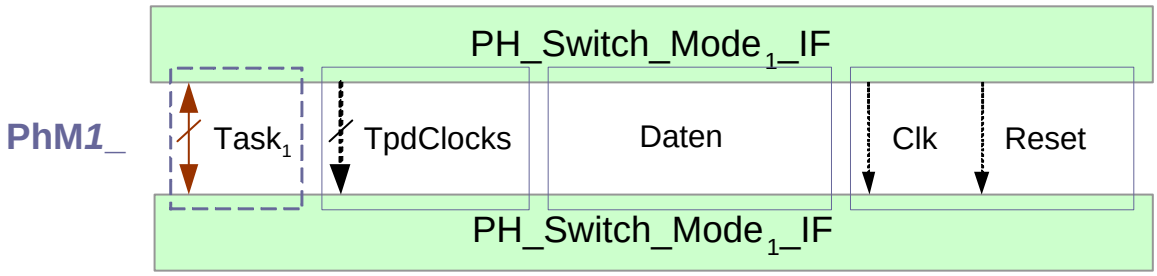

Abbildung 3.4: *Interface zwischen PH-Switch und einem PH-Mode.*

Ein Interface zwischen Protocol Handler Mode und dem Protocol Handler Switch setzt sich aus mehreren Teilen zusammen. Zunächst werden alle Signale der zugeordneten Tasks verbunden. Das Interface wird um Leitungen für die Datenübertragung erweitert. Sobald der PH Mode Daten vom Task empfängt, werden die Daten über diese Leitungen weitergegeben. Zusätzlich wird je eine Leitung für den Systemtakt und ein Resetsignal hinzugefügt. Siehe auch Abbildung (3.4).

Die Schnittstelle zur Control Unit, Abbildung (3.5), enthält ebenfalls Signale für Systemtakt und Reset. Zusätzlich existieren verschiedene Steuer- und Statusleitungen.

Das Interface, das den Protocol Handler mit dem Sequence Handler verbindet, enthält

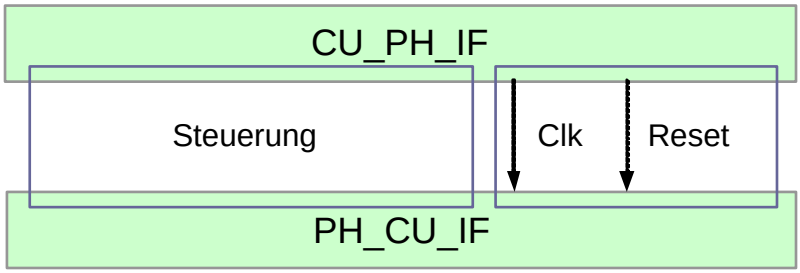

Abbildung 3.5: *Interface zwischen PH-Switch und der Control Unit.*

lediglich Datenleitungen, um die Nutzdaten zwischen den PH Modes und dem Sequence Handler zu transferieren.

3.1.2 Der Sequence Handler (SH)

Die von einem Protocol Handler Mode extrahierten Nutzdaten werden an den Sequence Handler zur Verarbeitung weitergegeben. Der Sequence Handler besitzt, wie der Protocol Handler, einen Switch, der alle SH internen Module verbindet. Analog zum Protocol Handler gibt es im Sequence Handler auch Sequence Handler Modes. Diese sind für die Verarbeitung der Nutzdaten verantwortlich. Außer dem Switch und den Sequence Handler Modes gibt es im Sequence Handler noch einen Speicher, in dem die vom Protocol Handler verarbeiteten Daten zwischengespeichert werden können.

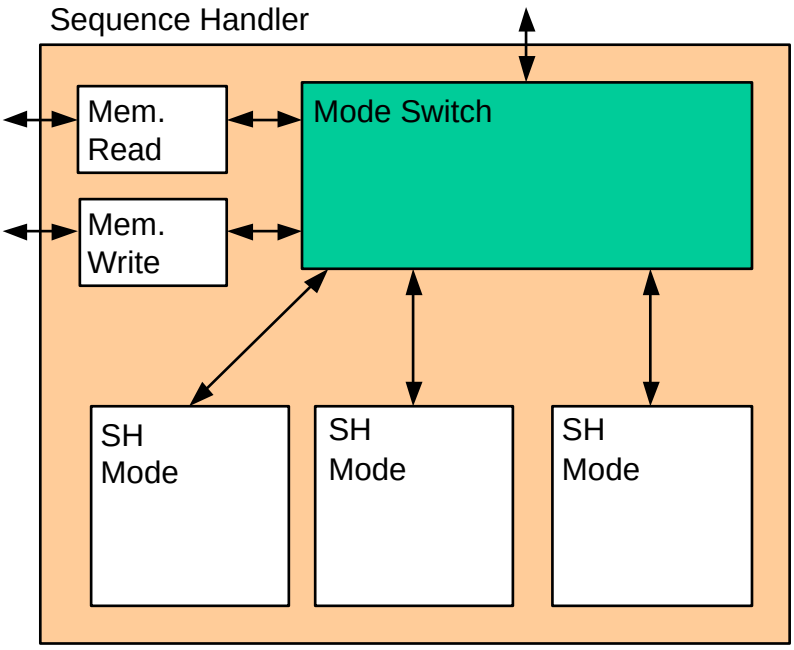

Abbildung 3.6: *Aufbau des Sequence Handlers.*

Ein Mode im Sequence Handler ist ein spezieller Automat der die Nutzdaten einer Task in Nutzdaten einer anderen Task umwandelt. Dabei sind verschiedene Modifikationen der Daten möglich.

Jeder Mode implementiert ein so genanntes IFD-Mapping. Das IFD Mapping ist eine Sprache mit der „Mapping Equations“ definiert werden können. Eine Gleichung kann Nutzdaten einer Task in Nutzdaten einer anderen Task umwandeln. Dazu stehen vier kombinierbare Grundoperation zu Verfügung.

- Zuweisung von konstanten Werten.
- Umsortieren von Bits.
- Ausgaben von boolschen Funktionen.
- Ausgaben von endlichen Zustandautomaten.

Ein IFD-Mapping kann mehrere Gleichungen enthalten. Weitere Informationen zu IFD-Mappings sind in [8] zu finden.

Sowie ein SH Mode aktiviert wird, werden die notwendigen Daten aus dem Speicher geladen und verarbeitet. Die verarbeiteten Daten werden wieder im Speicher abgelegt. Eine direkte Kommunikation zwischen Protocol Handler und Sequence Handler Mode findet zu keinem Zeitpunkt statt. Sobald die verarbeiteten Daten im Speicher abgelegt wurden, kann ein PH Mode diese anfordern und an einen Task senden.

Der Switch im Sequence Handler verbindet alle Module des Sequence Handlers untereinander und bildet die Schnittstelle zum Speicher und zur Control Unit. Das Interface zum Protocol Handler wird im Sequence Handler über den Speicher realisiert. Automaten nehmen die Daten vom Datenbus entgegen und legen diese im Speicher ab. Die Sequence Handler Modes können dann aus dem Speicher die notwendigen Daten für die IFD-Mappings beziehen und die berechneten Daten wieder im Speicher ablegen.

Die Abbildung (3.7) zeigt die Schnittstelle zur Control Unit. Neben den Steuerleitungen gibt es wieder eine Leitung für den Systemtakt und ein Resetsignal.

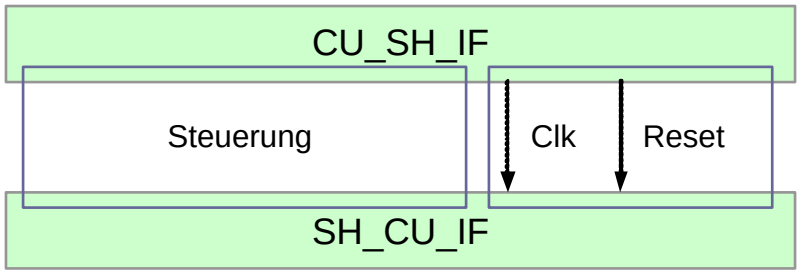

Abbildung 3.7: *Interface zwischen SH-Switch und der Control Unit.*

Die Schnittstelle zwischen Sequence Handler Switch und SH Mode enthält neben den Signalen Clock und Reset, Datenleitungen für die Kommunikation mit dem Speicher.

3.2 Datenfluss im Protocol und Sequence Handler

Der Datenfluss im Protocol und Sequence Handler wird in diesem Kapitel exemplarisch vorgestellt werden. Es existieren zwei Tasks, die beide mit dem Interface Block verbunden sind. Der Task A sendet Daten an den IFB. Task B wartet auf Daten, die der IFB an Task B versendet sobald er sie von Task A empfangen hat.

1. Task A versendet Daten. Der Protokollautomat des zugehörigen PH Modes empfängt die Daten.
2. Der PH Mode, an dem Task A angeschlossen ist, extrahiert die Nutzdaten und leitet diese an den Sequence Handler weiter.
3. Der Sequence Handler nimmt die Daten vom Protocol Handler entgegen und legt diese im Speicher ab.
4. Alle Daten wurden vom PH Mode des Tasks A angenommen. Der PH Mode wartet nun auf neue Aktivitäten des Tasks.
5. Der zugehörige SH Mode wandelt die empfangenden Daten um und schreibt die umgewandelten Daten wieder in den Speicher.
6. Task B ist in einem Zustand des Protokolls, in dem Daten empfangen werden können. Folglich befindet sich der zugehörige Protocol Handler Mode (PHM B) in Sendebereitschaft.
7. Der PH Mode des Tasks B holt die zuvor umgewandelten Daten aus dem Speicher des Sequence Handlers, fügt sie in das Protokoll von Task B ein und sendet sie somit.
8. Der Task B nimmt die Daten an.

Die Abbildung (3.8) zeigt die Abhängigkeiten der Datenübertragung von Task A zu Task B in einem Sequenzdiagramm. Die aktiven Phasen des Akteurs “PH Mode A” und “PH Mode B” zeigen die Gültigkeit der Nutzdaten im System an.

3.3 Steuerung des Interface Blocks

Damit im IFB alles funktioniert muss eine zentrale Einheit alle Zugriffe auf gemeinsame Ressourcen überwachen und steuern. Im IFB heißt diese zentrale Einheit Control

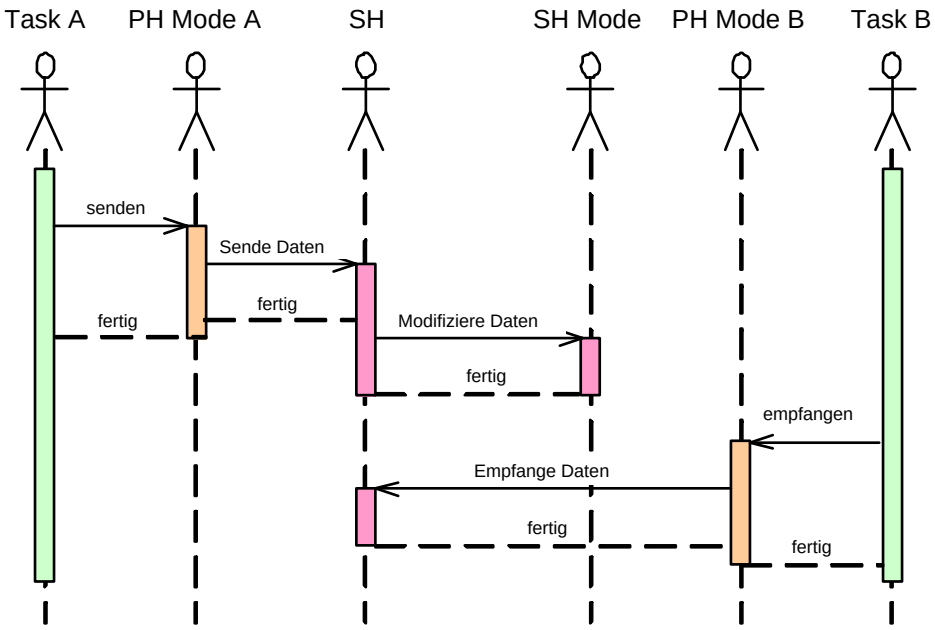

Abbildung 3.8: *Ablauf einer Kommunikation.*

Unit. Diese Control Unit muss den Speicherzugriff im Sequence Handler überwachen und die Datenbusse zwischen Protocol Handler und Sequence Handler verwalten. Um diese Aufgaben auszuführen müssen verschiedene Steuerleitungen von der Control Unit zum Sequence Handler und zum Protocol Handler erstellt werden.

Die Control Unit muss festlegen, welcher PH Mode über den Datenbus mit dem Sequence Handler kommunizieren darf und welcher Sequence Hander Mode Modifikationen an Daten vornehmen kann. Jeder Protocol Handler Mode muss daher an die Control Unit melden wann ein Buszugriff erforderlich ist. Des Weiteren muss die Control Unit jederzeit den aktuellen Zustand der Daten im Speicher des Sequence Handlers kennen und wissen, welcher SH Mode die Daten verarbeiten kann. Die durch die Transformation der Daten im Sequence Handler entstandenen Daten müssen ebenfalls durch die Control Unit verwaltet werden.

Dabei muss die Control Unit beachten, dass stets die Kausalität eingehalten wird. Eine Task kann erst Daten empfangen, wenn zuvor die sendende Task die notwendigen Daten übertragen hat.

Die beschriebenen Funktionen erfordern Erweiterungen des Interfaceblocks, die im Kapitel 4 vorgestellt werden.

3.3.1 Rekonfiguration

Nachdem ein IFB synthetisiert worden ist, kann der IFB z.B. auf einem FPGA geladen werden. Da FPGAs im laufenden Betrieb partiell neu konfiguriert werden können, bietet es sich an, den IFB so zu designen, dass Teile des IFBs zur Laufzeit ausgetauscht werden können.

Verschiedene Anwendungsszenarien sind für die Rekonfiguration denkbar. Zum einen kann ein IFB mit maximaler Ausprägung für einen FPGA generiert werden. Dieser auf dem FPGA geladene IFB unterstüzt dann n Protocol Handler Modes und damit n Tasks. Genutzt werden aber nur $n - m$. Soll nun ein weiterer Tasks angeschlossen werden, so muß nicht der ganze IFB neu generiert werden, was letztendlich Ausfallzeiten nach sich ziehen würde, man kann stattdessen mit Hilfe der Rekonfiguration in den freien FPGA Speicher einen neuen Mode laden. Ist der Ladevorgang abgeschlossen, so wird der PH Mode einfach hinzugeschaltet.

Ein anderer Fall, in dem die Rekonfiguration helfen kann, tritt auf, wenn eine Task durch eine neue Task ersetzt werden soll, alle anderen Tasks aber weiterarbeiten sollen. Alle Bereiche des FPGAs, die von der auszutauschenden Task genutzt werden, werden deaktiviert. Nachdem die neue Konfiguration geladen wurde, werden die betroffenen Bereiche wieder eingeschaltet.

Bei beiden Beispielen bleibt der IFB für die anderen Tasks voll funktionsfähig. Damit die Rekonfiguration funktioniert muss der Interfaceblock entsprechend um eine Steuerung für die Rekonfiguration erweitert werden. Alle PH Modes und SH Modes müssen durch entsprechende Logik komplett vom IFB getrennt werden können.

Das dynamische Hinzufügen von neuen Komponenten kann nur durchgeführt werden, wenn die Datenbusse im Protocol Handler für den Datentransfer groß genug sind. Es können nur PH Modes und SH Modes ausgetauscht werden. Werden zusätzliche Datenleitungen im Switch benötigt, so ist eine Rekonfiguration nicht möglich.

4 Echtzeiterweiterungen des IFB

In diesem Kapitel werden die, im Rahmen dieser Arbeit entstandenen, Erweiterungen des IFBs vorgestellt. Das sind insbesondere die Erweiterungen der Control Unit, die Echtzeitanwendungen ermöglichen sollen. Dabei werden Scheduler verwendet, die den Zugriff auf die Ressourcen regeln.

4.1 Architekurerweiterungen

Damit in der Control Unit alle notwendigen Informationen für einen Scheduler bereitgestellt werden können, müssen einige Module des IFBs erweitert werden.

4.1.1 Frames

In einem Protokoll eines Tasks können Verzweigungen auftreten. Man Unterteilt Protokolle daher in Grundblöcke. Dabei betrachtet man das Protokoll als Graph, die Zustände des Protokolls werden zu Knoten und die Transitionen zu Kanten. Ein Grundblock ist dann eine maximale geordnete Menge von Knoten auf einem Kreisfreien Graphen. Nur der erste und letzte Zustand eines Grundblocks darf einen Grad größer 1 haben. Nähere Informationen zu Grundblöcken in Protokollen sind in [12] angegeben.

Der IFB verbindet Tasks mit unterschiedlichen Protokollen miteinander. Diese Protokolle enthalten verschiedene Grundblöcken mit Zuständen, die Daten übertragen. Damit der Scheduler nicht über jedes einzelne Datenbit informiert werden muss, werden die Bits zu Datenpaketen und zusammenhängende Datenpakete zu Frames zusammengefasst. Die Control Unit muss somit nur über ankommende Frames informiert werden.

Wie Abbildung (4.1) zeigt, werden aneinandergrenzende Nutzdaten zu Datenpaketen zusammengefasst. Die Datenpakete werden aus den Nutzdaten, die mit **D** markiert sind, immer in Rechtecken zusammengefasst und enthalten keine Kontrolldaten **C**. Serielle Nutzdaten werden gegenüber parallelen Daten beim Zusammenfassen bevorzugt, daher entstehen in Abbildung (4.1) drei Datenpakete.

Ein Frame enthält ein oder mehrere überlappende Datenpakete. Die Datenpakete **1** und **2** aus Abbildung (4.1) bilden den Frame **A**, das dritte Datenpaket hat keine Überschneidungen mit anderen Datenpaketen und bildet daher einen eigenen Frame **B**.

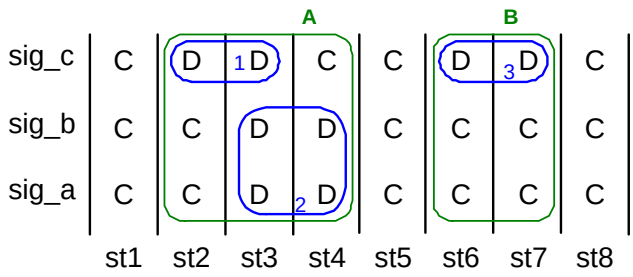

Abbildung 4.1: *Ein Protokoll mit seinen Kontroll- und Nutzdaten. (Teil 1)*

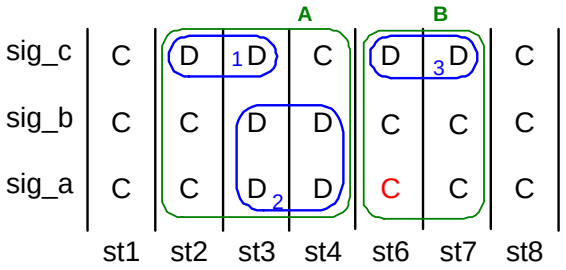

Abbildung 4.2: *Ein Protokoll mit seinen Kontroll- und Nutzdaten. (Teil 2)*

Der Zustand **st5** trennt die Frames voneinander. Wird dieser entfernt, so entsteht das Protokoll in Abbildung (4.2). Es haben sich keine neuen Überlappungen gebildet. Die Aufteilung der Frames ändert sich daher nicht. Sobald das rot markierte Bit in ein Datenbit umgewandelt wird, entsteht das Protokoll in Abbildung (4.3). Aufgrund der Überlappungen werden alle Datenpakete dem Frame **A** zugeordnet.

Jedes Datenpaket bekommt im Speicher des Sequence Handlers einen exklusiven Speicherplatz. Die aus den Datenpaketen gebildeten Frames besitzen eine systemweit eindeutige ID. Die letzte Stelle dieser FrameID gibt an, ob der Frame Daten von einer Task empfängt oder Daten an eine Task sendet. In Tabelle (4.1) sind Beispiele für FrameIDs

FrameID dezimal	FrameID binär	Frame lesen	Frame schreiben
5	0010**1**	×	
8	0100**0**		×

Tabelle 4.1: *Beispiele für FrameIDs.*

angegeben. Die Spalten „lesen“ und „schreiben“ zeigen an, ob der Frame Daten von einer Task empfängt (lesen) oder Daten an einen Frame sendet (schreiben).

Damit die FrameIDs für den IFB nutzbar werden, muss im PH Mode der Protokoll Automat modifiziert werden. Vor und hinter den Zuständen des Protokolls, die einen Frame bilden, werden neue Kontrollzustände eingefügt. Die beiden Zustände vor den Daten leiten mit Hilfe der Control Unit eine Datenübertragung ein (**establisch**). Der Zustand nach den Datenzuständen beendet die Datenübertragung (**release**). Diese Erweiterung

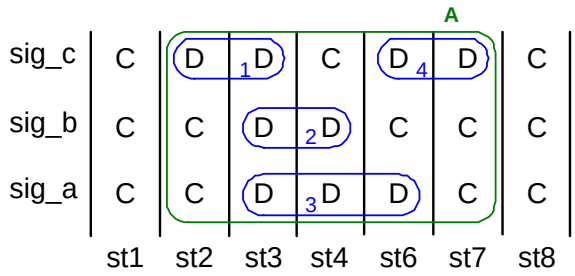

Abbildung 4.3: *Ein Protokoll mit seinen Kontroll- und Nutzdaten. (Teil 3)*

des Protokolls erfolgt, ohne dass für die angeschlossene Task ein Veränderung im Ablauf sichtbar ist. In Abbildung (4.4) wird ein Grundblock eines Protokolls dargestellt. Die

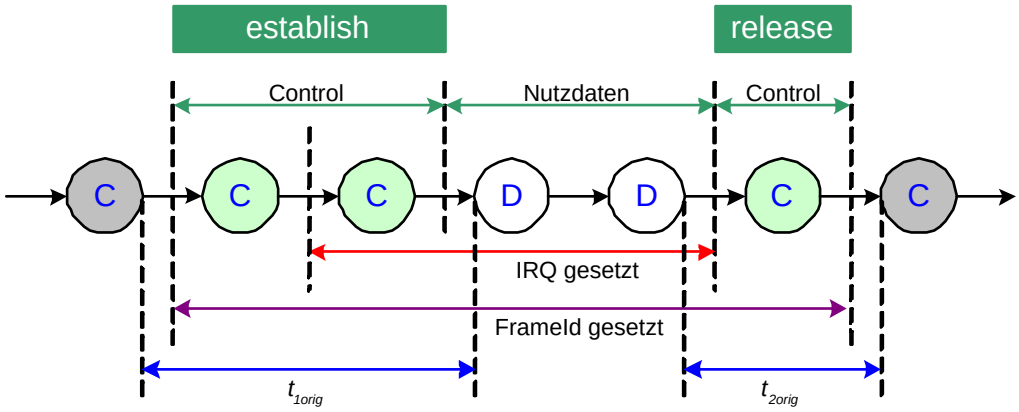

Abbildung 4.4: *Ein um Kontrollzustände erweiterter Frame.*

Kontrollzustände des Protokolls sind grau. Die Datenzustände **D** des Frames werden von neuen Kontrollzuständen (grün) umschlossen. Werden der abgebildete Grundblock in einem Protokoll und der erste grüne Kontrollzustand erreicht, so wird die FrameID auf den Bus gelegt.

Das Protokoll wechselt mit der nächsten steigenden Taktflanke und ohne Bedingung in den nächsten Zustand, der zusätzlich zur FrameID den IRQ des Protocol Handler Modes setzt. An dieser Stelle muss die Control Unit den Datenbus für den PH Mode freigeben damit das Protokoll in den nächsten Zustand wechseln kann.

Für die Control Unit markieren die Frames Bereiche, die unterbrechungsfrei eingelesen werden müssen. Wird eine FrameID und ein IRQ durch einen Protocol Handler gesetzt, so kann die Control Unit prüfen ob die entsprechenden Datenpakete verfügbar sind. Ist das der Fall, so kann der Scheduler dem Protocol Handler Mode Zugriff auf dem Bus gestatten.

Um den entstandenen zeitlichen Overhead der eingefügten Zustandsübergänge zu kompensieren, kann in zeitbehafteten Protokollen eine Zeitkorrektur durch Modifikation der

betroffenen Transitionen innerhalb des Frames erfolgen. Dazu muss der IFB schneller getaktet sein als die Task.

$$t_{\text{neu}} = t_{1orig} - n \cdot Clk \tag{4.1}$$

$$n = \text{Anzahl der zusätzlichen Zustandsübergänge} \tag{4.2}$$

Die berechnete Zeit t_{neu} muss auf die neue entstandenen Transitionen aufgeteilt werden. Wird die Zeit t_{neu} negativ, ist das Protokoll nicht ausführbar. Damit die Daten an den Sequence Handler weitergegeben werden können, müssen zwischen Datenzustände ebenfalls zusätzlische Kontrollzustände integriert werden. Die Zeitkorrektur muss hier ebenfalls angewendet werden.

Ein Protokoll mit drei Grundblöcken kann dann wie in Abbildung (4.5) aussehen. Die Entscheidung, ob der obere oder untere Grundblock ausgeführt wird, ist nur von den Transitionen der Task abhängig. Für jeden Grundblock kann eine Periode t_P und eine Deadline t_D definiert werden.

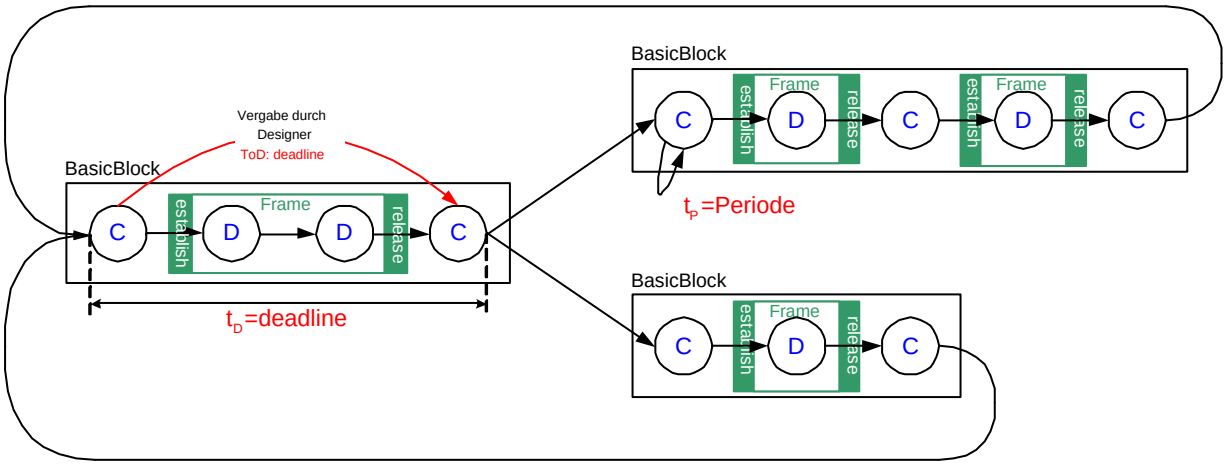

Abbildung 4.5: *Protokoll mit drei Grundblöcken.*

4.1.2 Erweiterungen für die Kommunikation zwischen Protocol Handler und Sequence Handler

Durch die Einführung der FrameIDs muss die Kommunikation zwischen Protocol Handler und Sequence Handler erweitert werden. Die erweiterte Schnittstelle ist in Abbildung (4.6) zu sehen. Um mit dem Sequence Handler zu kommunizieren, muss der Protocol Handler die Daten auf den Bus Data_X und die FrameID auf den Bus FrameID_X legen. Der Sequence Handler kann nun die FrameID auslesen und abhängig von dieser die Datenpakete an die entsprechenden Stellen im Speicher schreiben.

Der Zugriff auf den Bus wird durch die Control Unit gesteuert. Für die Kommunikation stehen zwei unidirektionale Datenbusse zur Verfügung. Somit können Lese- und Schreib-

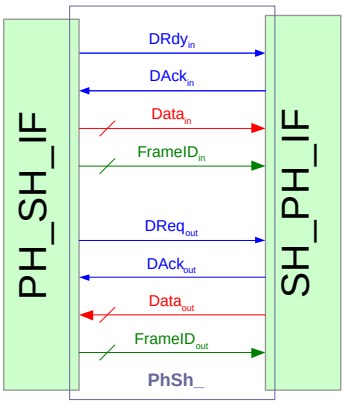

Abbildung 4.6: *Schnittstelle, die den Protokollhandler und Sequenzhandler verbindet.*

zugriffe immer parallel Ausgeführt werden. Um Sicherzustellen, dass keine Daten verloren gehen, wird für die Kommunikation ein Fully Interlocked Protokoll verwendet. Dazu werden für jeden Datenzustand innerhalb eines Frames zusätzliche Kontrollzustände eingefügt, die den Fully Interlocked Handshake mit dem Sequence Handler implementieren (Abbildung (2.1)). Die Zeitkorrektur der Transitionen kann hier ebenso angewendet werden.

Bevor der Sequence Handler Daten annimmt prüft er die FrameID. Mit dieser ID können die Positionen der einzelnen Datenpakete im Speicher bestimmt werden. Sobald alle Daten vom Sequence Handler empfangen wurden, wird der Bus wieder freigegeben.

Für beide Kommunikationsrichtungen gilt, dass der Protocol Handler eine FrameID an den Sequence Handler senden muss. Sollen Daten an den Sequence Handler gesendet werden, so kann der Sequence Handler entscheiden an welche Speicherstellen die Daten geschrieben werden. Bei Lesevorgängen kann der Sequence Handler die zugehörigen Daten, entsprechend der FrameID, aus dem Speicher an den Protocol Handler senden.

4.2 Die Control Unit

Um den umfangreichen Anforderungen an die Control Unit gerecht zu werden, wurde diese in vier Teilkomponentenen unterteilt.

- Scoreboard
  Sorgt für eine gepipelinete Ausführung nach dem EVA Prizip unter berücksichtigung der Kausalität. Dazu steuert das Scoreboard den Sequence Handler und verwaltet Informationen über Frames.
- Scheduler Ctrl_{in}
  Scheduler für eingehende Daten, übernimmet die Arbitrierung der Busse Data_{in} und $\text{FrameID}_{\text{in}}$

- Scheduler Ctrl_{out}
  Scheduler für ausgehende Daten, steuert die Arbitrierung der Busse Data_{out} und $\text{FrameID}_{\text{out}}$
- Rekonfiguration
  Verwaltet die einzelnen Teile des IFBs die durch die Rekonfiguration ausgetauscht werden können.

Die Scheduler arbitrieren die Zugriffe der Protocol Handler auf die Busse zum Sequence Handler. Dazu beziehen sie Informationen über die Frames aus dem Scoreboard.

4.2.1 Scoreboard

Das Scoreboard in der Control Unit ist für die Steuerung des Sequence Handlers zuständig. Gleichzeitig können die Scheduler mit Hilfe des Scoreboard prüfen, ob ein Frame gelesen oder geschrieben werden darf. Datenpakete, die als Eingabe für einen Sequence Handler Mode dienen sind Eingangsdatenpakete. Die Datenpakete, die durch einen SH Mode generiert werden sind Ausgabedatenpakete. Das Scoreboard speichert zu jedem Datenpaket die Information, ob es frei oder belegt ist.

Die Verarbeitung der Daten wird vom Scoreboard gesteuert und läuft nach dem EVA Prinzip ab:

Eingabe Lesen der eingehenden Nutzdaten
Zunächst prüft der Scheduler, ob ein gegebener Frame gelesen werden darf. Das Scoreboard ermittelt das, indem geprüft wird, ob alle Datenpakete des zu lesenden Frames im Speicher verarbeitet wurden und damit als frei markiert sind. Dem Scheduler wird signalisiert, dass der Frame gelesen werden darf. Daraufhin erteilt der Scheduler dem Protocol Handler Mode den Zugriff auf den Datenbus. Dieser überträgt seine Daten an den Sequence Handler. War die Übertragung der Daten erfolgreich, so signalisiert der Scheduler dem Scorboard, dass der Frame gelesen wurde. Damit markiert das Scoreboard die Datenpakete des Frames als belegt. Andernfalls würden die Daten verworfen.
Folgende Annahmen werden durch das Scorboard sichergestellt:

- Ein Frame enthält n Datenpakete. Ein Eingangsdatenpaket wird maximal einmal von jedem Sequence Handler Mode verarbeitet.
- Ein Ausgangsdatenpaket kann nicht als Eingangsdatenpaket genutzt werden.
- Das Scoreboard gibt an, dass ein Frame gelesen werden kann, wenn alle Eingangsdatenpakete frei sind.
- Nachdem der Scheduler ein Frame gelesen hat, meldet er dies dem Scoreboard. Dieses setzt alle zugehörigen Datenpakete auf belegt.

Verarbeitung Die Modifikation der Nutzdaten.

Ein Sequence Handler Mode modifiziert die Nutzdaten der eingelesenden Datenpakete und bildet damit die neuen Datenpakete für die Ausgabe. Jeder Sequence Handler Mode ist die Implementierung eines IFD-Mappings. Das Scoreboard kennt zu jedem SH Mode alle Eingangs- und Ausgangsdatenpakete. Sobald alle Eingangsdatenpakete zu einem SH Mode verfügbar sind und alle Datenpakete die vom diesem SH Mode erzeugt werden frei sind, initiiert das Scoreboard die Verarbeitung.

Ein SH Mode, der seine Arbeit abgeschlossen hat, setzt das `ModeComplete`- Signal für das Scoreboard. Dieses markiert mit Hilfe dieser Information die erzeugten Ausgangsdatenpakete als belegt und die benutzten Eingangsdatenpakete als frei. Die folgende Punkte fassen die Steuerungsaufgaben zusammen:

- Sobald alle für einen Sequence Handler Mode notwendigen Datenpakete als gelesen markiert sind und alle Ausgabedatenpakete als frei markiert sind, startet das Scoreboard diesen Sequence Handler Mode.
- Hat ein Sequence Handler Mode seine Arbeit abgeschlossen, so werden die Eingangsdaten des Modes als frei und die Ausgangsdaten als belegt markiert.

Ausgabe Ausgabe der zu versenden Nutzdaten

Der Scheduler prüft mit Hilfe des Scoreboards, ob ein Frame versendet werden darf. Dazu prüft das Scoreboard ob alle Datenpakete des Frames als gültig markiert sind. Der Scheduler vergibt dann den Buszugriff, und meldet dem Scoreboard den erfolgreichen Versand, wenn der Bus wieder freigegeben wurde. Das Scoreboard markiert daraufhin die versendeten Datenpakete als frei. Die beiden Punkte geben die Aufgaben des Scorboards für die Datenrückgabe wieder:

- Ein Frame kann ausgegeben werden, wenn alle enthaltenden Ausgangsdatenpakete belegt sind.
- Wurde ein Frame ausgegeben, so werden die entsprechenden Ausgabedatenpakete als frei markiert.

Die Abbildung (4.7) zeigt die Verbindung des Scoreboards mit dem Protocol Handler und dem Scheduler Ctrl_{out}. Der Bus `FrameProcessed` meldet zu jedem Protocol Handler Mode, ob alle Datenpakete des aktuellen Frames als belegt markiert sind, und damit, ob der Frame ausgegeben werden kann. Der Scheduler sendet über den Bus `FrameSent` eine FrameID an das Scoreboard, um diesem zu signalisieren, dass der zugehörige Frame ausgegeben wurde.

Über die Busse `FrameReceived` und `FrameReady` kommuniziert der Scheduler Ctrl_{in} mit dem Scoreboard. Der Bus `FrameReceived` verhält sich wie der Bus `FrameSent` und der Bus `FrameReady` übermittelt, wie `FrameProcessed`, ob der aktuelle Frame als frei markiert ist und damit empfangen werden kann.

Damit das Scoreboard diese Informationen für die Scheduler bereitstellen kann, benötigt es zu jedem Protocol Handler Mode die aktuelle FrameID.

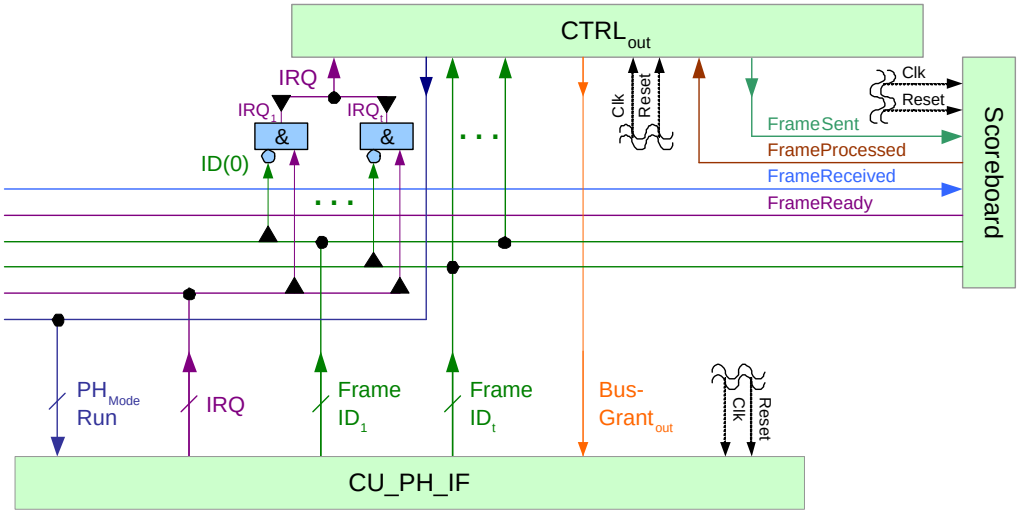

Abbildung 4.7: *Logische Schaltung um FrameIDs zu filtern.*

4.2.2 Die Scheduler Ctrl_{in} und Ctrl_{out}

Ein IFB besitzt keinen Prozessor. Die Scheduler des IFBs müssen also keine Rechenzeit auf verschiedene Prozesse aufteilen, stattdessen steuern die Scheduler den Buszugriff im System. Die Datenbusse können Daten nur unidirektional transportieren. Es gibt daher zwei Datenbusse: Der erste überträgt die Daten zum Sequence Handler während der zweite Datenbus die Daten vom Sequence Handler an den Protocol Handler sendet. Die beiden Datenbusse sind voneinander unabhängig, weshalb zwei unabhängige Scheduler eingesetzt werden. Der erste Scheduler Ctrl_{in} ist für die Datenübertragung vom Protocol Handler Mode zum Sequence Handler verantwortlich. Der zweite Scheduler Ctrl_{out} ist für die Übertragung der Daten vom Sequence Handler zurück zum Protocol Handler zuständig. Beide Scheduler kommunizieren mit dem Scoreboard um Informationen über die im System vorhandenen Frames zu erlangen.

Da es zwei Scheduler mit unterschiedlichen Aufgaben gibt müssen nun die Daten, die vom Protocol Handler an die Scheduler gegeben werden, gefiltert werden. Der Scheduler Ctrl_{in} soll nur über Frames informiert werden die Daten lesen, also Daten an den Sequence Handler weiterleiten. Dazu werden alle FrameIDs markiert. Wie in Tabelle (4.1) zu sehen ist, gehören alle ungeraden FrameIDs zu Frames, die gelesen werden.

Durch eine einfache logische Schaltung, die in Abbildung (4.7) zu sehen ist, wird erreicht, dass der IRQ eines PH Modes nur am Scheduler Ctrl_{in} ankommt, wenn die zugehörige FrameID ungerade ist.

Ein einfacher Scheduler, der nach Highest Priority First arbeitet, ist in Abbildung (4.8) als Moore Automat dargestellt. Der Zustand „Schedule Frame“ ist der Idle Zustand des Automaten. Der Automat besitzt für jeden PH Mode einen „GrantBus_t“ Zustand, wobei t die Nummer des jeweiligen PH Modes ist. Aus dem Idle Zustand gelangt man

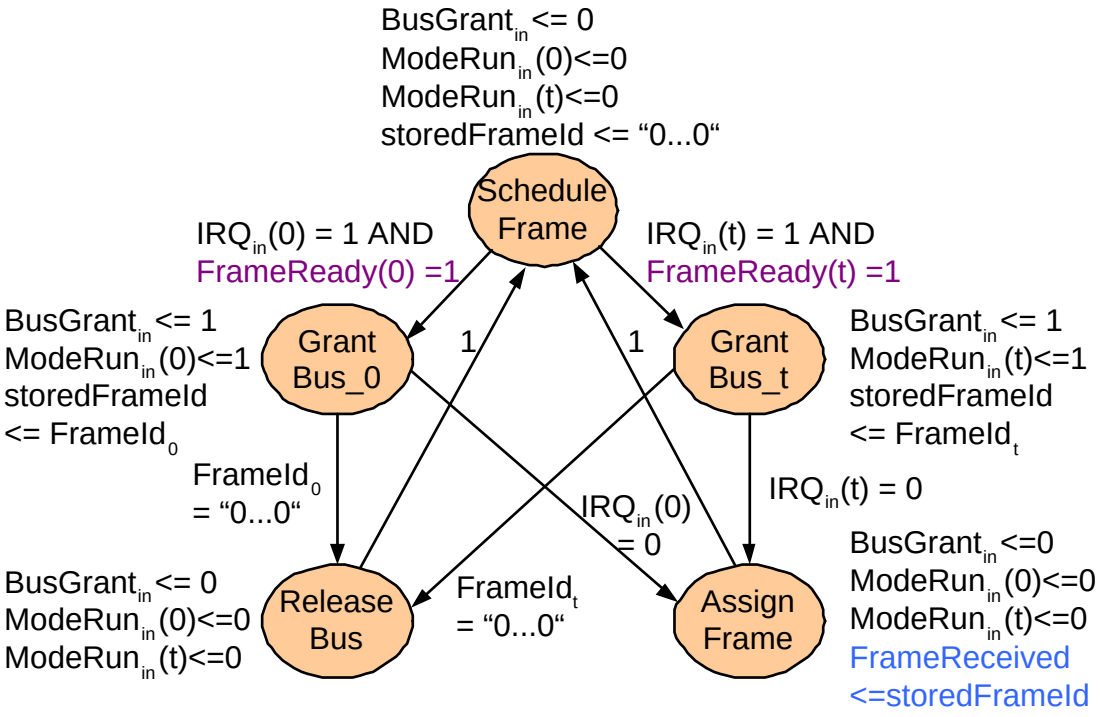

Abbildung 4.8: *Der HPF Scheduler.*

nur dorthin, wenn ein entsprechender IRQ gesetzt ist und das Scoreboard meldet, dass der aktuelle Frame für diesen PH Modus gelesen werden darf.

Im Zustand „GrantBus_t“ wird der Datenbus für einen PH Mode freigegeben. Dazu wird das Signal `ModeRun(t)` auf Logisch 1 gesetzt. Gleichzeitig wird das Signal `BusGrant` auf 1 gesetzt. Der Protokollautomat t ist, durch das Signal `ModeRun(t)<=1`, in den ersten Datenzustand des Frames übergegangen und versendet nun die Daten über den Datenbus, der im Protocol Handler Switch durch das Signal `BusGrant` freigegeben wurde. Diesen Vorgang wird Busarbitrierung genannt.

Es gibt zwei Wege den „GrantBus_t“ Zustand zu verlassen. Wenn die FrameID zur gleichen Zeit wie der IRQ auf „0“ gesetzt wird, bedeutet dies, dass der Frame nicht ordnungsgemäß gelesen wurde. Der Scheduler wechselt in den Zustand „ReleaseBus“ , gibt den Bus wieder frei und springt in den Zustand „ScheduleFrame”. Alle gelesenen Daten werden verworfen.

Wird der IRQ hingegen vor der FrameID auf „0“ gesetzt so wurde der Lesevorgang korrekt abgeschlossen. Der Scheduler wechselt in den Zustand „AssignFrame“ und sendet die gelesene FrameID an das Scoreboard. Mit einem unbedingten Zustandsübergang wechselt der Scheduler wieder in den Idle Zustand „ScheduleFrame“ .

4.2.3 Rekonfiguration

Die Control Unit soll die bestehenden Ideen zur Rekonfiguration [8] des Systems unterstützen. Dazu wird der Automat aus Abbildung (4.9) in die Control Unit integriert.

Nach einer externen Aktivierung, wechselt der Automat in den Zustand `Halt Mode`. Der gewünschte Protocol oder Sequence Handler Modus wird deaktiviert indem die Verbindung mit den Daten- und Steuerleitungen unterbrochen wird. Sobald der Modus deaktiviert ist, wechselt der Rekonfigurationsautomat in den Zustand `Reconfigure`. Nun kann der Modus durch einen neuen Modus ersetzt werden. Dies geschieht durch eine externe RCU (Reconfiguration Control Unit). Ist die Rekonfiguration abgeschlossen, so wechselt der Automat in den Zustand `Start Mode` und aktiviert den rekonfigurierten Modus. Das geschieht, indem der Modus wieder an die Daten- und Steuerleitungen angeschlossen wird. Der Rekonfigurationsautomat wechselt danach in den Ruhezustand `Run`. Der neue Modus kann wie gewohnt angesprochen werden.

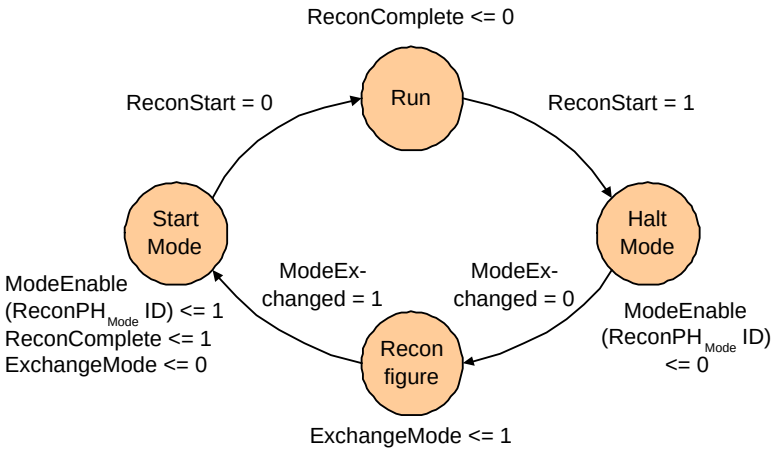

Abbildung 4.9: *Rekonfigurationsautomat in der Control Unit.*

5 Schedulabilityanalyse

Mit Hilfe einer Ausführbarkeitsanalyse und einer Schedulabilityanalyse soll vor der Synthese geprüft werden, ob der IFB allen Anforderungen der Tasks gerecht werden kann. Dieses Kapitel beschreibt verschiedene Annahmen und Voraussetzungen, die für die Ausführbarkeits- und die Schedulabilityanalyse notwendig sind.

Einige Voraussetzungen werden durch die Architektur vorgegeben. So gibt es zwei getrennte Scheduler (Kapitel 4.2.2), die für die Busarbitrierung zwischen Protocol Handler und Sequence Hander verantwortlisch sind.

5.1 Voraussetzungen für die Ausführbarkeits- und Schedulabilityanalyse

Als Grundlage für die Analysen werden folgende Annahmen festgelegt.

Annahme 1: Eine Task arbeitet stets korrekt und blockiert nie den Protokollablauf.

Annahme 2: Die angeschlossenen Tasks verursachen durch ihre Anforderungen keinen Deadlock im IFB. Dies würde auftreten, wenn alle Tasks einen Frame ausgeben wollen, vorher aber keine Daten zur Ausgabe an den IFB geliefert wurden.

Annahme 3: Das Scoreboard der Control Unit kann immer alle Konflikte, die durch IFD-Mappings zwischen Datenpaketen entstehen, auflösen. Somit entstehen keine Deadlocks bei der Berechnung der Ausgangsdatenpakete durch den Sequence Handler.

Annahme 4: Die Berechnung der Pakete ist immer möglich, so dass der IFB das Protokoll nie verzögert. Aus der Sicht von Tasks sind die Protocol Handler Modes immer Idle.

Annahme 5: Ein Protokoll darf keine toten und nicht erreichbaren Zustände besitzen. Für eine Analyse muss zudem sichergestellt sein, dass es nur einen Zyklus im Protokoll gibt. Gibt es Zyklen mit einer festen Anzahl n an Wiederholungen, werden diese aufgelößt, indem der betroffene Grundblock n-mal in das Protokoll eingefügt wird.

Annahme 6: Ein Frame ist nicht unterbrechbar.

Annahme 7: Für die Schedulabilityanalyse sind durch den Benutzer Perioden definiert worden.

Annahme 8: Nur Protokolle, die ausführbar sind, werden zur Schedulabilityanalyse zugelassen.

Für die Analysen müssen die Teile eines Protokolls berücksichtigt werden, die Nutzdaten enthalten. Dies sind die Frames. In einem Protokoll gibt es häufig mehr als einen Frame. Um die Analyse zu vereinfachen, werden alle Frames eines Protokolls in einen Ersatzframe umgewandelt. Dabei wird der Worst Case Fall angenommen.

Die Kenngrößen eines Frames sind in Tabelle (5.1) aufgelistet. Die Schedulabilityanalyse benötigt diese Größen des Ersatzframes für jedes Protokoll. Die Zeit t_P für eine Periode kann auch als Frequenz f angegeben werden.

Periode	t_P
Deadline	t_D
Ausführungszeit	t_{ex}
Frequenz	$f = \frac{1}{t_P}$

Tabelle 5.1: *Für das Scheduling relevante Kenngrößen eines Frames.*

Sollen direkt aufeinanderfolgende Frames zusammengefasst werden, so lassen sich mit folgenden Gleichungen Ersatzzeiten für den Worst Case Fall ermitteln.

$$t'_{ex} = \sum_{i=1}^{k} t_{ex_i} \tag{5.1}$$

$$t'_D = \sum_{i=1}^{k} t_{D_i} \tag{5.2}$$

$$f'_P = \max_i f_{P_i} \tag{5.3}$$

Für Frames in einem Protokoll, die parallel zueinander liegen, können folgenden Formeln genutzt werden um Ersatzzeiten zu bestimmen.

$$t'_{ex} = \max_i t_{ex_i} \tag{5.4}$$

$$t'_D = \min_i t_{D_i} \tag{5.5}$$

$$f'_P = \sum_i f_{P_i} \tag{5.6}$$

Die Bildung eines Ersatzframes für ein Protokoll soll nun anhand des Protokolls in Abbildung 5.1 demonstriert werden. Zunächst werden die Frames 3 und 4 zusammengefasst. Die Frames werden nacheinander ausgeführt, daher werden die Gleichungen (5.1), (5.2) und (5.3) verwendet.

Dabei entsteht das Protokoll in Abbildung (5.2). Die Frames 3 und 4 wurden durch den neuen Frame $3'$ ersetzt. Nun liegen die Frames 2 und $3'$ parallel zueinander. Mit

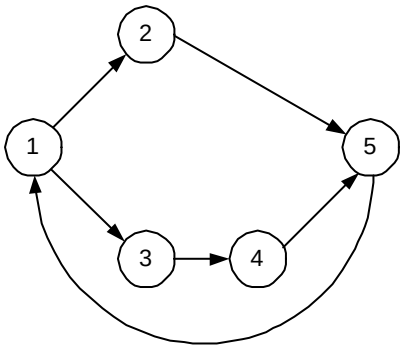

Abbildung 5.1: *Reduktion eines Protokolls Teil 1.*

Hilfe der Gleichungen (5.4), (5.5) und (5.6) können die beiden Frames zu Frame 2′ zusammengefasst werden.

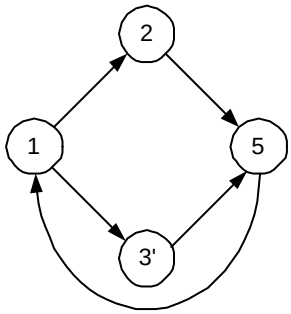

Abbildung 5.2: *Reduktion eines Protokolls Teil 2.*

Es entsteht Protokoll (5.3). Die drei Frames die nacheinander ausgeführt werden, können wieder mit den Gleichungen (5.1), (5.2) und (5.3) zusammengefasst werden. Dabei entsteht der Ersatzframe des Protokolls (Abbildung 5.4)).

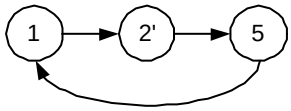

Abbildung 5.3: *Reduktion eines Protokolls Teil 3.*

Mit den ermittelten Daten für alle verwendeten Protokolle können nun die Ausführbarkeits- und Schedulabilityanalyse durchgeführt werden.

5.1.1 Ausführbarkeitsanalyse

Die Ausführbarkeitsanalyse soll überprüfen, ob der IFB für die angeschlossenden Tasks schnell genug arbeitet. Der Protokollautomat im Protocol Handler und die Aktivierung

Abbildung 5.4: *Reduktion eines Protokolls Teil 4.*

durch die Control Unit benötigen zusätzliche Zustände. Diese neuen Zustände im Protokoll einer Task können den Ablauf des Protokolls verzögern, da der IFB ist eine synchrone Architektur ist.

Die Ausführbarkeitsanalyse soll überprüfen ob durch die Verzögerungen im Protokollablauf alle Zeiten eingehalten werden können. In der Tabelle (5.2) sind für die verschiedenen Phasen der Kommunikation die benötigten IFB Takte aufgelistet.

	Name	IFB Teil	IFB Takte
1.	Establish incoming	PH, Ctrl	3
2.	Read Data	PH, SH	$6 \cdot \iota$
3.	Release	PH, Ctrl, Score.	3
4.	Process	SH, Score.	$2 + \psi$
5.	Establish outgoing	PH, Ctrl	3
6.	Write Data	PH, SH	$6 \cdot \omega$
7.	Release	PH, Ctrl, Score.	3

Anzahl eingehende Datenzustände	ι
Anzahl ausgehende Datenzustände	ω
Zustände der Datentransformation	ψ

Tabelle 5.2: *Zeiten für einen Protokollablauf.*

Ist das betrachtete Protokoll zeitbehaftet, so wird eine Zeitkorrektur, wie in Kapitel 4.1.1 beschrieben, durchgeführt. Sind die Zeiten der Transition nach der Zeitkorrektur größer oder gleich dem IFB Takt (siehe 2.4), so ist das Protokoll ausführbar.

In der Tabelle (5.2) wurde eine vollständige Kommunikation von einer Task A über den Protocol und Sequence Handler zu einer Task B aufgelistet. Die Schritte 1-3 nehmenen die Daten entgegen, Schritt 4 modifiziert diese und die Schritte 5-7 versenden die Daten. Um die Frage zu beantworten, ob ein Protokoll ausführbar ist, müssen nur die Schritte 1-3 und 5-7 betrachtet werden. Während dieser Zeit kommuniziert ein Protocol Handler Mode mit einer Task.

Wird ein nicht zeitbehaftetes Protokoll berachtet benötigt der IFB

$$3 + 6 \cdot \iota + 3 \text{ Takte}$$

um die Daten eines Frames mit einem Datenzustand entgegenzunehmen. Werden die IFB Takte umgerechnet, so erhält man bei einem IFB Systemtakt von 10kHz eine

Ausführungszeit von

$$\begin{aligned} f_{\text{IFB}} &= 10\text{kHz} \\ \iota &= 1 \\ t_{ex} &= f_{\text{IFB}} \cdot (3 + 6 \cdot \iota + 3) \\ &= \frac{(3 + 6 \cdot 1 + 3)}{10\text{kHz}} \\ &= 1,2\text{ms} \end{aligned} \tag{5.7}$$

Diese Ausführungszeit muss kleiner sein, als die vom Benutzer definierte Deadline $t_{ex} < t_D$. Ist das der Fall, so ist das Protokoll ausführbar.

Mit der Ausführbarkeitsanalyse kann nur die Kommunikation der Tasks mit dem IFB betrachtet werden. Zeitliche Abhängigkeiten zweier Tasks untereinander werden hier nicht berücksichtigt. Das IFD-Mapping müsste um eine Definition der Abhängigkeiten von Tasks erweitert werden.

5.1.2 Schedulabilityanalyse für 1 bis n Tasks

Eine Schedulabilityanalyse für ein System mit einer Task ist nicht nötig. Das Protokoll muss sich die Ressourcen nicht mit einem anderen Protokoll teilen. Es existiert daher ein Schedule, wenn das Protokoll Ausführbar ist.

Sollen zwei Protokolle auf dem IFB ausgeführt werden, so muss geprüft werden, ob es einen gültigen Schedule gibt. Sofern alle Informationen vorhanden sind, kann man die Analyse ausführen. Da Frames nicht unterbrechbar sind, muss ein Non Preemptive Scheduler verwendet werden.

Die Analyse kann für das zyklische Scheduling durchgeführt werden. Dieses Verfahren wurde in Kapitel 2.3.2 vorgestellt. Als Eingabe für die Analyse werden die zuvor berechneten Worst Case Frames der verschieden Protokolle verwendet. Die Protokolle werden gleichzeitig zum Zeitpunkt $t = 0$ms gestartet. Die Abbildung (5.5) zeigt zwei Tasks, die mit dem IFB verbunden sind. Zu beiden Tasks sind Deadline, Periode und Ausführungszeit angegeben.

Das zyklische Scheduling legt einen festen Ablauf der Protokolle fest. Dabei wird ein „Major Cycle“ gebildet, der sich immer wiederholt. Innerhalb dieses Major Cycles muss ein fester Ablauf definiert werden. In Tabelle (5.3) sind die Zeiten der Beispieltasks angegeben.

Um eine Lösung zu finden, wird zunächst die Länge des Major Cycles bestimmt. Das kgV der beiden Taskperioden ist 120ms. In Zeile 3 der Abbildung wird der Task A zweimal ausgeführt um auf die Länge des Major Cycles zu kommen.

Es gibt zwei Möglichkeiten die Tasks zu schedulen. Die Abbildung (5.3) zeigt die beiden möglichen Lösungen. Die zweite Lösung schlägt fehl, da die Deadline von Task A nicht

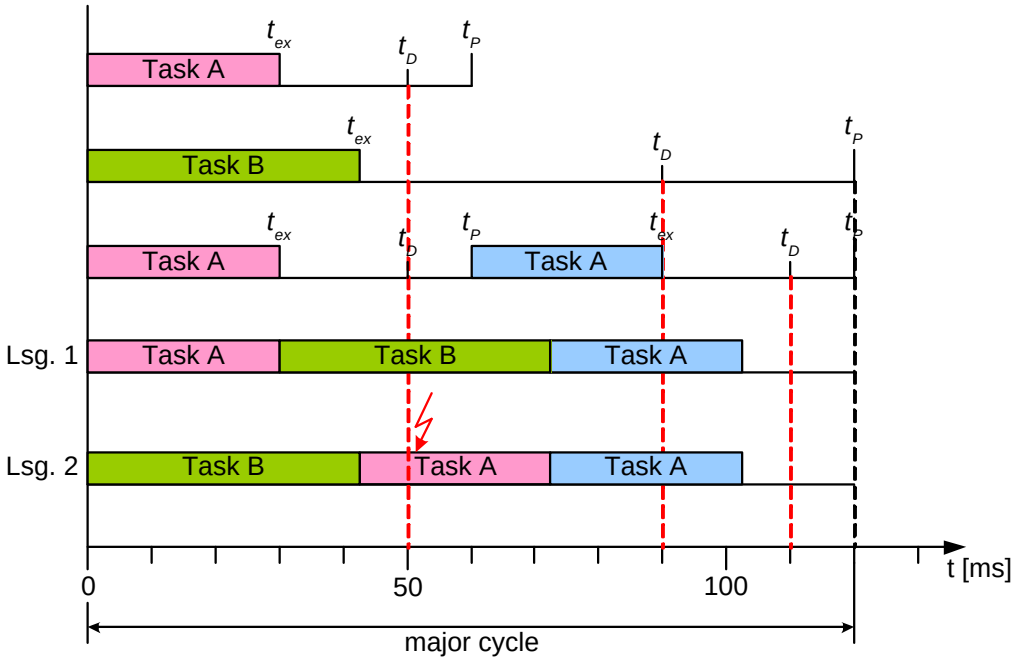

Abbildung 5.5: *Scheduling mit zwei Tasks.*

	Task A	Task B
t_P	60ms	120ms
t_D	50ms	90ms
t_{ex}	30ms	40ms

Tabelle 5.3: *Zeitangaben zum Beispiel in Abbildung (5.3).*

eingehalten werden kann. Mit Lösung 1 können alle Zeiten eingehalten werden. Wenn es keine Lösung gibt ist das System nicht schedulbar.

Bei der Analyse können die eingehenden und ausgehenden Frames immer getrennt behandelt werden, da diese unterschiedliche Ressourcen im IFB verwenden. Dies muss bei der Bildung der Worst Case Frames beachtet werden.

Eine Schedulabilityanalyse für n Tasks, wird wie die Analyse für zwei Tasks durchgeführt. Die Anzahl der möglichen Lösungen beträgt $n!$.

5.2 Verfahren für den IFB

Für den IFB ergeben sich 2 Scheduling Fälle. Im ersten Fall sind alle notwendigen Informationen wie Periode und Deadline im IFB modelliert worden, es kann zunächst die Ausführbarkeitsanalyse und im Anschluss die Schedulabilityanalyse durchgeführt werden. Aus den Analysen folgt ein statischer Scheduler für das zyklische Scheduling

oder die Erkenntnis, dass das System nicht schedulbar ist.

Der zweite Fall muss ohne Angaben zu Perioden auskommen. Damit fehlen die, für die Schedulabilityanalyse, notwendigen Informationen. In diesem Fall kann im IFB ein Scheduler eingesetzt werden, der anhand von statischen oder dynamischen Prioritäten die Buszugriffe regelt. Beispiele sind „Highest Priority First“ (statische Prioritäten) oder „First Come First Serve“ (dynamische Prioritäten)

Nur im ersten Fall können die Deadlines der Protokolle für harte Realzeit eingehalten werden.

6 Zusammenfassung und Ausblick

In diesem Kapitel wird die Studienarbeit mit einer Zusammenfassung der behandelten Themen abgeschlossen. Die entscheidenden Konzepte zur Erweiterung des IFBs werden dabei noch einmal angesprochen. Im Anschluss wird es einen Ausblick auf mögliche Erweiterungen der Control Unit geben.

6.1 Zusammenfassung der Arbeit

Im Rahmen dieser Studienarbeit wurde der IFB zu einem deadline-konformen Protokollkonverter für heterogene verteilte Anwendungen erweitert. Zunächst wurde dazu die Architektur des IFBs erweitert, damit mehrere duplexfähige Tasks unterstützt werden können. Dazu wurden die bisherigen Verbindungen der Komponenten in der Makro-Struktur auf dedizierte Verbindungen umgestellt.

Um zu verhindern, dass die Ressourcen des IFBs von mehreren Tasks gleichzeitig genutzt werden, wurde in die Control Unit eine zentrale Steuerung integriert. Diese Steuerung teilt sich auf zwei Bereiche auf. Der erste Bereich überwacht die Nutzdaten im Sequence Handler und steuert deren Umwandlung.

Der zweite Bereich regelt den Zugriff auf die Ressourcen zwischen Protocol und Sequence Handler mit Hilfe von Schedulern. Für die Scheduler kann eine Ausführbarkeits- und Schedulabilityanalyse zeigen ob der IFB den geforderten Echtzeitanforderungen gerecht werden kann.

Die so erweiterte Control Unit unterstützt mit den genannten Steuerungsmöglichkeiten, gepipelinete Kommunikationszyklen nach dem *Eingabe- Verarbeitung- Ausgabe* Prinzip.

6.2 Ausblick

Um den Interfaceblock weiter zu verbessern, können verschiedene Ansätze bertrachtet werden. So müssen zurzeit die Nutzdaten einer Task vollständig eingelesen worden sein, bevor ein Mode im Sequence Handler mit der Verarbeitung beginnen kann. Besonders Frames die sehr viele Datenzustände beinhalten verzögen die Abarbeitung der Daten enorm. Ziel währe es also entweder Frames in kleinere Subframes aufzuteilen, oder den Modes im Sequence Handler zu ermöglichen vollständige Datenpakete aus noch nicht

vollständig gelesenen Frames zu verwenden. Die Steuerung in der Control Unit muss dementsprechend angepasst werden.

Ein weiterer Punkt für Verbesserungen ist die Unterstützung für weitere Schedulingverfahren. Der Benutzer kann dann den für seine Anwendung sinnvollsten Scheduler auswählen.

Literaturverzeichnis

[1] Giorgio C. Buttazzo. Hard Real-Time Computing Systems. Kluwer Academic Publishers, 1997.

[2] Stefan Ihmor. Entwurf von Echtzeitschnittstellen am Beispiel interagierender Roboter. Diplomarbeit, Universität Paderborn 2001.

[3] Franz J. Ramming. RTOS 1/ IRTOS. Vorlesung, Universität Paderborn, 2004.

[4] Franz J. Ramming. Grundlagen der Technischen Informatik. Vorlesung, Universität Paderborn, 2004.

[5] J. Richling. Eigenschaften mobiler und eingebetteter Systeme. Vorlesung, Humbold Universität zu Berlin, 2003/04.

[6] J. Teich. Digitale Hardware/Software Systeme. Springer, Berlin, 1997.

[7] R.L Graham, E.L. Lawler, J.K. Lenstra, and A.H.G Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. Annals of Discrete Mathematics, 1979.

[8] S. Ihmor, W. Hardt. Synthesis of Communication Structures and Protocols in Distributed Embedded Systems submitted to RSP, 2005.

[9] Hermann Kopetz. Real-Time Systems: Design Principals for Distributed Embedded Applications. Kluwer Academic Publishers, Norwell, Massachusetts 02061 USA, 1997.

[10] Adrian Wiedemann. Das ISO-OSI-Modell. http://www.osi-modell.de. Webseite.

[11] Andrew S. Tanenbaum. Computer-Netzwerke. Prentice Hall, 2003. 4. überarbeitete Auflage.

[12] Tobias Loke. Synhtese von Kommunikationsstrukturen in verteilten eingebetteten Systemen. Studienarbeit, Universität Paderborn, 2005.