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

Synthese von Kommunikationsstrukturen in verteilten eingebetteten Systemen

Studienarbeit

Studiengang Informatik

von

Tobias Loke
Alte Brauerei 15
33098 Paderborn

betreut durch

Dipl.-Inform. Stefan Ihmor

vorgelegt bei

Prof. Dr. rer. nat. Franz Josef Rammig

im

März 2005

„Lehren bedeutet nicht,
etwas in jemanden hineinbringen,
sondern aus ihm heraus.“

Dank und Erklärung

Dieses Dokument entstand im Rahmen einer Studienarbeit in der Arbeitsgruppe von Prof. Dr. rer. nat. Franz Josef Rammig (HNI) der Universität Paderborn.

Für das interessante Thema der Studienarbeit möchte ich mich bei Franz J. Rammig und Stefan Ihmor bedanken. Besonderer Dank gilt Stefan Ihmor für seine hervorragende Arbeit als Betreuer dieser Studienarbeit. Weiterhin danke ich Dieter Averberg für das Korrekturlesen und seine geduldige Unterstützung bei Problemen mit Linux. Für das abschließende Korrekturlesen dieser Arbeit danke ich besonders Giulia Calani.

Abschließend möchte ich meinen Eltern und meinem Bruder danken, die mich stets unterstützt haben.

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.

Paderborn, im März 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Der Markt für eingebettete Systeme wächst sehr stark und wird nach Einschätzung von Experten bald den Markt für terminal-basierte Anwendungen übertreffen. Eingebettete Systeme findet man heute in fast allen Bereichen des täglichen Lebens, daher stellen sie einen bedeutenden wirtschaftlichen Faktor dar.

In eingebetteten Systemen steigen die Anforderungen an die Funktionalität immer weiter. Zur Realisierung der benötigten Funktionalität bei einer möglichst kurzen Entwicklungszeit, erlangt das so genannte *IP-based Design (Intelectual Property)* immer stärker an Bedeutung. Bei dem IP-based Design werden bereits vorhandene und getestete Komponenten *(IPs)* wieder verwendet. Daher ist die Integration von bereits vorhandenen Komponenten ein wesentlicher Bestandteil des Entwurfsprozesses von eingebetteten Systemen.

In verteilten eingebetteten Systemen wird so eine komplexe Funktionalität häufig durch die Komposition mehrerer Teilkomponenten realisiert. Dies erfordert eine Interaktion der Teilkomponenten. Die interagierenden Komponenten eines eingebetteten Systems sind die Bestandteile der so genannten *Kommunikationsstruktur*. Bei verteilten Systemen mit einer Vielzahl an Kommunikationskomponenten steigt die Anzahl der möglichen Kommunikationsstrukturen sehr schnell.

Eine weitere Herausforderung im Entwurfsprozess von eingebetteten Systemen stellt die Heterogenität der Komponenten dar, denn es existiert kein einheitlicher Standard unter den Herstellern von IPs. Verfahren zur Unterstützung der Entwickler durch einen möglichst automatisierten Design Prozess gewinnen daher immer mehr an Bedeutung. Das Problem bei der Integration inkompatibler Komponenten wird oft auf eine der beiden folgenden Arten umgangen:

1. Es wird auf die Verwendung inkompatibler Komponenten verzichtet. Dieser Ansatz führt dazu, dass nur Komponenten von einem Hersteller verwendet werden können. Die Auswahl der Komponenten ist dadurch stark eingeschränkt.

2. Zur Interaktion zwischen den Komponenten werden standardisierte Bussysteme verwendet. Jede Komponente, die in eine Kommunikationsstruktur aufgenommen werden soll, muss um eine entsprechende Schnittstelle für ein solches Bussystem

erweitert werden. Dies ist ein fehleranfälliger und aufwendiger Bestandteil der Entwicklung. Bei kleineren Komponenten führt er zu einem ungünstigen Verhältnis zwischen der Größe der Schnittstelle für das Bussystem und der eigentlichen IP.

Beide Möglichkeiten sind entweder unflexibel oder bringen einen hohen Aufwand mit sich. Zur Lösung dieses Problems wurde der Ansatz des *Interface Synthesis Design Flows* geschaffen. Er ermöglicht mit Hilfe einer Adapterstruktur, dem so genannten *Interface Block (IFB)*, die Einbindung inkompatibler Komponenten, ohne die Komponenten selbst ändern zu müssen. Bei der Erstellung von komplexeren eingebetteten Systemen, in denen eine Vielzahl von Komponenten an den Kommunikationsstrukturen beteiligt sind, zeigen sich Einschränkungen bei Verwendung des IFS-Flows:

- Die Kommunikationsstruktur kann im IFS-Flow jeweils nur um eine Gruppe von interagierenden Komponenten erweitert werden. Dies ist dadurch bedingt, dass der jeweils benötigte IFB einzeln erstellt werden muss. Es können nicht alle Kommunikationsstrukturen auf einmal erstellt werden.
- Die Erzeugung von Kommunikationsstrukturen mit Hilfe des IFS-Flows erlaubt eine Vielzahl von möglichen Lösungen für die Architektur aus interagierenden Komponenten und IFBs. Die Lösungen unterscheiden sich hinsichtlich ihrer Qualität im Bezug auf den Verbrauch von Ressourcen in der gegebenen Systemarchitektur. Die vorhandenen Ressourcen entscheiden darüber, ob eine komplexe Kommunikationsstruktur überhaupt realisiert werden kann.

Die aufgeführten Einschränkungen des IFS-Flows resultieren in einem reduzierten Automatisierungsgrad.

1.2 Aufgabenstellung

Für die optimierte Synthese von Kommunikationsstrukturen in verteilten eingebetteten Systemen wird ein erweiterter IFS-Flow benötigt. Im Rahmen dieser Studienarbeit sind Verfahren aufzuzeigen, mit denen die beschriebenen Einschränkungen des IFS-Flows kompensiert werden können.

Für die Synthese der vollständigen Kommunikationsstruktur ist ein Verfahren zur automatisierten Verteilung von IFBs zu erstellen. Das Verfahren muss möglichst effizient mit den vorhandenen Ressourcen umgehen, und z.B. die Länge von Verbindungen zwischen Komponenten minimieren.

Neben der automatisierten Verteilung ist ein Ansatz für anwendungsspezifische Protokolle (*Dedicated Protocols*) zu erstellen. Dabei sollen mehrere gegebene Protokolle zu einem Dedicated Protocol verschmolzen werden, um den Ressourcenverbrauch auf Kommunikationswegen zu verringern und somit komplexere Kommunikationsstrukturen zu ermöglichen.

1.3 Aufbau der Arbeit

Nach dieser Einleitung folgt das Kapitel zwei, in dem die verwendete Definition von Protokollen angegeben ist. Anschließend werden Begriffe aus der Graphentheorie vorgestellt, die später im Kapitel fünf verwendet werden. Am Ende des Kapitels wird der Begriff des Clusterings erläutert und die Prinzipien des Hierarchischen und Partitionierenden Clusterings genauer ausgeführt.

Zu Beginn des Kapitels drei wird der IFS-Flow beschrieben. Danach werden die Modelle des Spezifikationsgraphen und des Kommunikationsgraphen vorgestellt. Es folgt eine Übersicht zur Funktion des IFD-Mappings. Abschließend wird eine Erläuterung zum Interface Block und seinen strukturellen Komponenten gegeben.

Im Kapitel vier werden zunächst die Auswirkungen verschiedener Systemtopologien auf die benötigen Ressourcen der Systemarchitektur beschrieben. Für die automatische Verteilung werden Kostenfunktionen und Strategien vorgestellt, die bei der Verteilung berücksichtigt werden. Nach Erläuterung der Bedeutung von Clustering für die Verteilung von IFBs, wird das Verfahren zur Verteilung von IFBs selbst vorgestellt.

Kapitel fünf formuliert die Anforderungen für die Verwendung von Dedicated Protocols. Darauf folgt die Definition des Merged Protocols als mögliche Realisierung eines Dedicated Protocols. Das Konzept des Merged Protocols erfordert die Zerlegung eines Protokoll-Zustandsautomaten in seine Grundblöcke. Diese Zerlegung wird von einem Protokoll-Parser vorgenommen, der im folgenden Abschnitt erläutert wird. Am Ende des Kapitels wird der Algorithmus für die Berechnung eines Merged Protocols angegeben und die Auswirkungen der Verwendung eines Merged Protocols auf die Kommunikationsstruktur beschrieben.

Im letzten Kapitel sechs wird ein Überblick über die vorgestellten Verfahren mit einem Ausblick auf mögliche Erweiterungen gegeben.

2 Grundlagen

2.1 Protokolle

In dieser Studienarbeit wird ein Verfahren erweitert, mit dem die automatische Synthese von Schnittstellen und Protokollen ermöglicht wird, der *Interface Synthesis-Design-Flow (IFS-Design-Flow)*. Im IFS-Design-Flow sind Protokolle als Teil einer Schnittstellenbeschreibung definiert. In den Protokoll-Beschreibungen wird das Verhalten von Protokollen in Form eines endlichen Zustandsautomaten beschrieben (Finite State Machine = FSM). Die Beschreibung des Verhalten von Protokollen durch eine FSM und die verwendete Definiton der FSM wird im Weiteren beschrieben.

Ein Protokoll definiert das Verhalten einer Menge von gerichteten Verbindungen. Die Verbindungen werden durch so genannte Protocol Pins repräsentiert. Das Verhalten der Signale auf diesen Verbindungen und die Signalausgabe werden durch eine FSM beschrieben. In Protokollen ist das Verhalten durch eine Abfolge von Zuständen, die Signalausgabe durch eine entsprechende Ausgabe in einem Zustand modelliert. Die verwendete FSM entspricht einem Moore Automat.

Definition: $Protocol - FSM = \{S, X, Y, \delta, \lambda\}$

- S: Zustände
- X: Eingabe ($\Rightarrow$ Externe Signale für Transitionsbedingungen)
- Y: Ausgabe ($\Rightarrow$ Definiert den Wert eines Signals an einem Protocol Pin)
- δ: Zustandsübergangsfunktion: $(S \times (X, (Y|Y = IC))) \rightarrow S$
- λ: Ausgabefunktion: $S \rightarrow Y$

2.2 Graphentheorie

Zum Beschreiben des Verhaltens von Protokollen werden Protokoll-FSMs verwendet. Diese können als gerichtete Graphen visualisiert werden. In späteren Abschnitten dieser Studienarbeit werden Definitionen für Strukturen verwendet, welche sich am besten mit Begriffen aus der Graphentheorie beschreiben lassen.

Ein Graph ist eine Visualisierung von Objekten und deren Beziehungen untereinander. Die Objekte werden im Graphen durch Knoten repräsentiert. Die Beziehungen zwischen den Objekten werden durch Kanten dargestellt. Im folgenden Abschnitt wird die, in dieser Arbeit verwendete, Definition von Graphen und ihren Eigenschaften gemäß [Har69] vorgestellt.

Ein Graph $G = (V, E)$ besteht aus:

- V: eine Menge von Knoten (Vertices) $V = \{v_1, v_2, ..., v_n\}$
- E: einer Menge von Kanten (Edges) mit $\forall e_i \in E : e = (v_i, v_j)$

In einem ungerichteten Graph ist jede Kante $e = (v_1, v_2)$ ein ungeordnetes Paar. Bei einem gerichteten Graphen ist jede Kante $e = (v_1, v_2)$ ein geordnetes Paar. Wenn $e_j = (v_1, v_2)$ und $e_k = (v_2, v_1)$ dann gilt $e_j \neq e_k$. Des Weiteren sei die Menge der Knoten eines Graphen G mit $V(G)$ bezeichnet. Die Menge der Kanten eines Graphen G sei mit $E(G)$ bezeichnet.

Pfad: Ein Pfad in einem Graphen ist eine Folge von Kanten. Für einen Pfad p in einem Graphen G von v nach w mit $v, w \in V$ ist eine Abfolge von Kanten $p = (e_1, e_2, ...e_n)$, wenn gilt $v \in e_1, e_i \cap e_{i+1} \neq \emptyset$ und $w \in e_n$.

Pfadlänge: Die Länge eines Pfades entspricht der Anzahl der Kanten auf dem Pfad.

Kreis: Ein Kreis ist ein Pfad in einem Graphen, bei dem der erste und letzte Knoten gleich sind. Da die sich die Abfolge der Knoten in einem Kreis wiederholt, wird ein Kreis auch als *Zyklus* bezeichnet.

Grad: Der Grad $d(v)$ eines Knoten v ist die Anzahl der Kanten e_i für die gilt: $v \in e_i$

Disjunkte Graphen: Zwei Graphen sind disjunkt wenn sie keine gemeinsamen Knoten besitzen. G_1 ist disjunkt zu G_2 wenn gilt: $V(G_1) \cup V(G_2) = \emptyset$

Verbindung: Auf Graphen lassen sich verschiedenste Operationen definieren wie z.B. das Komplement, die Vereinigung oder Differenz von Graphen. Eine der möglichen Operationen für die Kombination von Graphen ist die Verbindung (siehe Abbildung 2.1). Der Graph G sei das Ergebnis der Verbindung zweier disjunkter Graphen G_1 und G_2. Dann ist G definiert durch:

$$G = G_1 + G_2 \text{ mit}$$
$$V(G) = V(G_1) \cup V(G_2)$$
$$E(G) = E(G_1) \cup E(G_2) \cup \{v_1 v_2 \mid v_i \in V(G_i), i = 1, 2\}$$

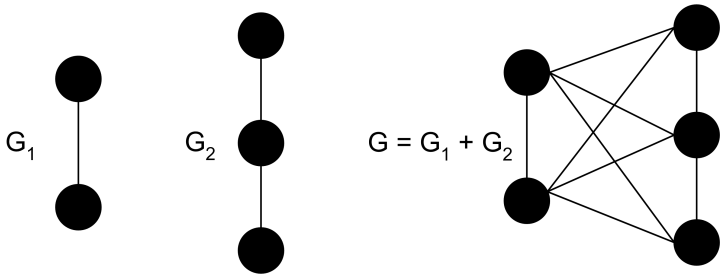

Abbildung 2.1: Verbindung von zwei Graphen

2.3 Protokoll Grundblöcke

Protokolle beinhalten neben den Signalen für die zu übertragenden Daten *(Nutzdaten)* weitere Signale zur Steuerung des Ablaufs oder Sicherung der Datenübertragung *(Redundanz)*. Je nach Art des Protokolls kann man daher den Ablauf des Protokolls in mehrere Abschnitte aufteilen, die entweder der Steuerung des Ablaufs oder der Datenübertragung dienen. Diese werden im weiteren Verlauf als *Grundblöcke* bezeichnet. Die Zerlegung eines Protokolls in seine Grundblöcke ist an mehreren Stellen des IFS-Flow relevant.

Analog zur Definition von Grundblöcken im Software-Design können Grundblöcke auch für Protokolle definiert werden. Gegeben sei der Graph einer Protokoll-FSM. Die Zustände des Automaten werden durch Knoten, die Transitionen durch die Kanten eines gerichteten Graphen repräsentiert.

Definition *Grundblock* (Basic Block = BB): *Ein Grundblock ist eine geordnete Menge von Zuständen, die auf einem kreisfreien Pfad maximaler Länge in einem Protokoll Graphen liegen. Auf diesem Pfad dürfen nur der erste und der letzte Knoten einen Grad größer eins haben. Der Knoten welcher dem Startzustand der Protokoll-FSM entspricht, darf nur als erster Knoten in einem solchen Pfad enthalten sein.*

Ein Grundblock beginnt, wenn eine der folgenden Bedingungen zutrifft:

1. Der Startzustand einer Protokoll-FSM ist immer der Anfang eines Grundblocks.
2. Zustände die nach einem Zustand mit mehreren Folgezuständen liegen (*Gabelung, fork of control*) sind immer der Anfang von Grundblöcken.
3. Zustände welche Nachfolger von mehreren Zuständen sind (*Zusammenführung, merge of control*) sind ebenfalls Anfangszustände eines Grundblocks.

Nur am Anfang oder am Ende eines BBs kann es daher zu einer Verzweigung im Ablauf eines Protokolls kommen. Im Grundblock selbst gibt es keine Verzweigung. Datenpakete sollten in syntaktisch korrekten Protokollen immer vollständig übertragen werden. Die vorzeitige Beendigung der Übertragung eines Datenpaketes sollte im Normalfall nicht

im Ablauf eines Protokolls vorkommen. Für den IFS-Flow wird vorausgesetzt, dass Datenpakete nicht über die Grenzen eines Grundblocks hinausgehen.

Wenn der Graph eines Protokollautomaten nur durch Grundblöcke und Transitionen zwischen diesen Grundblöcken dargestellt wird, so bezeichnen wir das im Weiteren als Kontrollflussgraph (Control Flow Graph = CFG).

Definition *Kontrollflussgraph* (CFG):
Ein Kontrollflußgraph $CFG = \{V; E\}$ besteht aus
$V = \{\text{Grundblöcke}\}$
$E \subseteq V \times V$, eine Menge von Kanten (= *Transitionen*) zwischen Grundblöcken.

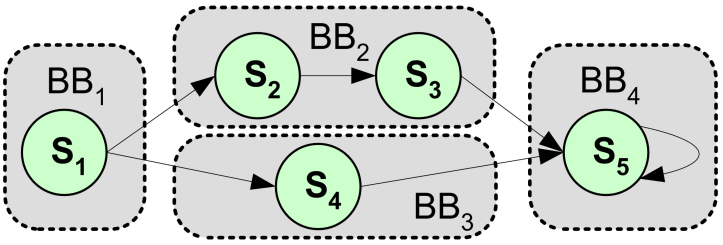

Abbildung 2.2: Beispiel für einen Control Flow Graph

Abbildung 2.2 zeigt einen CFG der aus 4 Grundblöcken BB_1 bis BB_4 besteht. Der CFG wurde zu einem Protokoll mit den Zuständen S_1 bis S_5 erstellt.

2.4 Clustering

Im folgenden Abschnitt wird der Begriff des Clusterings erläutert. Bei dem Clustering werden gleichartige Objekte in Gruppen (Cluster) zusammengefasst. Clustering wird in sehr unterschiedlichen Bereichen angewendet. Beispiele finden sich unter anderem in der Biologie, Medizin oder Mathematik. Viele Entscheidungs-, Optimierungs- und Lernprobleme können durch den Einsatz von Clusteringalgorithmen gelöst werden.

Aufgrund der vielen Anwendungsbereiche gibt es zahlreiche unterschiedliche Algorithmen, welche jeweils eine speziell auf das Problem zugeschnittene Form des Clusterings realisieren. Welche Objekte in einem Cluster zusammengefasst werden, wird durch die Ähnlichkeit der Objekte zueinander festgelegt. Die Ähnlichkeit kann durch ein oder mehrere Attribute eines Objekts bestimmt werden. Man unterscheidet zwischen quantitativen und qualitativen Attributen. Ähnlichkeit kann durch eine beliebige Größe definiert werden.

Oft wird die Distanz zwischen Objekten für das Bestimmen der Ähnlichkeit verwendet. Je kleiner die Distanz, desto höher die Ähnlichkeit. Man spricht dann von einer Distanzfunktion. Die Distanz zwischen zwei Elementen x und y der Menge D, wird mit $d(x, y)$

bezeichnet. Für die Distanzfunktion $d(x, y)$ müssen drei Bedingungen erfüllt sein:

$$\forall x, y \in D : d(x, y) \in \mathbb{R}^+ \tag{2.1}$$
$$d(x, y) = 0 \Leftrightarrow x = y \tag{2.2}$$
$$d(x, y) = d(y, x) \tag{2.3}$$

Allgemeiner betrachtet kann Clustering als eine spezielle Form der Klassifizierung gesehen werden. In [AKJ88] wird Clustering daher folgendermaßen beschrieben: "A clustering is a type of classification imposed on a finite set of objects". Der Ausgang einer Klassifikation ist eine Menge von Klassen und Objekte deren Zugehörigkeit zu den einzelnen Klassen bekannt ist. Diese bekannte Menge von Objekten wird als Trainingsmenge bezeichnet. Mit Hilfe der Informationen dieser Trainingsmenge wird dann versucht neue, unbekannte Objekte aufgrund ihrer Eigenschaften in Klassen einzuteilen. Da die Objekte und die Klassen der Trainingsmenge bereits bekannt sein müssen, wird diese Art der Klassifikation als *überwachtes Lernen* bezeichnet. Dem gegenüber steht die als *unüberwachtes Lernen* bezeichnete Art der Klassifizierung, das Clustering (siehe Abbildung 2.3).

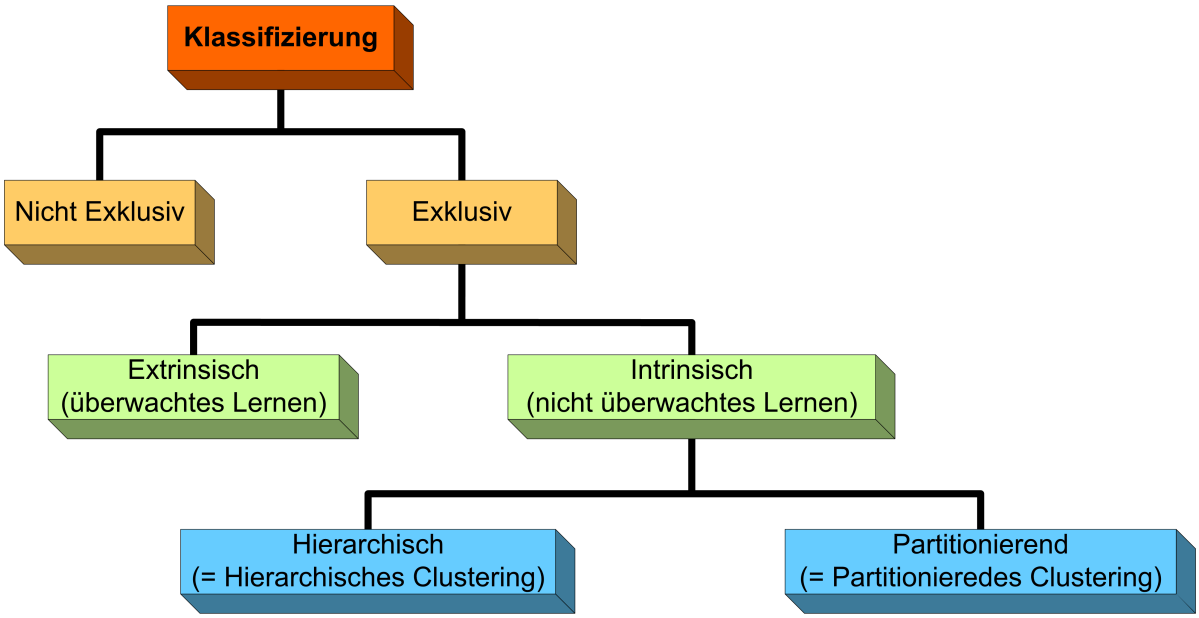

Abbildung 2.3: Zusammenhang von Klassifiktationsproblemen nach [GNL67]

Dieses kann wiederum in verschiedene Unterarten aufgeteilt werden, was sich nach der Art des Verfahrens richtet. Gemäß der Aufteilung nach [GNL67] sollen im folgenden die beiden Hauptarten der Verfahren vorgestellt werden, das hierarchische und das partionierende Clustering.

2.4.1 Hierarchisches Clustering

Bei dem hierarchischen Clustering wird eine Menge von Objekten rekursiv in eine Hierarchie von Clustern aufgeteilt, bis jeder Cluster genau ein Objekt enthält. Die Ergebnismenge der Cluster muss nicht vollständig disjunkt sein. Cluster auf tieferen Ebenen sind Untermengen von Clustern auf höheren Ebenen. Die Hierarchie der Ergebnismengen kann in einem Baumdiagramm (Dendogramm, siehe Abbildung 2.4) dargestellt werden. Die Wurzel des Baums entspricht der gesamten Menge der Objekte. Die Blätter entsprechen jeweils einem einzelnen Objekt. Die Darstellung der Hierarchie in einem

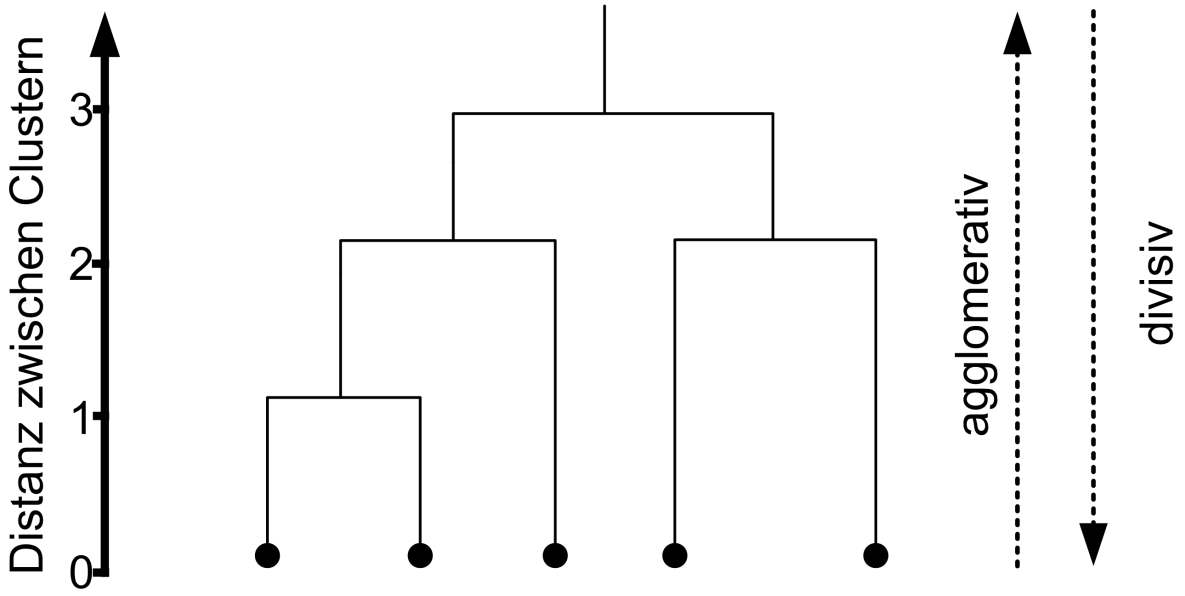

Abbildung 2.4: Beispiel für ein Dendogramm

Dendrogramm liefert eine schnelle Übersicht weiterer Eigenschaften der Objekte und Cluster zueinander. Je weiter eine Ebene des Dendogramms von der Wurzel entfernt liegt, desto größer ist die Nähe der einzelnen Objekte in einem Cluster. Mit steigendem Abstand zwischen Clustern auf einer Ebene steigt ihre Unterschiedlichkeit.

Man kann nur einen Teil der Mengen, welche ein hierarchischer Clusteringalgorithmus liefert, als Ergebnis betrachten. Wenn man nur Cluster auf einer bestimmten Ebene betrachtet erhält man disjunkte Cluster. Wählt man tiefere Ebenen erhält man eine feinere Einteilung mit mehr Clustern als in einer höheren Ebene. Algorithmen welche ein hierarchisches Clustering erzeugen benötigen eine Distanzfunktion um zwei einzelne Objekte zusammenzufassen und eine Distanzfunktion für das Zusammenfassen von Mengen von Objekten (Clustern). Im Folgenden werden einige verschiedene Arten von Verfahren zur Durchführung von hierarchischen Clustering beschrieben.

Agglomeratives Clustering

Bei dem agglomerativen Verfahren wird ein bottom-up Ansatz verwendet, um die Hierarchie der Cluster aufzubauen. Als Ausgangslage verwendet man Cluster mit genau einem Element (unterste Ebene). Die nächste Ebene der Hierarchie wird durch Zusammenfassen von mehreren Clustern auf unterster Ebene erzeugt. Dieser Vorgang wird so lange wiederholt, bis man auf der obersten Ebene der Hierarchie angekommen ist. Auf dieser Ebene sind alle Objekte in einem Cluster zusammengefasst.

Diversives Clustering

Diversive Verfahren verwenden einen top-down Ansatz und arbeiten genau in umgekehrter Reihenfolge zu agglomerativen Verfahren. Alle Objekte werden zuerst in einem Cluster zusammengefasst. Dieses wird in kleinere Cluster aufgeteilt, welche dann die nächste Ebene der Hierarchie bilden.

Eine Möglichkeit besteht darin ein Objekt mit der größten Distanz zu den übrigen Objekten zu suchen. Aus diesem Objekt wird ein neuer Cluster gebildet. Dann wird die durchschnittliche Distanz zwischen einem Objekt des ursprünglichen Clusters und allen Objekten des neu gebildeten Cluster berechnet. Von diesem Wert wird die durchschnittliche Distanz des gewählten Objektes zu allen anderen Objekten im gleichem Cluster abgezogen. Der so erhaltene Wert wird für jedes Objekt im ursprünglichem Cluster ermittelt. Das Objekt bei dem so der höchste Wert ermittelt wurde, ist den Objekten im neuen Cluster ähnlicher als den Objekten im eigenen Cluster.

Dieses Objekt wird in den neuen Cluster verschoben. Dieser Vorgang wird so lange wiederholt, bis alle errechneten Werte negativ sind. Dann ist die Menge der Objekte für das neue Cluster vollständig. Die so erhaltenen Cluster können nach der gleichen Methode weiter aufgeteilt werden, bis alle Cluster der untersten Ebene genau ein Element enthalten.

Inkrementelles Clustering

Bei inkrementellen Verfahren wird jeweils ein Objekt betrachtet und in die Cluster der Hierarchie eingefügt. Dabei wird die Hierarchie und damit der Graph des Dendogramm inkrementell aufgebaut (siehe auch [ZRL96]).

Ausgangslage ist ein Cluster mit einem Element. Dieser Cluster bildet die Wurzel der Hierarchie und des Dendogramm-Graphen mit dem man die Hierarchie darstellen kann. In die Cluster der Hierarchie werden weitere Objekte nach festen Kriterien eingefügt. Erfüllt ein Objekt die Bedingung für die vorhandenen Cluster nicht, werden weitere Cluster in die Hierarchie eingefügt. Die Knoten und Blätter des Dendogrammes entstehen so inkrementell.

2.4.2 Partitionierendes Clustering

Verfahren, die ein partitionierendes Clustering durchführen, liefern nach [AKJ88] zu n Objekten aus einem d-dimensionalen Vektorraum eine Aufteilung der n Objekte in k Cluster, so dass die Objekte in einem Cluster zueinander ähnlicher sind als zu Objekten in einem anderen Cluster.

Der Wert für k kann je nach Verfahren festgelegt oder durch das Clustering errechnet werden. Ein partitionierendes Verfahren teilt also eine Menge von Objekten so in disjunkte Cluster ein, dass jedes Objekt in genau einem Cluster ist und ein Cluster aus mindestens einem Objekt besteht.

Da die Cluster, welche das Ergebnis eines partitionierenden Verfahren darstellen, alle disjunkt sind, gibt es keine Hierarchie der Cluster untereinander. Für das Partitionierende Clustering wird häufig eine Bewertungsfunktion verwendet um die Qualität eines Clusterings auszudrücken. Mit Hilfe der Bewertungsfunktion wird versucht eine optimale Lösung für ein Clusteringproblem zu finden. Ist die Bewertungsfunktion eine Kostenfunktion, so wird ein optimales Clustering gefunden, indem die Kostenfunktion minimiert wird.

Um die richtige Aufteilung der Objekte auf die Cluster zu finden, könnte man alle möglichen Kombinationen ausprobieren. Die Anzahl der möglichen Kombinationen ist ein entscheidender Faktor für den Aufwand, der bei diesem Vorgehen entstehen würde.

Definition: *Es sei M eine Menge, es sei k eine natürliche Zahl.*
Eine Menge $\{M_1, M_2, ..., M_k\}$ von k paarweise disjunkten nicht leeren Teilmengen von M mit $M = \bigcup_{i=1}^{k} M_i$ heißt eine k-Partition von M.

Partitionierendes Clustering von n Objekten auf k Cluster bedeutet im mathematischen Sinne eine k-Partition von n zu bilden.

Die Anzahl der möglichen k-Partitionen einer n-elementigen Menge wird mit $S_{n,k}$ bezeichnet [KHK96]. Wenn n und k natürliche Zahlen sind, dann gilt für $S_{n,k}$:

$$S_{n,k} = S_{n-1,k-1} + k \cdot S_{n-1,k}$$

$$S_{n,k} = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-i} \binom{k}{i} (i)^n$$

Die Lösungen für S(n,k) sind die so genannten *Stirling Zahlen zweiter Ordnung*[1]. Bei 19 Elementen und 4 Partitionen ist $S(n,k) = 11.259.666.000$. Der Lösungsraum ist auch bei relativ kleinen Werten für n und k bereits sehr groß. Alle möglichen Lösungen zu errechnen und zu Bewerten ist daher oft nicht durchführbar.

Die meisten Verfahren verfolgen daher einen iterativen Ansatz, bei dem versucht wird, ein initiales Clustering zu optimieren. Die Qualität der gefundenen Lösung kann dabei

[1] James Stirling (22. April 1692 - 1770)

sehr stark von dem gewählten initialen Clustering abhängig sein. Es besteht die Gefahr, dass die gefundene Lösung nur aufgrund eines lokalen Extremum der Bewertungsfunktion akzeptiert wird. Durch wiederholtes Durchführen des Clusterings mit unterschiedlichen initialen Clusterings kann man mehrere Lösungen erzeugen. Nimmt man von diesen Lösungen die optimale, so kann die Wahrscheinlichkeit, eine Lösung aufgrund eines lokalen Extremum der Bewertungsfunktion zu akzeptieren, verringert werden.

Das Erstellen eines guten Clustering Algorithmus wird dadurch erschwert, dass es meist keine eindeutig Lösung für die Optimierung einer Bewertungsfunktion gibt. Die Optimierung der Bewertungsfunktion ist dann eine systematische Suche im Lösungsraum. Alle diese Faktoren führen zu einer immensen Anzahl von Ansätzen für Partitionierendes Clustering.

3 Interface Synthesis Design Flow

Alle im Rahmen dieser Arbeit erstellten Verfahren stehen im Zusammenhang mit dem Ansatz des *Interface Synthesis Design Flow*. Dieser wird in dem folgenden Kapitel, zusammen mit den Modellen des Spezifikations- und Kommunikationsgraphen, genauer erläutert.

3.1 Motivation für automatisiertes Design

Das Gebiet der eingebetteten Systeme entwickelt sich rasant hinsichtlich Komplexität und Vielfalt. Die meisten eingebetteten Systeme bestehen aus verteilten aber hochgradig untereinander verbundenen Anwendungen. Zuverlässigkeit und Effektivität sind wichtige Eigenschaften für das Design und die Implementation eines Kommunikationssystems. Die Verwendung von standardisierten Schnittstellen und Kommunikationsprotokollen sind wichtige Techniken, um die Zuverlässigkeit und Effektivität sicher zu stellen.

Die erhöhte Komplexität und die Notwendigkeit von kurzen Entwicklungszyklen zwingen den Designer förmlich die Wiederverwendung von Intellectual Properties (IP) zu berücksichtigen. Da die IPs von einer Vielzahl von verschiedenen Herstellern angeboten werden, ist man zwangsläufig mit verschiedensten inkompatiblen Schnittstellen und nicht konformen Protokollen konfrontiert. Daher müssen durch den Designer eines Systems Adapter erstellt werden, mit deren Hilfe IPs eingebunden werden können. Ein vollständiges Verständnis aller beteiligten Schnittstellen ist für diesen Prozess erforderlich. Die Einbindung bereits vorhandener IPs ist einer der zentralen und auch fehleranfälligen Aspekte in der Entwicklung von neuen eingebetteten Systemen [PRSV98].

Der so genannte *Interface Synthesis Design Flow* (*IFS-Flow*) ist eine Methode zum modellieren und automatischen Synthetisieren von rekonfigurierbaren Schnittstellen in eingebetteten Systemen (siehe Abbildung 3.2). Die Integration und flexible Wiederverwendung von IPs war ein integraler Bestandteil bei der Entwicklung des IFS-Flow. Daher bietet der IFS-Flow die Möglichkeit einer automatisierten Synthese von Schnittstellen als Adapter für inkompatible IPs [Ihm04].

Bei diesem Verfahren ist das vollständige Verständnis der einzubinden IP nicht mehr erforderlich. Das bedeutet auch, dass die innere Funktionalität einer IP geheim gehalten werden kann (*black box*). Diese Eigenschaften ermöglichen einen flexiblen und schnellen Entwurf von Prototypen. Für die automatisierte Integration von IPs muss nur die

Schnittstellenbeschreibung (Interface Description, IFD) zur Verfügung stehen. Die *IFS*

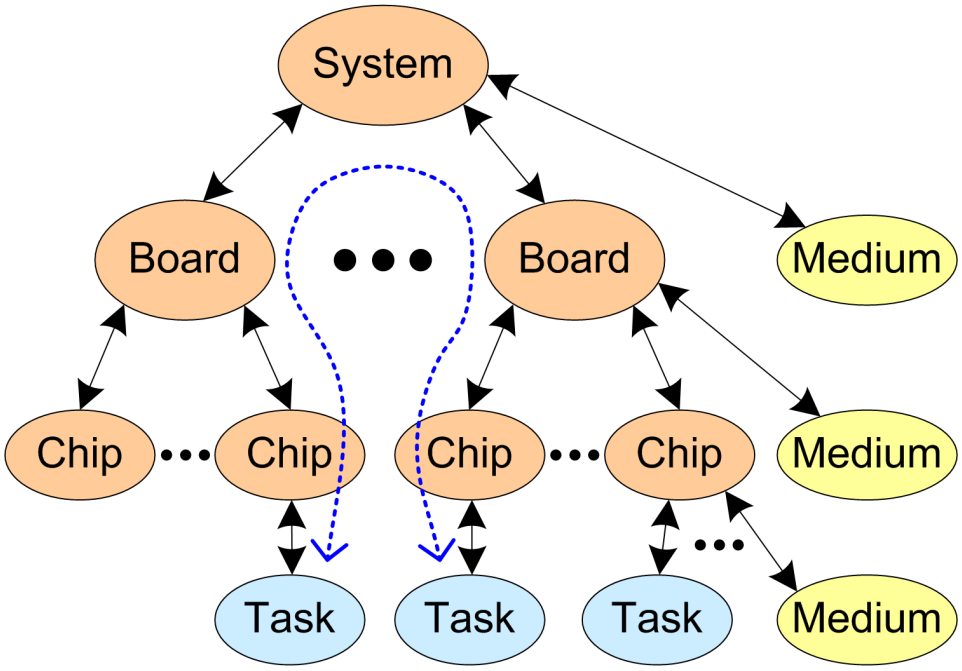

Abbildung 3.1: IFS System-Architektur

Systemarchitektur wurde für die Entwicklung komplexer Kommunikationsprototypen erdacht. Wie in Abbildung 3.1 dargestellt, wird eine hierarchische Struktur aus folgenden *System-Komponenten* zusammengestellt: *System*, *Board*, *Chip*, *Task* und *Medium*.

Die Verbindungen zwischen den Komponenten repräsentieren elektrische Verbindungen. Die System-Komponenten sind unterteilt in *Architektur-Komponenten* (System, Board, Chip) und *Kommunikations-Komponenten* (Task, Medium). Im Unterschied zu Medien, sind Tasks darauf beschränkt ausschließlich auf einem Chip positioniert zu sein. Beide Arten von Komponenten verfügen über Schnittstellenbeschreibungen, aber nur die Kommunikations-Komponenten besitzen Verhaltensbeschreibungen in Form von Protokoll-Zustands-Automaten. Zwei kompatible Tasks (oder Medien) können über die vorhandenen Verbindungen innerhalb der System-Architektur kommunizieren. Inkompatible Kommunikations-Komponenten hingegen werden über einen so genannten Interface Block (IFB) verbunden.

Grundlage für den Entwurfsprozess nach dem IFS-Flow bildet die abstrakte Beschreibung der Systemarchitektur. Sie wird als Menge von IPs beschrieben. Das Format für diese Beschreibungen ist das so genannte Interface Synthesis Format. Dieses Format beschreibt als XML-Schema die physikalische Struktur, die Eigenschaften und die Protokolle aller Komponenten. Mit diesen Informationen wird die automatische Synthese ermöglicht.

Für die Synthese werden zwei Arten von Beschreibungen benötigt: TPD und IFD. Die

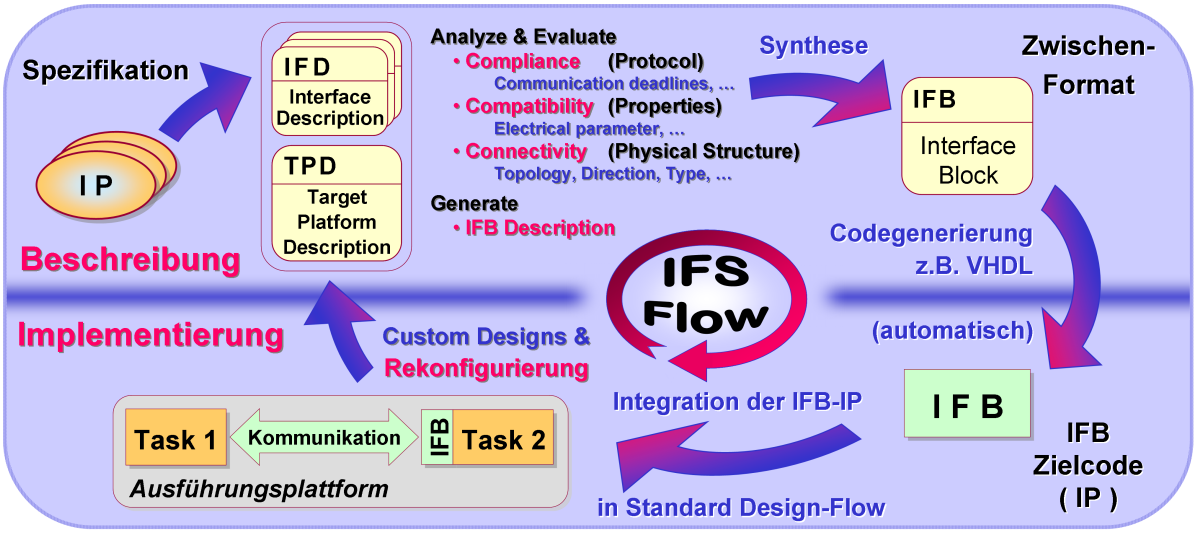

Abbildung 3.2: IFS-Flow [Ihm04]

Beschreibung der Zielplattform wird als *Target Plattform Descriptions (TPD)* bezeichnet. Sie beschreibt die Eigenschaften der Chips als Ausführungsplattform. Die Schnittstellen von Komponenten werden durch die *Interface Descriptions (IFD)* beschrieben. Die Beschreibung eines Systems und seiner Komponenten im IFS-Format kann mit Hilfe des Java-Werkzeuges *IFS-Editor* erzeugt werden.

Ist ein vollständiges System mit den gewünschten IPs im IFS-Format vorhanden, wählt der Entwickler die Schnittstellenbeschreibungen der Task- und Medienschnittstellen aus, welche verbunden werden sollen. Es folgt eine Analysephase, in welcher die Übereinstimmung, Kompatibilität und die Konnektivität der ausgewählten IFDs überprüft wird. Wenn alle notwendigen Bedingungen erfüllt sind, kann eine einfache, direkte Verbindung erstellt werden. Andernfalls, wenn die Protokolle inkompatibel sind, wird ein IFB synthetisiert.

Die Synthese teilt sich in zwei Phasen. In der ersten Phase wird eine abstrakte IFB Instanz in einem Zwischenformat erstellt. Dieses Zwischenformat kann als XML-Datei gespeichert, exportiert und importiert werden. In der zweiten Phase wird der endgültige Code der IFB Implementierung automatisch generiert. Die Zielsprache hängt davon ab, ob eine Hardware- oder Softwarelösung erstellt werden soll. Zur Zeit wird VHDL für die Synthese von Hardware unterstützt. Zum Schluss wird die so erzeugte IFB Instanz in die bereits existierende Implementierung der modellierten Systemarchitektur integriert.

Der IFB-Flow setzt dabei das Vorhandenensein der Implementierung einer jeden beschriebenen IP voraus. Die in dieser Arbeit vorgestellte Erweiterung des IFS-Flows ermöglicht eine automatisierte und optimierte Erstellung aller benötigten Schnittstellen in einem Design Zyklus. Dafür wird das Erstellen und automatische Verteilen von mehreren Multi-Task IFBs in den Synthese Schritten ermöglicht. Zusätzlich werden De-

dicated Protocols verwendet um den Kommunikationsoverhead zu reduzierten und strukturelle Engpässe im Kommunikations-System zu überwinden. Beide Erweiterungen des IFS-Flows ermöglichen das Beschleunigte und Optimierte Erstellen von Kommunikationssystemen.

3.2 Spezifikationsgraph

Ein Ansatz zur Modellierung auf Systemebene und Hardware / Software Codesign unter Berücksichtigung von Schnittstellen ist in [Tei97] gegeben. Das Designproblem wird durch einen Spezifikationsgraphen beschrieben. Dieser setzt sich aus einem Problem- und einem Architekturgraphen zusammen. Eine optimale Abbildung des Problemgraphen auf den Architekturgraphen ist die gesuchte Lösung.

Ein Problemgraph setzt sich aus Task-Knoten und Kommunikationsknoten zusammen, die zum Verbinden der Tasks genutzt werden. Die Task-Knoten müssen auf die Architekturkomponenten abgebildet werden, welche die ausführenden Plattformen innerhalb des Architekturgraphen darstellen.

Die Kommunikationsknoten hingegen müssen auf die Architekturkomponenten abgebildet werden, welche die Kommunikationskanäle darstellen. Dies ist ein kritischer Aspekt, da der Entwickler sicherstellen muss, dass die verwendeten Kommunikationskanäle zu den angebundenen Tasks, in Hinblick auf das Medium, das Protokoll und den benötigten Datendurchsatz passen.

3.3 Kommunikationsgraph

Die automatische Interface Synthese in Verbindung mit konfigurierbarer Hardware ermöglicht die Definition von Full Custom Interfaces welche auf der Zielplattform implementiert werden müssen. Full Custom Interfaces entstehen dadurch, dass nur die tatsächlich benötigten Teile der Schnittstellen in der Synthese beachtet werden. Welche Teile das sind, kann im IFD-Mapping spezifiziert werden. Der IFB ist als Kommunikationspartner für die anderen Komponenten transparent. Daher reduzieren sich die Kommunikationskanäle des Architekturgraphen auf einfache Verbindungen. Dies ermöglicht einen hohen Grad an Flexibilität in der Konstruktion von Kommunikationssystemen.

Der in Abbildung 3.3 dargestellte *Kommunikationsgraph* wurde durch die Verbindung der *IFS Systemarchitektur* mit dem Spezifikationsgraph Modell erzeugt. Der Architekturgraph leitet sich von den *Architekturkomponenten* der System-Architektur ab (siehe auch Abbildung 3.1).

Alle Task-Knoten (hier: $T_1, ..., T_5$) müssen auf die vorhandenen Chips abgebildet werden. Im Unterschied zum Spezifikationsgraph werden die Kommunikationsknoten auf IFBs

abgebildet. Mindestens ein (Multi-Task) IFB wird benötigt, um eine Menge von inkompatiblen Tasks zu verbinden. Die IFBs werden zur Design-Zeit angelegt und danach synthetisiert, wie im IFS-Flow beschrieben. Im Unterschied zum Spezifikationsgraphen ist der IFB, welcher als Umsetzung eines Kommunikationsknoten gesehen werden kann, an die Implementations-Plattform (Chip) gebunden.

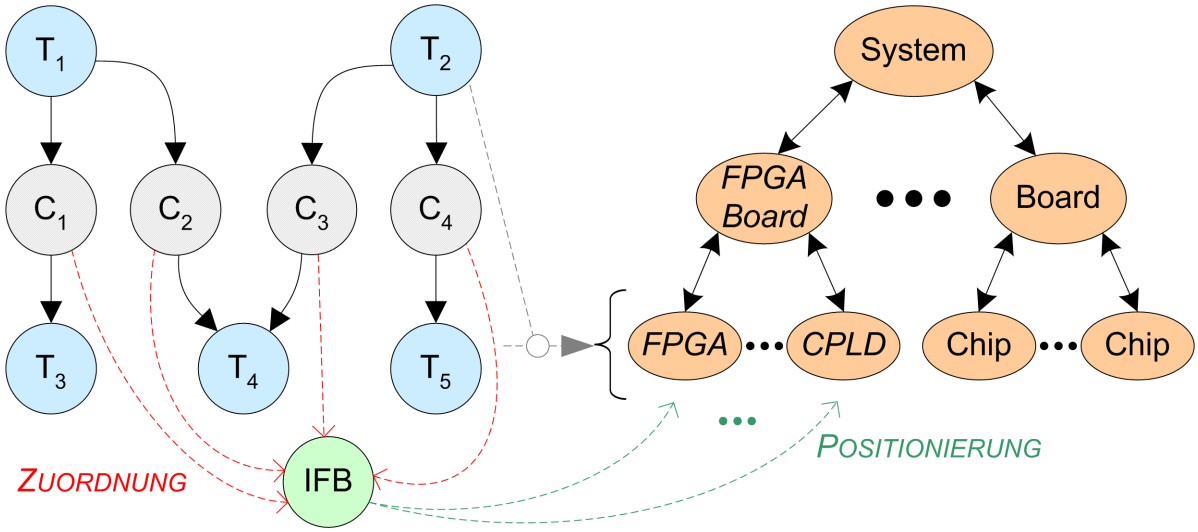

Abbildung 3.3: Kommunikationsgraph

3.4 Das IFD-Mapping

Jeder *Kommunikationsknoten* (hier:$C_1, ..., C_4$) repräsentiert ein *IFD-Mapping*. Ein IFD-Mapping besteht aus Mapping Equations und spezifiziert die Umsetzung der Informationen von eingehenden zu ausgehenden Daten innerhalb eines IFB. Die *IFD-Mapping Sprache* wurde definiert, um Mapping Equations $MapEqn$ der folgenden Form zu beschreiben:

$$MapEqn : data_{Out} <= f_{Map}(data_{In,1}, ..., data_{In,k});$$

Ein $data_{Out}$-Paket kann von mehreren $data_{In}$-Paketen abhängig sein. Um diese Abhängigkeiten zu definieren bietet die *IFD-Mapping Sprache* vier Grundoperationen. Diese Operationen können auch kombiniert werden.

1. Zuweisung konstanter Werte
2. Zuweisung umsortierter eingehender Bits

3. Anwenden von boolschen Funktionen auf eingehende Bits
4. Verwenden eines endlichen Zustandsautomaten (FSM) zum Erzeugen von Bits

Die Datenpaket, welche sich aus einem Protokoll extrahieren lassen, können als Matrix repräsentiert werden. Daher wurde die Notation für das Umsortieren von Bits oder das Zuweisen von konstanten Werten an die Matrix Notation von Matlab angelehnt.

$$\text{z.B.: } P_{Out}[1..2] <= P_{In}[2..1];$$

Für komplexere Funktionen, wie verknüpfte boolsche Funktionen oder die Verwendung einer FSM, wurden weitere Konstrukte in die IFD-Mapping Sprache aufgenommen.

$$P_{Out} <= \text{ if } (P_{In}[0] =' 0')\{P_{In}\} \text{ else } \{''0110''\};$$

Um mehrere IFD-Mappings verschiedener Tasks in einem IFB zu bearbeiten, kann ein *Multi-Task IFB* konstruiert werden. Das Zuordnen (Binding) eines IFD-Mappings zu einem IFB bedeutet das Implementieren des IFD-Mappings im IFB. Um das Verteilen von IFD-Mappings, unter Berücksichtigung einer gegebenen System-Architektur, weiter zu erörtern ist ein Grundverständnis der der internen IFB Struktur erforderlich.

3.5 Der Aufbau des Interface Blocks (IFB)

Ein Interface Block (IFB) ist ein Adapter Modul, das durch den IFS-Flow dynamisch erzeugt wird. Der IFB dient als Schnittstelle zwischen Kommunikationskomponenten. Durch die so erzeugte Schnittstelle können IPs integriert werden, ohne Änderungen an den IPs selbst vornehmen zu müssen. Der IFB erlaubt dabei in Zusammenspiel mit dem IFD-Mapping auch das Einbinden inkompatibler IPs, da das IFD-Mapping Funktionen bietet, welche über einfaches Abbilden von Pins hinausgehen.

Für das Erzeugen eines IFB werden die Schnittstellenbeschreibungen der verbundenen Tasks [Ihm03b] zusammen mit dem IFD-Mapping ausgewertet und die benötigen Komponenten modular synthetisiert. Ein IFB realisiert die Umsetzung der Mapping Equations. Mapping Equations ermöglichen die Umwandlung von Datenpakten die zwischen inkompatiblen Protokollen übertragen werden. Über einen IFB wird daher das Verbinden von Komponenten mit unterschiedlichen Protokollen ermöglicht.

Das ist einer der größten Vorteile des IFS-Flows. Die Erzeugung von IFBs ist daher ein zentraler Aspekt des IFS-Flows [Ihm03a, Ihm02b, Ihm02a]. Die Kommunikationsstruktur im Inneren eines IFB wird als *IFB-Makrostruktur* bezeichnet. Die IFB-Makrostruktur wird durch eine Reihe von endlichen Zustandsautomaten (*Finit State Machine = FSM*) realisiert. Die einzelnen Komponenten dieser Struktur werden im Weiteren detailierter beschrieben.

3.5.1 Control Unit (CU)

Die Struktur des IFB unterteilt sich in drei Komponenten: *Protocol Handler* (PH), *Sequence Handler* (SH) und *Control Unit* (CU) (siehe Abbildung 3.4).

Die CU regelt die interne Steuerung zwischen den einzelnen Komponenten der Makrostruktur des IFB. Dazu sendet die CU Steuersignale zum PH und SH und wertet deren Statussignale aus. In einem Multi-Task IFB übernimmt die CU auch das Scheduling des Zugriffs einzelner Komponenten auf die IFB internen Ressourcen.

Die CU beinhaltet zwei Scheduler. Einer steuert, welcher PH-Modus gerade Daten empfangen darf. Dazu gibt er einen entsprechenden Bus im IFB frei, über den eingehende Daten von einem PH-Modus in den Speicher für eingehende Daten geschrieben werden. Der andere Scheduler entscheidet welcher PH-Modus Daten versenden darf. Dazu wird ebefalls ein entsprechender Zugriff auf den Bus verwaltet, der mit dem Speicherbereich für die ausgehenden Daten verbunden ist. Als Scheduling Verfahren können verschiedene Varianten verwendet werden.

Ein Scorebord stellt sicher, dass die Kausalität während der Ausführung des *Eingabe-Verarbeitung-Ausgabe (EVA)* Zyklus erhalten bleibt. Der EVA Zyklus in einem IFB läuft nach dem Prinzip des Pipelining ab.

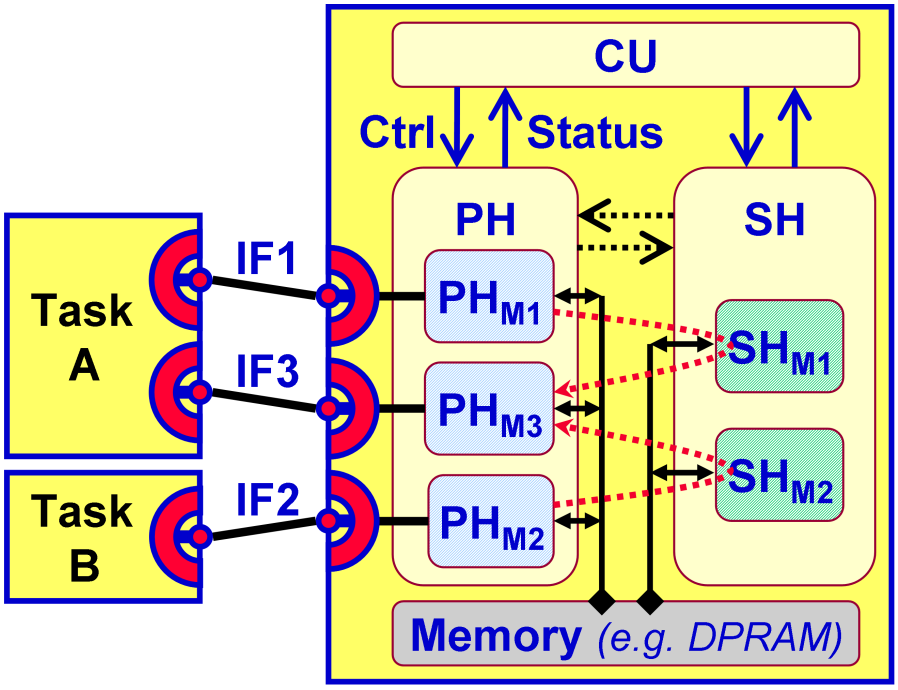

Abbildung 3.4: IFB-Makrostruktur

3.5.2 Protocol Handler (PH)

Sequence und Protocol Handler sind modular aus so genannten Modi aufgebaut. Diese Modi realisieren Funktionen und sind jeweils durch eine FSM beschrieben. Der Protocol Handler besitzt einen Modus pro Kommunikationverbindung mit einer angebundenen Task. Der PH realisiert eine oder mehrere Schnittstellen zu den Kommunikationspartnern, die über einen IFB verbunden sind. Über diese Schnittstellen laufen die ein- und ausgehenden Daten. Für jede verbundene Schnittstelle einer Task wird ein Protocol Hander Mode PH_M im PH benötigt, der das entsprechende Protokoll der angeschlossenen Task verarbeitet.

Ein PH_M implementiert dazu das komplementäre Protokoll aus einer Task, welches in der dazugehörigen IFD beschrieben ist. Hierfür wird die Protokoll FSM von Task t aus der IFD extrahiert und eine FSM generiert, welche ein komplementäres Protokoll realisiert. Diese FSM wird um zusätzliche Zustände erweitert. Mit diesen Zuständen wird die Interaktion mit der Control Unit und dem Sequence Handler ermöglicht. Ein PH_M übernimmt das Einfügen und Extrahieren von Daten eines Protokolls gemäß der Mapping Equation des Protokolls. Alle nicht im IFD-Mapping vorkommenden Daten werden ignoriert. Die Rohdaten werden in einem Speicherelement des Sequence Handlers zwischengespeichert.

3.5.3 Sequence Handler (SH)

Der Sequence Handler beinhaltet eine Menge von Modi, welche die Umsetzung des IFD-Mappings realisieren. Für jede Mapping Equation in einem SH existiert eine Sequence Handler Mode (SH_{M_x}). Die Rohdaten im Speicherelement werden durch den SH_M modifiziert. Dazu werden die gespeicherten Daten umsortiert oder modifiziert, so wie es im IFD-Mapping spezifiziert ist. Dabei sind nicht nur strukturelle, sondern auch semantische Änderungen der Daten möglich. Für die Kommunikation zwischen unterschiedlichen Protokollen ist das erforderlich. Die modifizierten Daten werden an einer anderen Stelle im Speicherelement abgelegt. Ist ein Datenpaket vollständig, werden die Daten im ausgehenden Protokoll durch einen zugehörigen PH_M zusammengefügt.

Ein Hauptvorteil des Modularen Aufbaus des IFB, ist das Umsetzen der vollständigen Schnittstellen Funktionalität, ohne dass der Produkt-Automat aller Ursprungs- und Ziel-FSMs erstellt werden muss. Die Verwendung von Modi erlaubt eine einfache Anpassung und Erweiterung eines IFB. Sogar eine Rekonfiguration zur Laufzeit ist möglich [Ihm04].

Die Verwendung von IFD-Mappings in Verbindung mit den SH Modes erlauben eine maximale Flexibilität im Datenaustausch. Ein automatisches IFD-Mapping zwischen den empfangene und gesandten Daten ist unmöglich ohne vollständiges Verständnis der Semantik der ausgetauschten Informationen. Die Verwendung von IPs unterschiedlicher Hersteller unter Annahme einer globalen Semantik für Daten ist illusorisch. Daher sind IFD-Mappings unverzichtbar.

4 Automatische Verteilung

Gemäß des in 3.3 vorgestellten Modells des Kommunikationsgraphen, bedeutet das Erstellen eines Kommunikationsstruktur das Abbilden von Kommunikationsknoten auf IFBs. Die IFBs wiederum müssen ebenfalls auf Chips in der Systemarchitektur positioniert und realisiert werden.

In diesem Kapitel wird daher zuerst betrachtet, welche Auswirkungen die Position eines IFBs auf die Ressourcen der Systemarchitektur hat. Dann werden entsprechende Kostenfunktionen vorgestellt, um die Art und Größe der benötigen Ressourcen für die Realisierung einer Kommunikationsstruktur zu bestimmen. Dann folgt die Vorstellung der einzelnen Abschnitte eines Verfahren, mit dem die benötigte Verteilung von IFBs auf Chips automatisiert werden kann.

4.1 Systemtopologien unter Verwendung von IFBs

Verschiedene Anzahlen von IFBs und verschiedene Positionierungen innerhalb einer Systemtopologie haben Auswirkungen auf die Kommunikationsverbindungen. Welche Auswirkungen das sind und welche Systemressourcen davon betroffen sind, wird in den beiden folgenden Abschnitten an einem Beispiel erläutert.

4.1.1 Verwendung eines einzelnen IFBs

Im IFS-Flow wird eine Menge von interagierenden Kommunikationskomponenten bestimmt. Wenn diese inkompatibel sind und über einen IFB kommunizieren sollen, wird die Position des IFB auf einem Chip in der Systemarchitektur vom Entwickler festgelegt. Von allen Kommunikationskomponenten müssen nun entsprechende Verbindungen vom Chip auf dem sie realisiert sind bis zum IFB gefunden werden (routing). Die Menge und Länge der benötigten Verbindungen hängt stark von der Postion des IFBs innerhalb der Systemtopologie ab.

In Abbildung 4.1 ist eine Systemarchitektur zu sehen, in der auf dem Chip 3 ein IFB positioniert wurde. Die Pfeile, welche direkt Tasks verbinden, geben an welche Komponenten miteinander kommunizieren wollen. Durch die Systemarchitektur verbinden Pfeile IFBs und Tasks. Diese visualisieren die Anzahl und Länge der benötigen Verbindungen. Die Breite (Anzahl Bits) dieser Verbindungen ist nicht angegeben.

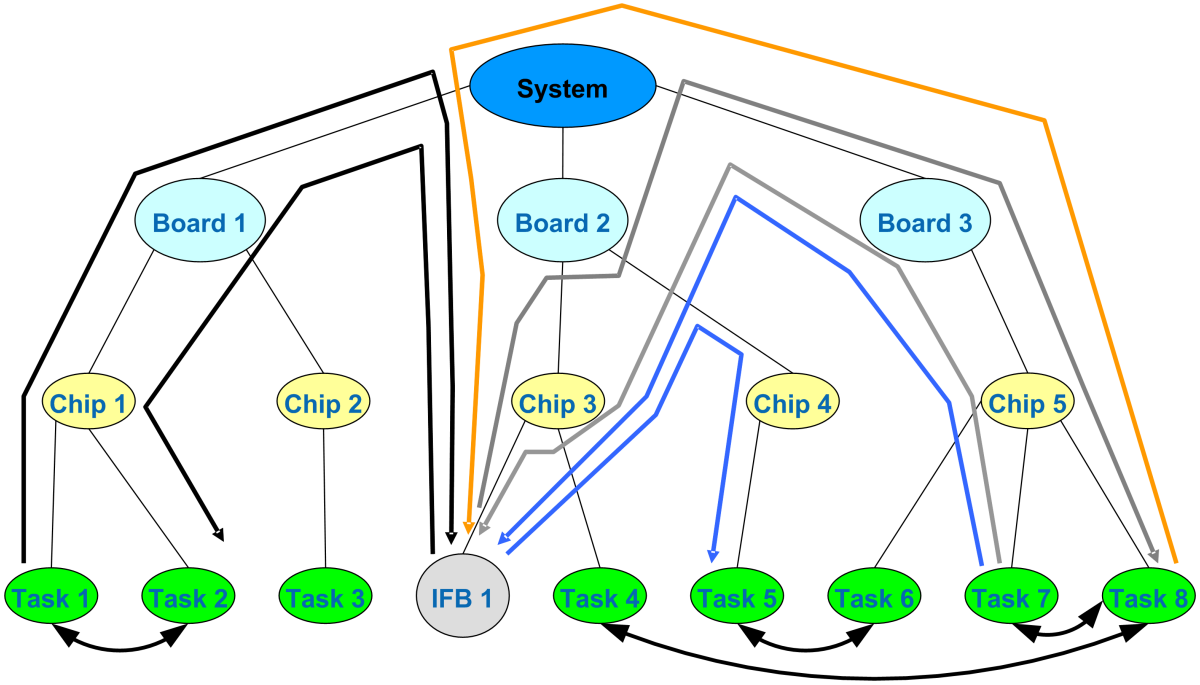

Abbildung 4.1: Systemtopologie unter Verwendung eines einzelnen IFBs

Wird der IFB auf einem Chip positioniert, auf dem sich Komponenten befinden, die über den IFB kommunizieren sollen, so müssen für diese Komponenten keine Verbindungen durch die Systemarchitektur bereitgestellt werden. Es müssen nur ausreichend lokale Verbindungsmöglichkeiten zwischen den Schnittstellen des IFBs und den Kommunikationskomponenten bestehen. Positioniert man hingegen für die gleiche Menge an Kommunikationskomponenten den IFB auf einem Chip im System auf dem sich keine Komponenten befinden, die über den IFB kommunizieren sollen, so ist die Anzahl und Länge der benötigten Kommunikationsverbindungen höher.

Die Anzahl der zur Verfügung stehenden Verbindungen zwischen den Komponenten der Systemarchitektur ist begrenzt. Kann nicht von allen Kommunikationskomponenten eine Verbindung zum IFB hergestellt werden, so kann die Kommunikationstruktur nicht umgesetzt werden. Durch eine gezielte Positionierung eines IFBs innerhalb der Systemtopologie werden weniger Verbindungen durch das System benötigt, und es lassen sich komplexere Kommunikationsstrukturen realisieren.

4.1.2 Verwendung mehrerer IFBs

Die gezielte Positionierung eines einzelnen IFBs kann die Anzahl der benötigten Verbindungen zwar reduzieren, allerdings nur für eine Teilmenge der interagierenden Kommunikationskomponenten. Die Anzahl der benötigten Verbindungen kann weiter minimiert

werden, indem mehrere IFBs verwendet werden. Über einen IFB kommuniziert dabei jeweils eine Teilmenge der Komponenten der Kommunikationsstruktur, welche alle untereinander interagieren. Ein Beispiel ist in Abbildung 4.2 zu sehen. Die interagierenden Komponenten sind die gleichen wie in Abbildung 4.1, aber die Anzahl der benötigten Verbindungen konnte stark verringert werden.

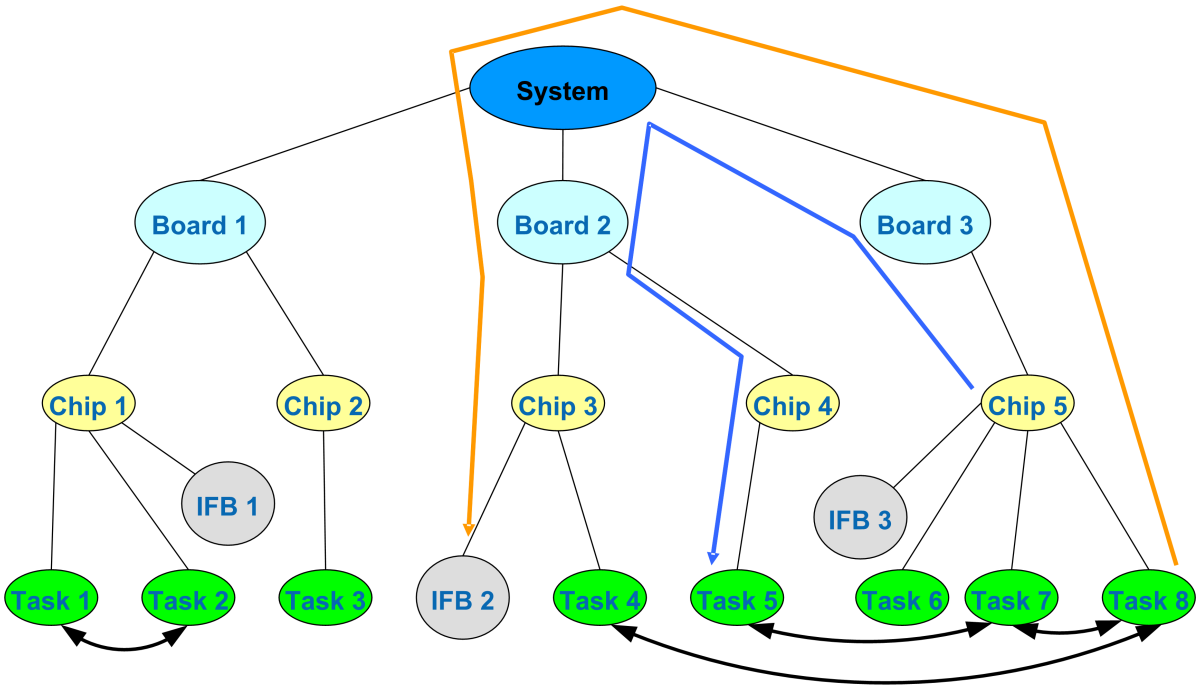

Abbildung 4.2: Systemtopologie unter Verwendung mehrerer IFBs

4.2 Automatische Verteilung von IFBs

Die automatische Verteilung von IFBs geht von einer festen Zuordnung von Tasks zu Chips und einer statischen Zusammenstellung von Kommunikationspartnern aus. Die zu erstellende Kommunikationsstruktur ist physikalischen Einschränkungen unterworfen, da die Systemarchitektur nur über eine begrenzte Anzahl von elektrischen Leitungen verfügt. Die verschiedenen Möglichkeiten der Zuordnung von IFD-Mappings zu IFBs, die Positionierung der IFBs auf einem der Chips der Systemarchitektur und die daraus resultierende Reservierung der notwendigen Verbindungen legen den Lösungsraum für ein Design der Kommunikationstruktur fest.

Die Position eines IFBs innerhalb der Systemarchitektur bestimmt im hohen Maße die Anzahl der benötigten Verbindungen. Um eine komplexe Kommunikationsstruktur zu realisieren ist eine möglichst effiziente Ausnutzung der gegebenen Ressourcen und daher

eine optimale Positionierung der IFBs in der Systemarchitektur erforderlich. Dieses Ziel wird erreicht, indem eine Kostenfunktion für Verbindungs- und Implementierungkosten eines IFBs aufgestellt wird. Anschließend wird ein Verfahren zur Minimierung dieser Kosten angegeben.

4.2.1 Kostenfunktionen

Die Gesamtkosten der Kommunikationsstrukturen in einer System-Architektur sind definiert als Summe aller Kommunkations- und Implementierungskosten. Diese Kosten werden von nun an als $C_{SysArch}$ bezeichnet. Es gilt $C_{SysArch}$ zu minimieren, um das Erstellen einer Kommunikationsstruktur unter Berücksichtigung der gegebenen System-Architektur zu ermöglichen.

$$C_{SysArch} = C_{Tasks} + C_{StaticIO} + \sum_{i=1}^{k} C(IFB_i) \tag{4.1}$$

Die Zahl k ist die Anzahl der verwendeten IFBs und entspricht dem k aus der Gleichung (2.4.2). Für die Synthese der Tasks und die Allokation von direkten Verbindungen entstehen statische Kosten, welche aber für die Optimierung der Gesamtkosten nicht berücksichtigt werden müssen. Nur die Kosten, welche bei der Umsetzung von IFBs anfallen, sind variabel. Im folgenden wird zwischen zwei Arten von Kosten unterschieden:

1. Ein IFB benötigt Ressourcen in Form von konfigurierbaren Logikblöcken (CLB) oder Speicherplatzt für seine Implementierung ($\Rightarrow$ *Implementierungskosten* C_{Impl}).

2. Zum Aufbau der Kommunikation zwischen dem IFB und den verbundenen Tasks werden Leitungen benötigt, um diese Verbindungen zu ermöglichen ($\Rightarrow$ *Verbindungskosten* $C_{Connect}$).

$$C(IFB_i) = C_{Impl}(IFB_i) + C_{Connect}(IFB_i) \tag{4.2}$$

Die Höhe der einzelnen Kosten ist von der Anzahl der IFD-Mappings, welche in einem IFB realisiert werden, abhängig. Für die weitere Erläuterung von C_{Impl} und $C_{Connect}$, gilt für einen IFB_i:

$$n = \#\text{IFD-Mapping zugewiesen zu } IFB_i \tag{4.3}$$

Implementierungskosten

Die *Implementierungskosten* C_{Impl} sind wie folgt definiert:

$$C_{Impl}(IFB_i) = (1 + n \cdot IFB_{MappingKosten}) \cdot IFB_{Struktur} + \sum_{x=1}^{\#PH_{Modes}} C(PH_{M_x}) + \sum_{x=1}^{\#SH_{Modes}} C(SH_{M_x}) \tag{4.4}$$

Der erste Summand der Gleichung beschreibt die Kosten, welche für die Implementierung der IFB Grundstruktur anfallen. Dazu gehören die CU, der Speicher für die Zwischenspeicherung der Datenpakete gemäß dem IFD-Mapping sowie Bussysysteme und Steuerleitungen für interne Signale. Für jedes IFD-Mapping werden eine erweiterte Kontrollstrukutr, weiterer Speicherplatz und zusätzliche Steuerleitungen benötigt.

Die Kosten erhöhen sich um den Faktor $IFB_{MappingKosten}$, welcher kleiner eins ist. Ist kein IFD-Mapping vorhanden ($n = 0$), so ist dieser Teil dennoch größer Null, da auch ohne IFD-Mappings Grundkosten für die CU anfallen. In diesem Fall kann die Implementierung des IFB jedoch entfallen. Der zweite Teil der Gleichung summiert die Kosten aller benötigten PH und SH Modes auf.

Verbindungskosten

Die *Verbindungskosten* $C_{Connect}$ sind wie folgt definiert:

$$C_{Connect}(IFB_i) = \sum_{m=1}^{n} \#conductions_m \cdot Länge_m \tag{4.5}$$

Die Anzahl der Kanten auf dem Pfad durch den Systemarchitekturgraphen ergibt die Länge einer Verbindung m ($Länge_m$). Für die Berechnung der Verbindungskosten könnten auch weitere Eigenschaften, welche in der IFD zur Verfügung stehen, in die Gleichung eingehen. So können bestimmte Konstellationen bevorzugt werden. Beispiele für diese Eigenschaften sind z.B. die Frequenz oder der Wert für den Jitter. Diese Eigenschaften wurden aber im Rahmen dieser Arbeit nicht weiter betrachtet.

Mit Hilfe der angegebenen Gleichungen können die Kosten $C(IFB)$ für einen IFB bestimmt werden.

4.2.2 Strategien für die Verteilung von IFBs

In heutigen Chips und FPGAs stehen eine riesige Menge an Transistoren und anderer Implementierungseinheiten zur Verfügung. Die Ressourcen der Kommunikation wurden im Laufe der Zeit immer mehr ein begrenzender Aspekt des SoC-Design (System-on-Chip).

Bei Entwicklungen auf FPGAs stehen innerhalb eines FPGAs nur eine sehr begrenzte Anzahl von langen Leitungen, welche über die gesammte Ausdehnung der FPGA Struktur laufen, zur Verfügung, zur Verfügung. Die I/O Pins für die off-chip Kommunikation eines FPGAs sind ebenfalls eine begrenzte Ressource. Besonders Kommunikationsverbindungen off-chip durch die System-Architektur sind daher extrem kostenintensiv und müssen somit minimiert werden.

Die Reduzierung der Verbindungskosten ist daher ein Hauptziel bei der Verteilung von IFBs. Reduzierte Verbindungskosten führen jedoch zu einer erhöhten Anzahl von IFBs, was wiederum in höheren Implementierungskosten resultiert. Abhängig von der gegebenen Systemarchitektur und dem aktuellen Designziel können verschiedene Strategien der Verteilung angemessen sein, wie in Abbildung 4.3 dargestellt wird.

1. **Minimale Verbindungskosten**
   Für jede Kommunikation zwischen einer Menge von interagierenden Tasks wird ein IFB zur Verfügung gestellt und so auf einem der Chips auf dem sich die Tasks befinden positioniert, dass die Verbindungskosten minimal sind. Diese Strategie wird den IFB auf dem Chip platzieren, auf welchem sich die Task mit der höchsten benötigten Anzahl an Verbindungen befindet.

   Der Nachteil dieser Strategie ist, dass jeder Menge kommunizierender Tasks ein IFB zur Verfügung gestellt wird und daher die Implementierungskosten sehr hoch sind.

2. **Minimale Implementierungskosten**
   Die Anzahl der IFBs wird auf ein Minimum reduziert, und die Anzahl der IFD-Mappings pro IFB wird maximiert. Die Auslastung der übrig bleibenden IFBs wird dadurch erhöht und die Implementierungskosten auf ein Minimum reduziert. Der Nachteil dieser Strategie ist, dass sie zu sehr hohen Verbindungskosten führen kann.

3. **Obere Schranke für die Anzahl an IFBs**
   Diese Strategie optimiert die Verbindungskosten unter der Berücksichtigung einer gegebenen oberen Schranke für die Anzahl der IFBs.

4. **Obere Schranke für die Implementierungskosten eines IFB**
   Diese Strategie begrenzt die lokalen Implementierungskosten durch das Setzen einer oberen Schranke für die Implementierungskosten eines IFBs. Überschreitet ein IFB diese Schranke werden IFD-Mappings aus dem IFB entfernt und in einem anderen IFB realisiert, in dem noch ausreichend Ressourcen zur Verfügung stehen.

5. **Einhalten einer Ausgeglichenheitsfunktion**
   Mit Hilfe einer Ausgeglichenheitsfunktion, welche von jedem IFB des Systems erfüllt sein muss, werden extreme Werte für Verbindungs- und Implementierungskosten vermieden. Ein wichtiges Ziel, welches mit dieser Strategie erreicht werden

kann, ist eine Mindestauslastung aller IFBs. Dazu kann man eine Ausgeglichenheitsfunktion angeben, welche eine Mindestanzahl von IFD-Mappings pro IFB fordert. Sind in einem IFB nicht ausreichend viele IFD-Mappings realisiert, so werden die noch vorhandenen IFD-Mappings entfernt und in benachbarte IFBs verschoben. Durch diese Strategie wird z.B. das Verwenden eines IFB für nur ein einziges IFD-Mapping vermieden.

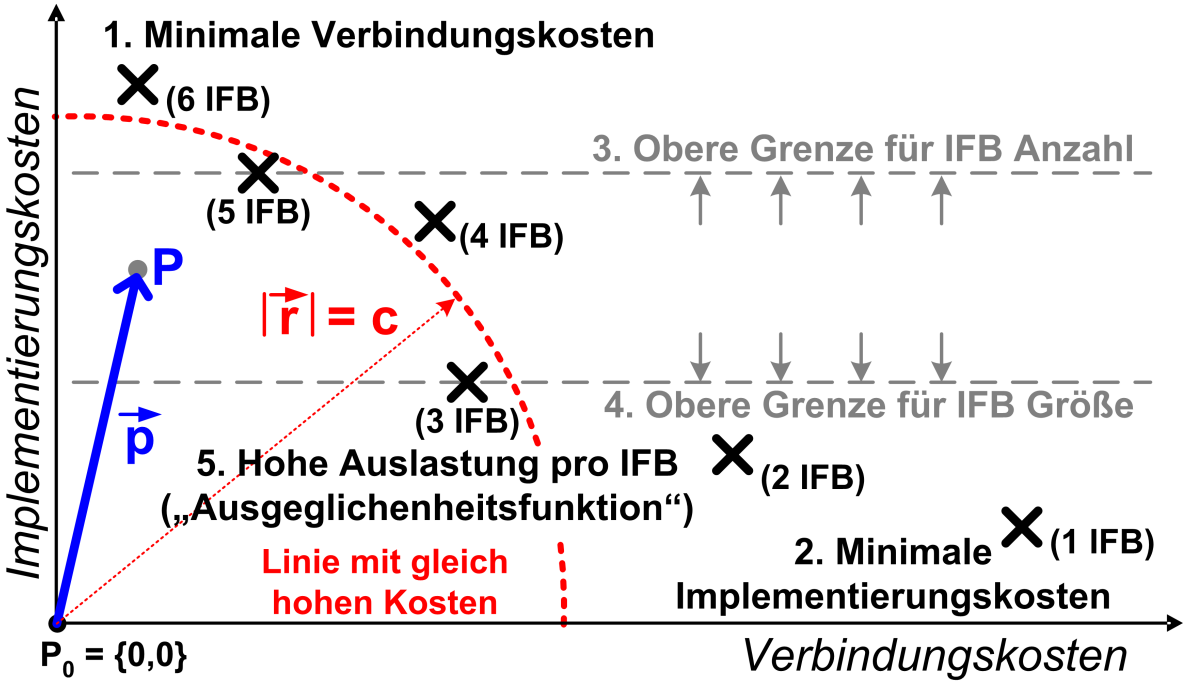

Abbildung 4.3: Strategien für die Verteilung von IFBs

Jedes Kreuz in der Abbildung 4.3 repräsentiert eine mögliche pareto optimale Lösung für eine feste Anzahl an IFBs. Der Kreisbogen definiert die Linie auf welcher die Gesamtkosten c als Summe von Verbindungs- und Implementierungskosten entsprechend der Gleichung (4.1) gleich sind (Äquipotentiallinie). Der Absolutwert $|\overrightarrow{p}|$ definiert die Kosten einer Lösung, welche für den Punkt P des Lösungsraumes anfallen. Die Optimierungsstrategien können in die Kostenfunktion integriert werden. Dazu werden die Quantoren a und b in die Terme der Implementations- und Verbindungskosten der Gleichung (4.2) eingefügt. Die Gesamtkosten werden dadurch zu einer gewichteten Summe, die sich je nach Wahl der Quantoren an die Strategie 1-5 anpassen lässt.

$$C(IFB_i) = a \cdot C_{Impl}(IFB_i) + b \cdot C_{Connect}(IFB_i) \tag{4.6}$$

4.2.3 Clustering als Verfahren zur Reduktion der IFB-Anzahl

Das Problem der Verteilung einer Anzahl von IFBs über die Systemarchitektur und die Entscheidung welches IFD-Mapping in welchem IFB implementiert wird, kann abstrakt als Partitionierendes Clustering Problem aufgefasst werden.

Die Anzahl der möglichen Aufteilungen von n Elementen auf k Cluster wird durch die Gleichung (2.4.2) beschrieben. Alle möglichen Aufteilungen zu berechnen und zu bewerten ist meist nicht möglich, da die Anzahl der möglichen Aufteilungen sehr hoch sein kann. Das Berechnen aller Möglichkeiten wird noch zusätzlich dadurch erschwert, dass die Anzahl der Cluster k in dem hier betrachteten Ansatz Teil des Ergebnisses ist.

Es wird ein partitionierendes bottom-up Clustering verwendet, um dieses Problem zu umgehen. Der Algorithmus, welcher dieses Clustering realisiert, verwendet eine Heuristik, so dass nur ein kleine Anzahl aller möglichen Aufteilungen betrachtet wird. Das Ergebnis ist eine Anzahl disjunkter Cluster ohne eine Hierarchie. Jedes Cluster wird durch einen IFB repräsentiert. Die auf die Cluster verteilten Elemente sind die IFD-Mapping-Gleichungen welche in den IFBs implementiert werden.

4.3 Ablauf der Verteilung von IFBs

Nachdem in den vorangehenden Abschnitten Aspekte wie Strategien, Ziele und Kosten für die automatische Verteilung von IFBs erläutert wurden, wird nun der Ablauf des Verfahrens selbst dargestellt.

4.3.1 Erstellen einer initialen Verteilung

Der heuristische Algorithmus beginnt mit dem Erstellen einer initialen Verteilung der IFBs mit einer entsprechenden Zuordnung der IFD-Mappings. Ein IFD-Mapping wird genommen und die Chips der Sender- und Empfängerseite der im IFD-Mapping aufgeführten Tasks bestimmt. Welche das sind, geht aus den Angaben des IFD-Mappings hervor. Ein IFB (= Cluster) wird auf dem Chip platziert, welcher die Empfänger Task eines IFD-Mappings enthält.

Der IFB wird nur auf dem Chip platziert, wenn die Verbindungs- und Implementationskosten die vorhandenen Ressourcen nicht überschreiten. Wenn das Platzieren auf der Empfängerseite nicht möglich ist, wird der IFB zur Senderseite migriert. Sollte das ebenfalls nicht möglich sein, wird der IFB nicht platziert und das IFD-Mapping, nach Verteilung aller anderen IFBs, im nächstgelegenen IFB implementiert.

Welcher IFB der nächstgelegene ist kann mit den Verbindungskosten, welche nur für diese eine IFD-Mapping enstehen, bestimmt werden. Für ein IFD-Mapping m sind die

Verbindungskosten für die Realisierung im IFB i:

$$C_{Connect}(IFB_i) = \#conductions_m \cdot Länge_m \tag{4.7}$$

Der IFB, beim dem die geringsten Verbindungskosten entstehen, ist der nächstgelegene IFB.

Wenn bei der Initialisierung zwei IFBs auf dem selben Chip platziert werden sollten, werden sie zu einem einzigen IFB zusammengeführt, indem alle IFD-Mappings in einem IFB zusammengefasst werden. Dies wird aber nur vorgenommen, wenn die Schedulability Analyse für diesen zusammengefassten IFB erfolgreich ist. Die Schedulability Analyse ist Teil des IFS-Design-Flows.

Das so erzeugte initiale Cluster und die weiteren Optimierungsschritte führen dazu, dass ein Großteil des Lösungsraumes für die Verteilung von IFBs nicht betrachtet wird.

4.3.2 Optimierung

Nach Erstellen der initialen Verteilung wird eine Optimierung vorgenommen, um die Verbindungskosten zu minimieren. Ein IFD-Mapping welches nur Tasks verbindet, die auf dem selben Chip positioniert sind, wird als *lokales IFD-Mapping* bezeichnet.

Lokale IFD-Mappings erzeugen keine Verbindungskosten für den IFB, da die Länge der Verbindungen gleich Null ist. Das Migrieren eines lokalen IFD-Mappings zu einen anderen IFB auf einem anderen Chip resultiert immer in einem Anstieg der Verbindungskosten. Daher werden bei der Optimierung der initialen Verteilung von IFBs nur IFBs berücksichtigt, welche nur nicht lokale IFD-Mappings beinhalten.

Bei solchen IFBs wird versucht alle beinhalteten IFD-Mappings in andere IFBs zu migrieren. Diese Migration wird nur durchgeführt, wenn die Verbindungskosten dadurch nicht erhöht werden. Wenn auf diese Weise alle IFD-Mappings aus einem IFB entfernt werden konnten, wird der IFB aus dem System entfernt.

Nach dieser Optimierung erfüllt die so erreichte Verteilung die Zielsetzung der Strategie „*Minimale Verbindungskosten*“.

4.3.3 Auswahl der Strategie

Nach der Optimierung wird die Strategie für den weiteren Ablauf des Clustering-Algorithmus ausgewählt. Die Strategie beeinflusst den Clustering-Algorithmus in Bezug auf die Modifikation der Kostenfunktion wie in Abschnitt 4.2.2 beschrieben. Die gewählte Strategie hat auch Auswirkungen auf die verwendete Abbruchbedingung des Algorithmus.

4.3.4 Clustering Algorithmus

Der Cluster Algorithmus versucht die Implementierungskosten der initialen Verteilung so weit zu reduzieren, bis die Vorgaben der gewählten Strategie erfüllt sind oder keine weitere Reduktion mehr möglich ist.

Zum Reduzieren der Implementierungskosten werden alle IFD-Mappings von einem ausgewählten IFB migriert. Wenn alle IFD-Mappings migriert werden konnten wird der IFB entfernt. Eine verringerte Anzahl an IFBs führt zu verringerten Implementierungskosten. Da die Tasks fest an den Chip gebunden sind, werden durch die Migration der IFD-Mappings die Verbindungskosten erhöht. Um die Verbindungskosten so wenig wie möglich zu erhöhen und um überhaupt alle IFD-Mappings eines IFB entfernen zu können, müssen verschiedene Aspekte berücksichtigt werden:

- Die Migration von lokalen IFD-Mappings zu anderen IFBs führt meistens zu einer größeren Zunahme der Verbindungskosten als das Migrieren von nicht lokalen IFD-Mappings.

- Die Implementierungskosten können nur reduziert werden, wenn alle IFD-Mappings eines IFB migriert werden können und der IFB anschließend entfernt wird.

Da die Ressourcen zur Implementation der IFD-Mappings beschränkt sind, ist es nicht immer möglich alle IFD-Mappings eines ausgewählten IFB zu migrieren. Daher ist die Wahrscheinlichkeit einen IFB mit wenigen IFD-Mappings zu leeren größer als bei einem IFB mit vielen IFD-Mappings. Für die Entscheidung von welchem IFB die IFD-Mappings migriert werden sollen, wird das Entscheidungskriterium *interner* und *externer Grad* eines IFB definiert.

Der Wert des internen Grades ergibt sich aus der Anzahl aller Verbindungen die, aufgrund von lokalen IFD-Mappings eines IFBs zwischen Tasks und PH-Modes, benötigt werden. Der externe Grad repräsentiert die Anzahl der benötigten Verbindungen für alle nicht lokalen IFD-Mappings eines IFB.

Der Algorithmus startet mit dem IFB, welcher den niedrigsten internen Grad hat. Falls es mehrere IFBs mit dem niedrigsten internen Grad gibt, wird davon der IFB mit dem niedrigsten externen Grad ausgewählt. Die Migration der IFD-Mappings dieses so ausgewählten IFBs findet unter den gleichen Bedingungen statt wie in Abschnitt 4.3.1 und 4.3.2 beschrieben.

Wenn der IFB entfernt werden konnte wird die Abbruchbedingung geprüft und der nächste IFB über den Grad bestimmt. Wenn der IFB nicht entfernt werden konnte, wird er markiert damit er nicht sofort nochmals ausgewählt werden kann. Dann wird der nächste IFB bestimmt.

4.4 Integration im IFS-Editor

Der in 3.1 Erwähnte IFS-Editor ermöglicht in weiten Bereichen bereits die technische Realisierung des IFS-Flow. Die Erstellung und manuelle Positionierung eines einzelnen IFBs ist mit Hilfe eines Assistenten im IFS-Editor möglich. In diesem Assistenten ist das in 4.3 erstellte Verfahren eingebunden (siehe Abbildung). Der Assistent wird auch die Auswahl der gewünschten Strategie zur Verteilung ermöglichen.

Das erstellte Verfahren bringt auch Vorteile, wenn man nur einen einzigen IFB erzeugen will. Man kann als Strategie für die Verteilung die *obere Schranke für die Anzahl an IFBs* wählen und dann diese Schranke auf eins setzen. So erhält man eine konstenminimale automatische Positionierung.

Die Menge der Kommunikationskomponenten und welche Komponenten miteinander kommunizieren wird im IFD-Mapping angegeben. IFD-Mappings sind nicht vollständig im IFS-Format spezifiziert und können noch nicht mit dem IFS-Editor erstellt oder verarbeitet werden (Stand: 08.03.2005). Daher war das erstellte Verfahren zum Zeitpunkt der Fertigstellung dieser Arbeit noch nicht vollständig funktional im IFS-Editor eingebunden.

Die Implementierung und Integration des IFD-Mappings in den IFS-Editor wird aber voraussichtlich im zweiten Quartal 2005 abgeschlossen, so dass die funktionale Integration der automatischen Verteilung kurze Zeit später abgeschlossen werden kann.

5 Dedicated Protocol

5.1 Ansätze für die Verwendung eines Dedicated Protocols

Wenn der Clustering Algorithmus stoppt, weil keine weitere Migration von IFD-Mappings mehr möglich ist, kann der Grund dafür das Fehlen freier Verbindungen in der Systemarchitektur sein. In diesem Fall könnte ein *Dedicated Protocol* verwendet werden, um die Verbindungskosten zwischen vorhandenen IFBs zu reduzieren. Dies wird durch das Zusammenfassen von zwei Protokollen zu einem Dedicated Protocol erreicht.

Ein solches Protokoll verwendet nur das Maximum der benötigten Verbindungen der beiden gegebenen Protokolle, anstelle der Summe. Wenn durch die Verwendung eines Dedicated Protocols die Anzahl der verwendeten Verbindungen reduziert werden konnte, kann der Clustering Algorithmus fortgesetzt werden.

5.2 Merged Protocol als Realisierung eines Dedicated Protocols

Für die Berechnung eines Merged Protocols aus zwei Protokollen müssen die folgenden Beschränkungen eingehalten werden:

- Die gemeinsame Verwendung von Verbindungen erlaubt nur die Ausführung eines Grundblocks aus einem Protokoll zu einem Zeitpunkt, während das andere Protokoll warten muss.
- Die Transitionen zwischen den Grundblöcken dürfen keine zeitlichen Abhängigkeiten beinhalten.
- Das Wechseln der Ausführung von einem Protokoll zu einem anderen ist nur am Ende eines Grundblocks möglich, damit die Verarbeitung von Datenpaketen nicht unterbrochen wird.
- Das Merged Protocol muss ein deterministisches Verhalten aufweisen.

Der hier vorgestellte Ansatz für Dedicated Protocols kombiniert zwei Protokolle zu einem neuen sogenannten *Merged Protocol*. Dazu werden die CFGs der beiden Protokolle zu einem neuen CFG kombiniert. Kombinieren zweier CFGs bedeutet das *Einfügen gültiger Transitionen* zwischen die Grundblöcke der beiden Protokolle P_A und P_B. Daher ist es nicht möglich von einem inneren Zustand eines Grundblocks zwischen P_A und P_B zu wechseln.

Ein Merged Protocol kann nicht durch einfaches Erzeugen der Verbindung der beiden CFGs erstellt werden. Das Ergebnis wäre die Vereinigung der beiden Graphen mit einer neuen Kante von jedem Knoten des ersten Graphen zu jedem Knoten des zweiten Graphen. Mit Ausnahme von trivialen Fällen (nur ein Grundblock im CFG) würde dieser neue Graph ungültige Pfade enthalten. Die Pfade in einem Merged Protokoll P_{AB} müssen alle gültig sein.

Ein einzelner Pfad in einem Graphen eines Merged Protocols P_{AB} ist gültig, genau dann, wenn alle Abfolgen von Grundblöcken aus P_A und P_B einer möglichen Abfolge in P_A oder P_B entsprechen. Es darf also im Merged Protocol keine Abfolge von Zuständen auftreten, die in den ursprünglichen Protokollen nicht möglich war.

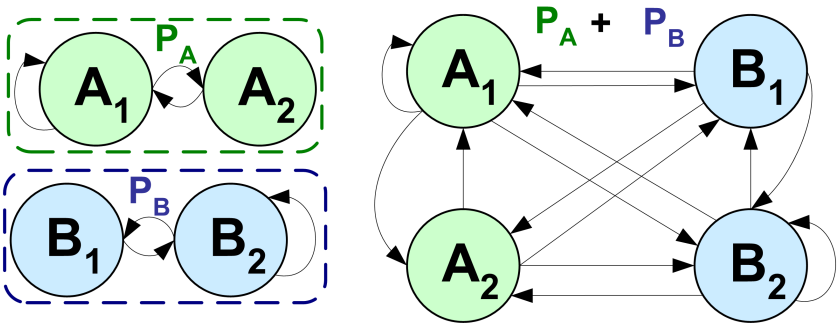

Abbildung 5.1: Verbindung von P_A und P_B mit ungültigen Pfaden

In Abbildung 5.1 ist ein Beispiel für die Verbindung der CFGs von Protokoll A und Protokoll B angegeben ($P_A + P_B$). Diese Verbindung enthält ungültige Pfade. Der Pfad A_2, B_2, A_2 würde zweimal den Grundblock A_2 besuchen, ohne das zwischendurch der Grundblock A_1 vorkam. Diese Abfolge ist im Ursprungsprotokoll P_A nicht möglich und daher ungültig.

Um keine ungültigen Pfade in einem Merged Protocol zu erzeugen, gibt es zwei mögliche Ansätze:

1. Die Transitionsbedingung einer neu eingefügten Transition wird um eine zusätzliche Bedingung erweitert. Dazu wird ein zusätzliches Element, welches den letzten ausgeführten Grundblock für jedes Protokoll speichert, mit in der Transitionsbedingung berücksichtigt. Die erweiterte Transitionsbedingung wird dann zusätzlich zu den ursprünglichen Bedingungen beachtet. Eine Transition schaltet dann

abhängig davon, ob der letzte für ein Protokoll ausgeführte Grundblock ein legaler Vorgänger zu dem Ziel-Grundblock der Transiton ist. Der aus dieser Methode resultierende Graph ist, bis auf die geänderten Transitionsbedingungen, vergleichbar mit der Verbindung von zwei Graphen.

2. Transitionen, die zu einer ungültigen Abfolge von Grundblöcken führen, darf es zu keinem Zeitpunkt der Ausführung geben. Beim Einfügen einer neuen Transition zu einem Grundblock aus Protokoll A müssen daher immer die Vorgänger aus dem gleichen Protokoll berücksichtigt werden. In dieser Form der Lösung werden Grundblöcke geklont, so dass die Information über die Abfolgen der Vorgänger im Pfad selbst kodiert wird. Der resultierende CFG enthält daher alle möglichen gültigen Abfolgen von Grundblöcken aus beiden Protokollen. Zusätzliche Elemente zum Speichern der Vorgänger oder Erweiterungen der Transitionsbedingungen sind nicht erforderlich. Das Ergebnis dieses Verfahrens ist ein Protokoll, welches von den vorhandenen Strukturen des IFS-Flows ausgeführt und interpretiert werden kann.

Das Erweitern der Transitionsbedingungen wäre verhältnismäßig aufwändig. Für jedes Merged Protocol wären zusätzliche Historyelemente zum Speichern des vergangenen Verlaufs der beinhalteten Protokolle notwendig. Diese Elementen müssten aufwändig beschrieben und letztendlich synthetisiert werden.

Das kodieren der Informationen über den Verlauf im CFG selbst ist eleganter, da das berechnete Merged Protocol wie jedes andere Protokoll behandelt und mit vorhandenen Methoden synthetisiert werden kann. Daher wurde der zweiten Methode der Vorrang gegeben und ein Algorithmus zur automatischen Berechnung eines Merged Protocols entwickelt.

Ein Beispiel für ein berechnetes Merged Protocol ist in Abbildung 5.2 zu sehen. Oben in der Abbildung sind die CFGs der beiden Ausgangsprotokolle P_A und P_B zu sehen. Der Startgrundblock ist der Grundblock, welcher mit dem Startzustand des Protokolls beginnt. In P_A und P_B hat der Startgrundblock jeweils den Index 1. Beide Protokolle enthalten neben einfachen Transitionen zu nachfolgenden Grundblöcken jeweils eine Selbsttransition. Das aus diesen beiden Protokollen berechnete Merged Protocol ist im unteren Teil der Abbildung zu sehen. In jedem Zustand des Graphen P_{AB} ist jeweils eine Nummer in runden Klammern unter der Bezeichnung des Zustandes aufgeführt. Dies gibt die Reihenfolge an, in welcher die Zustände beim Erstellen des Graphen erzeugt wurden. Die Anzahl der Grundblöcke im Merged Protocol hat sich erhöht, um alle gültigen Abfolgen von Zuständen des CFG von P_A und P_B abbilden zu können.

5.3 Protokoll Parser

Der Algorithmus zum Erstellen eines Merged Protocol benötigt als Eingabe die CFGs der beiden Protokolle. Es werden daher die Grundblöcke der beiden Protokolle benötigt.

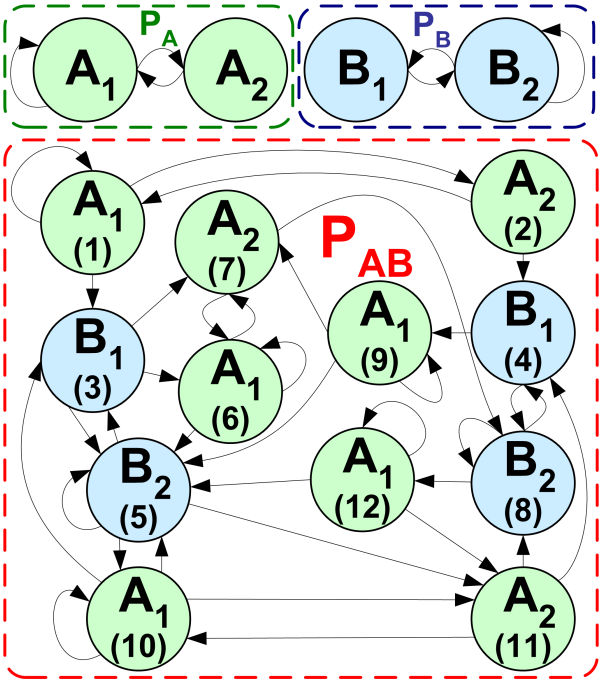

Abbildung 5.2: Merged Protocol

Für diese Aufgabe wurde im Rahmen dieser Studienarbeit das Verfahren des *Protokoll Parsers* entwickelt.

Dieses Verfahren liefert zu einer gegebenen Protokoll-FSM eine Menge mit Grundblöcken. Zur Integration dieses Verfahrens in den IFS-Editor wurde eine Implementation in Java erstellt. Mit dieser Java Methode wurden verschiedene Aspekte für die Erstellung von Merged Protokolls evaluiert. Zudem kann der Protokoll Parser noch für eine Reihe anderer Bereiche im IFS-Editor verwendet werden, welche ebenfalls die Grundblöcke einer Protokoll-FSM benötigen.

Dazu gehören die Visualisierung von Protokollen im IFS-Editor und die Extraktion von Datenpaketen für das Erstellen eines IFD-Mappings. Zusätzlich werden beim Durchführen des Verfahren syntaktische Fehler eines eingegebenen Protokolls ermittelt, wenn diese sich in Form von unerreichbaren Zuständen in der Protokoll-FSM bemerkbar machen. Diese Art von syntaktischen Fehlern können durch *Überspezifikation* entstehen [PS91].

Das Prinzip des Ablaufs beim Protokol Parser entspricht dem der *Tiefensuche*. In der folgenden Sequenz von Abbildungen ist der Ablauf des Algorithmus an einem Graphen einer Protkoll-FSM exemplarisch dargestellt. Die Ellipsen B1-B4 entsprechen dabei den gefundenen Grundblöcken.

Der Algorithmus erstellt zunächst einen Grundblock in dem nur der Startzustand des Protokolls liegt (Schritt 1). Solange es nur einen nachfolgenden Zustand gibt und dieser noch nicht in einem Grundblock liegt, werden die Nachfolger mit in den Grundblock

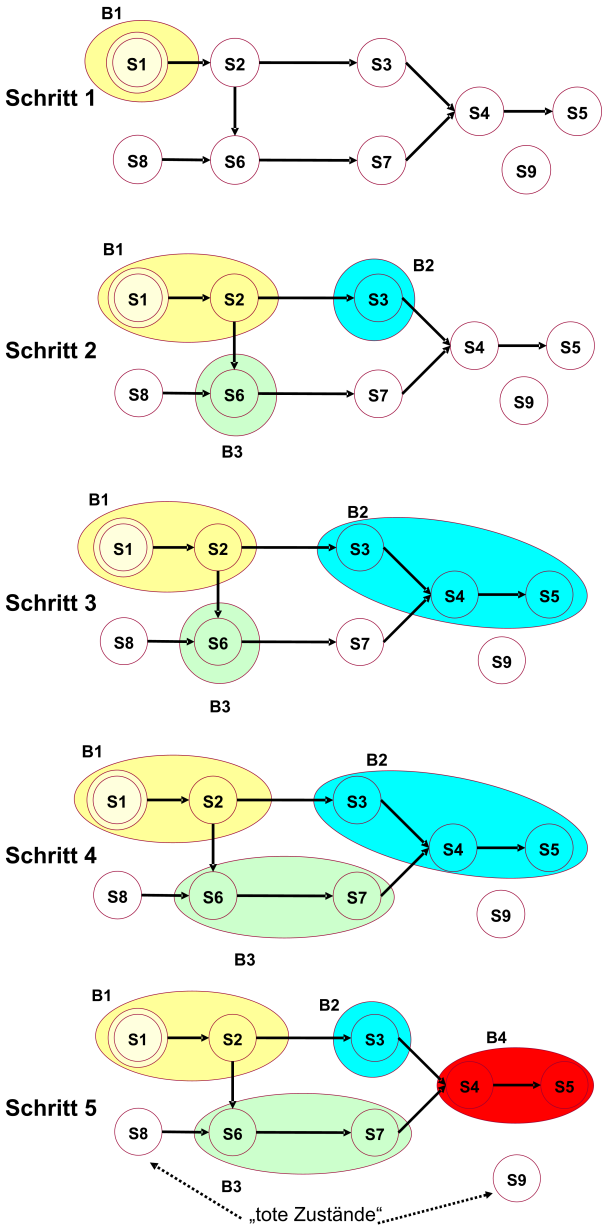

Abbildung 5.3: Protokoll Parser, Ablauf

aufgenommen. Sobald es mehrere Transitionen gibt, wird der aktuelle Grundblock als *abgeschlossen* markiert und für jeden Nachfolgezustand ein neuer Grundblock erstellt (Schritt 2).

Wenn der Nachfolgezustand bereits in einem Grundblock liegt (siehe S4 in Schritt 4), dann wird der aktuell bearbeitet Grundbock ebenfalls als *abgeschlossen* markiert und der Grundblock des Nachfolgezustands aufgeteilt (Bild 5). Dies wird solange durchgeführt, bis alle vorhandenen Grundblöcke als *abgeschlossen* markiert wurden.

Sind alle Grundblöcke als *abgeschlossen* markiert und es gibt noch Zustände die zu keinem Grundblock gehören, so handelt es sich um nicht erreichbare Zustände (*tote Zustände*). Diese Zuständen können durch Fehler in der eingegebenen Spezifikation des Protokolls entstehen.

5.4 Berechnung eines Merged Protocols

Die Berechnung eines Merged Protocols kann automatisch durch einen iterativen Algorithmus erfolgen. Zuerst werden dafür die CFGs von zwei übergebenen Protokollen erzeugt. Dies geschieht mit Hilfe des Protocol Parsers. Dann wird eine Pfad Komposition der beiden CFGs vorgenommen. Das Ergebnis ist der CFG eines Merged Protocols.

Der folgende Algorithmus arbeitet auf Listen von Grundblöcken. Ein Grundblock kann zwei verschiedene Zustände haben: *unbehandelt* oder *abgeschlossen*. Anfangs ist jeder neu erzeugte Grundblock im Zustand *unbehandelt*. Wenn alle Transitionen zu allen nachfolgenden Grundblöcken generiert wurden und alle fehlenden nachfolgenden Grundblöcke des aktuell bearbeiteten Grundblocks erstellt sind, dann wird der aktuelle Grundblock (BB_{curr}) als *abgeschlossen* markiert.

Die Liste der Nachfolger des aktuellen Grundblocks ergibt sich aus den Nachfolgern welche der Grundblock im eigenen Protokoll hat. Hinter diese kommen die Nachfolgerlisten seiner Vorgänger, wobei der aktuelle Grundblock nicht selbst aus diesen Nachfolgerlisten übernommen wird, da sonst jeder Grundblock eine Selbstransition hätte. Am Anfang des Algorithmus bekommt der erste Grundblock den Startgrundblock des anderen Protokolls mit in die Nachfolgerliste.

Pseudo Code für die Generierung eines Merged Protocol

```
01 BB_curr = erster BB des ersten Protokoll
02 WHILE(BB_curr != null) DO
03   FOR(i in succ(BB_curr)){
04     IF (succ(BB_curr)[i] passt zu einem BB_abgeschlossen) {
05       erzeuge Transition zu BB_passend
06       aktualisiere(succ(BB_curr), pred(BB_passend)) }
07     ELSE IF(succ(BB_curr) passt zu einem BB_unbehandelt){
08       erzeuge Transition zu BB_passend
09       aktualisiere (succ(BB_curr), pred(BB_passend)) }
10     ELSE {
11       erzeuge neuen BB_neu
12       unbehandelt BB List.add(BB_neu)
13       aktualisiere(succ(BB_curr), pred(BB_neu) }
14   } // END FOR
15   markiere BB_curr als abgeschlossen
16   BB_curr = unbehandelt BB List.getNext()
17 } // END WHILE
```

Der Algorithmus verwendet die folgenden Definitionen und Funktionen:

- Die beiden übergebenen Protokolle werden mit P_1 und P_2 bezeichnet
- $\text{succ}(\text{BB}_x)$ liefert eine Liste mit allen nachfolgenden Grundblöcken von x
- $\text{pred}(\text{BB}_x)$ liefert eine Liste mit allen vorangehenden Grundblöcken von x
- BB $a \in P_1$ ist passend zu $\text{BB}_{abgeschlossen}$ b wenn $\text{succ}(b) = \text{succ}(a) \,/\, b$
- BB $a \in P_1$ ist passend zu $\text{BB}_{unbehandelt}$ b wenn gilt:
  - $\text{succ}(\,\forall\ \text{BB} : (\text{pred}(b) \in P_1)\,) \equiv \text{succ}(a)$
  - $\forall\ (\,\text{pred}(b) \in P_2\,) \in \text{pred}(a)$

In der Tabelle 5.1 ist der Ablauf des Algorithmus und der Aufbau der verschiedenen Listen beim Erstellen des Merged Protocols aus P_A und P_B zu sehen. Es handelt sich um die gleichen Protokolle P_A und P_B wie in Abbildung 5.2. Die Tabelle zeigt den Zustand der einzelnen Listen jeweils nachdem der aktuelle Grundblock abgearbeitet und als abgeschlossen markiert wurde.

Nr.	abgeschlossen	unbehandelt	aktueller BB	Nachfolger
1	1	2,3	$A_1(1)$	A_1, A_2, B_1
2	1,2	3,4	$A_2(2)$	A_1, B_1
3	1,2,3	4,5,6,7	$B_1(3)$	B_2, A_1, A_2
4	1,2,3,4	5,6,7,8,9	$B_1(4)$	B_2, A_1
5	1,2,3,4,5	6,7,8,9,10,11	$B_2(5)$	B_1, B_2, A_1, A_2
6	1,2,3,4,5,6	7,8,9,10,11	$A_1(6)$	A_1, A_2, B_2
7	1,2,3,4,5,6,7	8,9,10,11	$A_2(7)$	A_1, B_2
8	1,2,3,4,5,6,7,8	9,10,11,12	$B_2(8)$	B_1, B_2, A_1
9	1,2,3,4,5,6,7,8,9	10,11,12	$A_1(9)$	A_1, A_2, B_2
10	1,2,3,4,5,6,7,8,9,10	11,12	$A_1(10)$	A_1, A_2, B_1, B_2
11	1,2,3,4,5,6,7,8,9,10,11	12	$A_2(11)$	A_1, B_1, B_2
12	1,2,3,4,5,6,7,8,9,10,11,12	-	$A_1(12)$	A_1, A_2, B_1, B_2

Tabelle 5.1: Merge Algorithmus

Die ersten drei Schritte des Merged Algorithmus sind in 5.4 zu sehen. Im ersten Schritt wird der Startgrundblock des Protokoll A in den neuen Merged Protokoll Graphen eingefügt. Er erhält die Nummer 1 (aktueller BB: $A_1(1)$). Die Nachfolgerliste ergibt sich aus den Nachfolgern von A_1 (A_1, A_2) und dem Startgrundblock aus Protkoll B (B_1).

Als nächstes werden die Transitionen zu den Grundblöcken der Nachfolgerliste eingefügt. Dabei wird als erstes versucht einen passenden *abgeschlossenen* Grundblock zu finden. Existiert solch ein passender Grundblock nicht, wird in der Liste der *unbehandelten* Grundblöcke weitergesucht. Für den Nachfolger A_1 des aktuellen Grundblocks wird nun ein passender Grundblock gefunden. In diesem Fall $A_1(1)$ selbst.

Für die restlichen Nachfolger A_2 und B_1 wird kein vorhandener Grundblock gefunden, daher werden entsprechend neue Grundblöcke ($A_2(2)$ und $B_1(3)$) erstellt. Da nun Transitionen zu allen Grundblöcken der Nachfolgerliste erstellt wurden, wird $A_1(1)$ als *abgeschlossen* markiert. Die Bearbeitung geht mit dem nächsten *unbehandelten* Grundblock ($A_2(2)$) weiter.

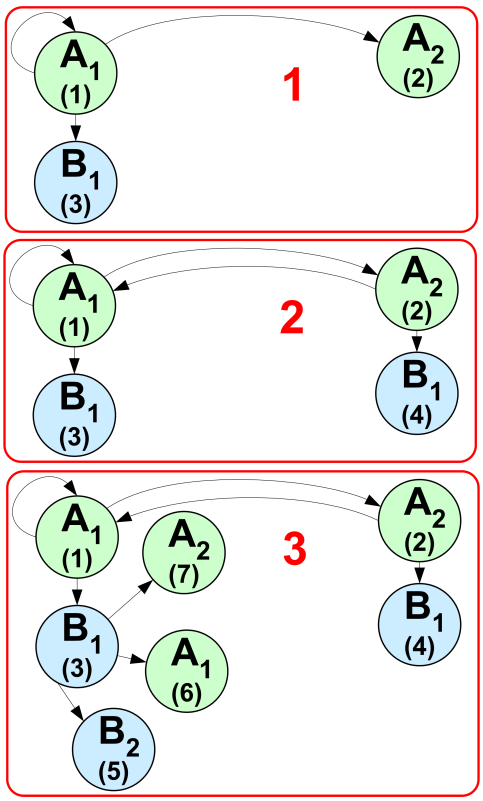

Abbildung 5.4: Merged Protocol 1-3

5.5 Auswirkung auf den IFB

Abbildung 5.5 zeigt zwei IFBs zwischen denen via eines Merged Protocols kommuniziert wird. In dem angegebenen Beispiel interagieren $T_1 \leftrightarrow T_3$ und $T_2 \leftrightarrow T_4$ miteinander. Alle Tasks verwenden inkompatible Protokolle. Die Konvertierung der Protokolle wird für $T_1 \leftrightarrow T_3$ und $T_2 \leftrightarrow T_4$ auf der für die Verbindungskosten optimalen Seite vorgenommen. Das bedeutet nur die kosten-minimalen Protokolle $P_A = min(P_1, P_3)$ und $P_B = min(P_2, P_4)$ müssen als Verbindungen durch die Systemarchitektur geführt werden.

Aus den so bestimmten Protokollen P_A und P_B wird das Merged Protocol P_{AB} generiert. Es existiert nur zwischen den beiden IFBs und ist somit „transparent“ für die Tasks. Die Anzahl der benötigten Verbindungen für P_{AB} reduziert sich auf das Maximum $max(\#Verbindungen(P_A), \#Verbindungen(P_B))$.

Dabei ist zu beachten, dass es sich um eine Kommunikation zwischen zwei IFBs handelt und die Daten nur einmal gemäß dem Mapping umgewandelt werden müssen. Auf der Seite wo das Mapping nicht durchgeführt werden muss, werden SH-Modes generiert, welche die Daten einfach unverändert weiterreichen (*triviale SH-Modes*).

Die Anzahl der benötigen Verbindungen (Verbindungskosten) wird reduziert, es werden aber mehr PH-Modes benötigt. Daher kommt es zu einem leichten Anstieg der Implementierungskosten. Da die Verbindungen in der Systemarchitektur die begrenztere Ressource sind, ist das nicht kritisch.

Ein Merged Protocol aus mehr als zwei Protokollen kann durch iteratives Anwenden des Algorithmus auf jeweils zwei Protokolle erstellt werden. Dabei reduziert sich die Anzahl der benötigten Verbindungen für $P_{A..X}$ auf
$max(\#Verbindungen(P_A), ..., \#Verbindungen(P_X))$.

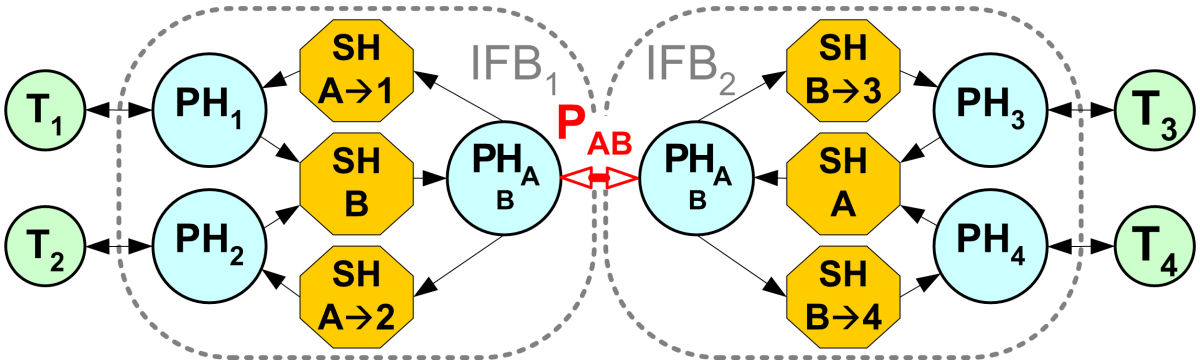

Abbildung 5.5: IFBs mit Merged Protocol

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung der Arbeit

Das Ziel dieser Arbeit war die Erstellung von Verfahren zur Unterstützung der Synthese von Kommunikationsstrukturen in verteilten eingebetteten Systemen mit Hilfe des IFS-Flows.

Dazu wurde zunächst das Modell des Kommunikationsgraphen basierend auf dem Spezifikationsgraph Modell vorgestellt. Für die Erstellung einer Kommunikationsstruktur gemäß dem Modell des Kommunikationsgraphen, ergaben sich die Anforderungen für eine automatische Verteilung und Abbildung der Kommunikationsknoten (realisiert durch IFBs) auf die Systemarchitektur.

Im weiteren Verlauf wurden die Auswirkungen der Positionierung von IFBs an unterschiedlichen Stellen der Systemarchitektur erläutert. Diese Auswirkungen resultierten in der Erstellung der verwendeten Kostenfunktionen. Für die automatisierte Verteilung wurde dann ein Clusteringverfahren erstellt, dass die automatische Verteilung von IFBs und die Zuordnung von IFD-Mappings ermöglicht. Basierend darauf wurden fünf verschiedene Verteilungsstrategien und ihre Auswirkungen auf die Kosten dargelegt.

Um Kommunikations-Engpässe zu überwinden, welche die automatische Verteilung einschränken, wurde das Prinzip des Merged Protocols vorgestellt. Dazu wurde der Alogorithmus definiert mit dem ein Merged Protocol automatisch berechnet werden kann. Die erstellten Verfahren verbessern den IFS-Flow aufgrund der vereinfachten und effizienten Synthese von Kommunikationstrukturen. Der Integration in das Konzept des IFS-Design-Flow wurde vorgestellt.

6.2 Ausblick auf weitere Modellierung

Die Präsentation der angegebenen Verfahren hat gezeigt, dass der IFS-Flow hinsichtlich der Synthese von Kommunikationsstrukturen erweitert werden konnte.

Für die Umsetzung des IFS-Flows wird der IFS-Editor verwendet. Mit ihm können die einzelnen Designschritte und die Synthese vorgenommen werden. Teile der in dieser Arbeit erstellten Verfahren sind bereits im IFS-Editor umgesetzt. Der im Rahmen dieser Arbeit entstandene Protokoll Parser wurde bereits vollständig. Er wird auch von ande-

ren Bereichen des IFS-Editors verwendet. Im IFS-Editor können Protokolle mit Hilfe des Protokoll Parsers visualisiert werden. Ein weiterer wichtiger Bereich, für den der Protokoll Parser zukünftig benötigt wird, ist die Erstellung der IFD-Mapping Equations.

Die Verfahren zur automatisierten Verteilung und Erstellung von Merged Protokollen sind bereits an entsprechenden Stellen im IFS-Editor positioniert, aber noch nicht vollständig funktional, da andere benötigte Komponenten noch nicht zur Verfügung stehen. Insbesondere fehlen IFD-Mapping equations. Ohne diese kann nicht ermittelt werden, welche Komponenten überhaupt miteinander kommunizieren wollen. Die Realisierung von IFD-Mapping equations wird aber in Kürze zur Verfügung stehen.

Nachdem IFD-Mappings mit dem IFS-Editor erstellt werden können, wird die zukünftige Arbeit im Bereich der Synthese von Kommunikationsstrukturen sich mit der Erstellung eines Beispiel Szenarios beschäftigen, um reale Werte für die vorgestellten Kostenfunktionen zu erhalten. Damit lassen sich dann die Größenordnungen für den Ressourcenbedarf der einzelnen Komponenten (wie z.B. PH-Modes oder Control Unit) ermitteln.

6.3 Ausblick für mögliche Erweiterungen

Nach der Ermittlung von Werten für die Kostenfunktion sind noch weitere Forschungsarbeiten im Rahmen von Studien- und Diplomarbeiten denkbar. Insbesondere sind die folgenden Themen geeignet, die sich im Verlauf der Arbeit ergeben haben, deren vollständige Ausarbeitung aber aus zeitlichen oder thematischen Gründen nicht möglich war:

- Optimierung der angegebenen Strategien für die automatische Verteilung von IFBs mit einer automatisierten Abänderung der Strategie, falls sich das Ziel der gewählten Strategie nicht erreichen lässt.
- Erweiterung des Verfahrens zur automatisierten Verteilung von IFBs, so dass die positionierten IFBs geforderte Kriterien für harte Realzeit einhalten können.
- Visualisierung der Verteilung bzw. des Clusterings von IFBs. Dabei könnte dem Benutzer die Möglichkeit zur Interaktion ermöglicht werden, um das Ergebnis aktiv zu beeinflussen.
- Erstellung eines Merged Protocols unter Verwendung der Erweiterung der Transitionsbedingungen. Diese würde das Synthetisieren von Elementen erfordern die den Verlauf (History) von Protokollen speichern können. Dieses Verfahren könnte Vorteile beim Erstellen eines Merged Protocols aus mehr als zwei Protokollen bieten, da sich bei diesem Verfahren die entstehende Anzahl der Zustände im wesentlichen auf die Summe der Zustände der einzelnen Protokolle beschränkt.

Literaturverzeichnis

[AKJ88] Richard C. Dubes Anil K. Jain. *Algorithms for clustering data.* Prentice-Hall, Inc., 1988.

[GNL67] W. T. Williams G. N. Lance. Mixed-data classificatory programs i - agglomerative systems. *Australian Computer Journal*, 1(1):15–20, January 1967.

[Har69] F. Harary. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.

[Ihm02a] Ihmor, Stefan and Visarius, Markus and Hardt, Wolfram. A Consistent Design Methodology for Configurable HW/SW-Interfaces in Embedded Systems. In *Proc. of the IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems: Design and Analysis of Distributed Embedded Systems*, Montreal, Canada, Aug. 2002.

[Ihm02b] Ihmor, Stefan and Visarius, Markus and Hardt, Wolfram. A Design Methodology for Application-specific Real-Time Interfaces. In *Proceedings of 2002 IEEE International Conference on Computer Design (ICCD): VLSI in Computers & Processors*, IEEE International Conference on Computer Design, Freiburg, Germany, Sept. 2002.

[Ihm03a] Ihmor, Stefan and Bastos Jr., Nilson and Klein, Rafael Cardoso and Visarius, Markus and Hardt, Wolfram. Rapid Prototyping of Realtime Communication - A Case Study: Interacting Robots. In *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP'03)*, June 2003.

[Ihm03b] Ihmor, Stefan and Visarius, Markus and Hardt, Wolfram. Modeling of Configurable HW/SW-Interfaces. pages 51 – 60, 2003.

[Ihm04] Ihmor, Stefan and Hardt, Wolfram. Runtime Reconfigurable Interfaces - The RTR-IFB Approach. In *Proceedings of the 11th Reconfigurable Architectures Workshop (RAW'04)*. IEEE Computer Society, 26 - 27 April 2004.

[KHK96] Friedrich Schwarz Karl-Heinz Kiyek. *Mathematik für Informatiker*, pages 32–33. B. G. Teubener Stuttgart, 1996.

[PRSV98] R. Passerone, J. Rowson, and A. Sangiovanni-Vincentelli. Automatic synthesis of interfaces between incompatible protocols. In *Proceedings of Design Automation Conference*, pages 8–13, 1998.

[PS91] Robert L. Probert and Kassam Saleh. Synthesis of communication protocols: Survey and assessment. *IEEE transactions on computers*, 40(4):468–469, April 1991.

[Tei97] J. Teich. *Digitale Hardware/Software-Systeme: Synthese und Optimierung.* Springer-Lehrbuch, Heidelberg, New York, Tokio, 1997.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, Juni 1996.