Emulator- und kostenbasierte Analyse von Network-on-Chip

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor-Ingenieur (abgekürzt: Dr.-Ing.)
genehmigte Dissertation

von

Diplom-Ingenieur, Diplom-Kaufmann

Martin Christian Neuenhahn

Geboren am 9. Dezember 1976

in Herten

2019

1. Referent:	Prof. Dr.-Ing. Holger Blume
Korreferent:	Prof. Dr.-Ing. Tobias Noll
Prüfungsvorsitz:	Prof. Dr.-Ing. Bernhard Wicht
Tag der Promotion:	26. Juni 2019

Für Jutta, Carlotta und Hugo

Vorwort

Die vorliegende Arbeit wurde während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der Rheinisch-Westfälischen Technischen Hochschule in Aachen begonnen. Mein herzlicher Dank gehört Herr Univ.-Prof. Dr.-Ing. Tobias G. Noll für die interessante Themenstellung, die Unterstützung und Betreuung, sowie die Übernahme des Korreferates.

Des Weiteren möchte ich ebenso herzlich Herrn Univ.-Prof. Dr.-Ing. habil. Holger Blume, dem Leiter des Instituts für Mikroelektronische Systeme an der Leibniz Universität Hannover, danken. Er gab mir die Möglichkeit, diese Arbeit an seinem Institut zu beenden. Ausdrücklich möchte ich mich ich für sein außerordentliches Interesse an dieser Arbeit, die engagierten Diskussionen und die Betreuung in der finalen Phase dieser Doktorarbeit bedanken.

Allen ehemaligen Kollegen, studentischen Hilfskräften, Studien-, Master- und Diplomarbeitern gebührt ein besonderer Dank für die unterstützende Zusammenarbeit und für die angenehme und konstruktive Bereicherung.

Mein größter Dank gilt meiner Familie. Meiner Frau Jutta, meiner Tochter Carlotta, meinem Sohn Hugo, meinen Eltern, meinem Bruder, Schwägerinnen, Schwager, Neffen, Schwiegereltern, Tanten u Onkel. Ohne ihre bedingungslose Unterstützung und den gebotenen Rückhalt wäre dieser Lebensweg nicht möglich gewesen.

Ebenso möchte ich mich bei meinen engen Freunden bedanken, die immer an mich geglaubt haben.

Düsseldorf, im Juli 2019 Martin C. Neuenhahn

Kurzfassung

Die Komplexität der Kommunikation auf aktuellen und zukünftigen Multi-Kern System on Chip ist mit gängigen Kommunikationsarchitekturen wie Bussen oder Punkt-zu-Punkt Verbindungen kaum zu beherrschen. Network-on-Chip (NoC) stellen eine mögliche Lösung dieses Problems dar. Im Rahmen dieser Arbeit wurde ein modulares und parametrisierbares Network-on-Chip entwickelt. Dies unterstütze eine Vielzahl von NoC-Parametern wie zum Beispiel Topologie, Routing Algorithmus und Vermittlungstechnik. Die erstellte VHDL Bibliothek für NoC ermöglicht die automatische Generierung von NoC-Beschreibungen in VHDL.

Für die Untersuchung der Performance, Fläche und Verlustleistung der modellierten NoC wurden exemplarische VLSI-Implementierungen der NoC-Komponenten mit Hilfe von Standardzellen erstellt. Zur Reduzierung der Kosten und Steigerung der Performance sind physikalisch optimierte Kernkomponenten in Kombination mit Standardzellen verwendet worden. Dies reduziert die Kosten der NoC-Implementierungen signifikant wobei die Parametrisierbarkeit erhalten bleibt. Die Kosten für die NoC-Komponenten in Abhängigkeit der NoC-Parameter wurden mit mathematischen Modellen beschrieben. Diese Modelle erlauben die Abschätzung der zu erwartenden Kosten schon in frühen Entwurfsphasen.

Neben den Kosten, die durch ein NoC verursacht werden, ist die Bestimmung der Performance eines NoCs wichtig. Es wurden verschiedene Techniken (z.B. Simulation auf unterschiedlichen Abstraktionsebenen, Emulation auf einem FPGA) implementiert, um die Performance von NoC zu bestimmen. Die Erweiterung der NoC-Beschreibung um weitere Bibliotheken zur Simulation mit SystemC und Colored-Petri-Nets, einer Emulation auf einem FPGA und der statischen Analyse ermöglichten einen Vergleich und eine Bewertung dieser Techniken. Die Analyse-Techniken wurden den unterschiedlichen Phasen im Entwurfs-prozess von NoC zugeordnet.

Durch die Vielzahl an NoC-Parametern ist der Entwurf eines optimalen NoC sehr komplex und aufwändig. Die Erkenntnisse dieser Arbeit wurden in einer Entwurfsmethodik zusammengeführt. Dieses Spiral-Modell ermöglicht eine effiziente, automatisierte Implementierung von NoC.

Bei dem Vergleich der implementierten NoC-Komponenten mit Beispielen aus der Literatur konnte die Effizienz und Leistungsfähigkeit gezeigt werden. Für Anwendungsbeispiele aus der Literatur und zufälligen Datenverkehr konnte der Entwurfsraum für NoC erfolgreich untersucht und jeweils Pareto-optimale NoC identifiziert werden. Die Analyse des Kommunikationsverhalten eines realen Multi-Core Prozessors mit 61 Prozessorkernen und Abbildung auf den FPGA-basierten Emulator für NoC zeigen, dass die vorgestellte Methodik grundsätzlich gut für den Entwurf und die Analyse von NoC geeignet ist.

Abstract

The complexity of communication on current and future multi-core system on chip can hardly be mastered with common communication architectures such as buses or point-to-point connections. Networks-on-Chip (NoC) are a possible solution to this problem. In the context of this work a modular and parameterizable Network-on-Chip was developed. It supports a multitude of NoC parameters such as topology, routing algorithm and switching technology. The created VHDL library for NoC allows the automatic generation of NoC descriptions in VHDL.

For the investigation of the performance, area and power consumption of the modelled NoC exemplary VLSI implementations of the NoC components were created using standard cells. To reduce costs and increase performance, physically optimized core components are used in combination with standard cells. This significantly reduces the cost of NoC implementations while maintaining parameterizability. The costs for the NoC components depending on the NoC parameters are described with mathematical models. These models allow the estimation of the expected costs of a NoC already in early design phases.

Besides the costs caused by a NoC the determination of the performance of a NoC is important. There are different techniques (e.g. simulation on different abstraction levels, emulation on an FPGA) to verify this. The extension of the NoC description by further libraries for the simulation with SystemC and Colored-Petri-Nets, an emulation on an FPGA and the static analysis made a comparison and an evaluation of these techniques possible. Based on this, the analysis techniques were assigned to the different phases of the NoC design process.

Due to the large number of NoC parameters, the design of an optimal NoC is very complex and time-consuming. Therefore, the findings of this work were combined in a design methodology. This spiral model enables an efficient, automated implementation of NoC.

The comparison of the NoC components generated by the proposed method with implementations from the literature shows the efficiency of these components. For application examples from literature and for random data traffic, the design space for NoC could be successfully investigated and Pareto-optimal NoC identified. The analysis of the communication behavior of a real multi-core processor with 61 processor cores and mapping to the FPGA-based emulator for NoC shows that the presented methodology is well suited for the design and analysis of NoC.

Keywords: Emulation; Cost Estimation; Network-on-Chip; FPGA

Schlagwörter: Emulation; Kostenschätzung; Network-on-Chip; FPGA

Inhaltsverzeichnis

1 Einleitung

Aktuelle und zukünftige Systeme der digitalen Signalverarbeitung, wie mobile Telekommunikations- oder Multimedia-Anwendungen, haben einen stetig wachsenden Bedarf an Rechenleistung. Gleichzeitig sollen Systeme, auf denen diese Anwendungen ausgeführt werden, eine möglichst geringe Verlustleistung verursachen. Eine weitere Anforderung ist eine kurze Entwicklungszeit der benötigten Hard- und Software, sowie eine möglichst große Flexibilität. Dies bedeutet, dass Anwendungen dem System hinzugefügt oder vom System entfernt werden können.

Durch die Fortschritte der VLSI-Technologie lassen sich immer kleinere Strukturen auf einem Chip realisieren. Bei gleichbleibender Chipfläche führt dieser Trend zu immer mehr Transistoren auf einem Chip [1], welche die Realisierung so genannter Systems-on-Chip (SoC) ermöglichen. SoC integrieren komplette Hardwaresysteme auf einem einzelnen Chip, der aus mehreren funktionalen Einheiten (Prozessorkerne, DSP-Kerne, Speicher, eingebettete FPGAs (eFPGA) etc.) bestehen kann. Die funktionalen Einheiten eines SoC können aus identischen Architekturblöcken (homogene SoC) oder aus unterschiedlichen Architekturblöcken bestehen (heterogene SoC) [2]. Damit sind SoC eine Möglichkeit, um die oben genannten Anforderungen an Systeme der digitalen Signalverarbeitung zu erfüllen. Aktuell werden SoC mit bis zu 64 funktionalen Einheiten kommerziell gefertigt.

Die steigende Komplexität dieser Chips erforderte neue Design-Techniken. Dies bezieht sich sowohl auf die Entwicklung der Hardware, als auch auf die Entwicklung von Software für SoC [3]. Für die Hardware-Entwicklung wird ein modularer Aufbau des SoC angestrebt, bei dem einzelne funktionale Einheiten über spezifizierte Schnittstellen verfügen und so an einer anderen Stelle im SoC oder auch in einem anderen SoC wieder eingesetzt werden können (IP-Reuse). Eine weitere Maßnahme, die sowohl den Hardware- wie auch den Software-Entwurf eines SoC betrifft ist die Entkopplung von Kommunikation und Funktion, wie sie bereits im ISO/OSI-Referenzmodell für Computersysteme vorgestellt wurde [4]. Durch diese Trennung können funktionale Einheiten und die Kommunikationsstruktur eines SoC parallel entwickelt werden. Darüber hinaus können funktionale Einheiten, ohne genaue Kenntnis der Kommunikationsstruktur, über eine spezifizierte Schnittstelle Daten austauschen.

An die Kommunikationsstruktur werden von den Systemen, die auf ein SoC abgebildet werden, hohe Anforderungen gestellt. Die Kommunikationsstruktur soll einen hohen Datendurchsatz bei gleichzeitig geringer Latenz ermöglichen. Zusätzlich können, je nach Systemanforderungen, weitere Quality-of-Service-Anforderungen (QoS), wie beispielsweise garantierte Durchsatzraten oder eine maximal zulässige Datenfehlerrate gefordert werden. Eine Kommunikationsstruktur sollte darüber hinaus flexibel und skalierbar sein in dem sie für unterschiedliche Anwendungen und Systeme geeignet [5] und auch für zukünftige, größere

SoC zu verwenden ist. Dies führt zu geringeren Entwicklungszeiten und geringerem Entwicklungsaufwand [6]. Die vorgenannten Anforderungen sollen bei möglichst geringen Kosten erfüllt werden, die sich auf einem SoC beispielsweise in der Fläche und/oder der Verlustleistung des SoC ausdrücken lassen.

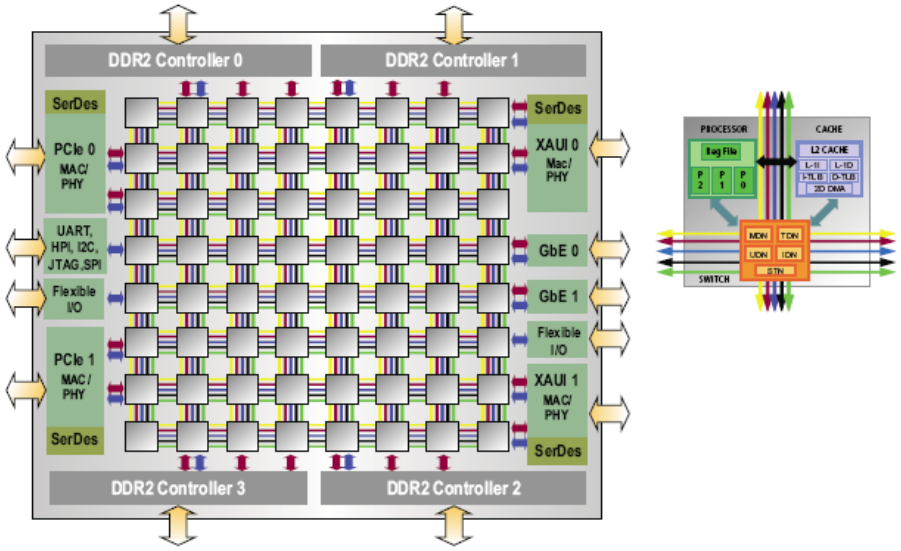


Abbildung 1.1: Exemplarisches homogenes SoC mit 64 Prozessorkernen [7]

Ein weiteres Problem, was insbesondere für zukünftige CMOS-Prozesse und entsprechend kleine Strukturgrößen an Relevanz gewinnt, ist die relative Verzögerung, die durch eine Verbindungsleitung verursacht wird. Diese steigt bei Technologien mit kleineren Strukturgrößen für globale Leitungen weiter an. Verursacht wird dieser Effekt durch die trotz Strukturverkleinerung unveränderte Länge der globalen Leitungen bei gleichzeitig steigendem Widerstand der Leitung.

Die Länge der globalen bleibt häufig konstant, da durch die sinkende Strukturgröße die Siliziumfläche der Chips nicht kleiner wird, sondern mehr Transistoren auf derselben Siliziumfläche verwendet werden. Durch zusätzliche Maßnahmen (Repeater, Booster) kann dieser Anstieg reduziert, jedoch nicht komplett kompensiert werden [8]. Im Gegensatz dazu sinkt die Verzögerung, die durch logische Gatter verursacht wird. Für zukünftige Technologien wird bei langen, globalen Leitungen die Übertragungsgeschwindigkeit in den logischen Gattern bestimmt. Daher sind möglichst kurze Leitungen für die globale Kommunikation mit hohen Datenraten wünschenswert.

Klassische Kommunikationsarchitekturen, wie zum Beispiel Busse oder Punkt-zu-Punkt-Verbindungen können diese Anforderungen nicht erfüllen [9]. Insbesondere sind diese Strukturen nicht beliebig skalierbar. Neben segmentierten oder hierarchischen Bus-Systemen [10] sind Network-on-Chip (NoC) eine häufig vorgeschlagene Lösung, um diesen Anforderungen an die Kommunikationsstruktur gerecht zu werden [11,12,13].

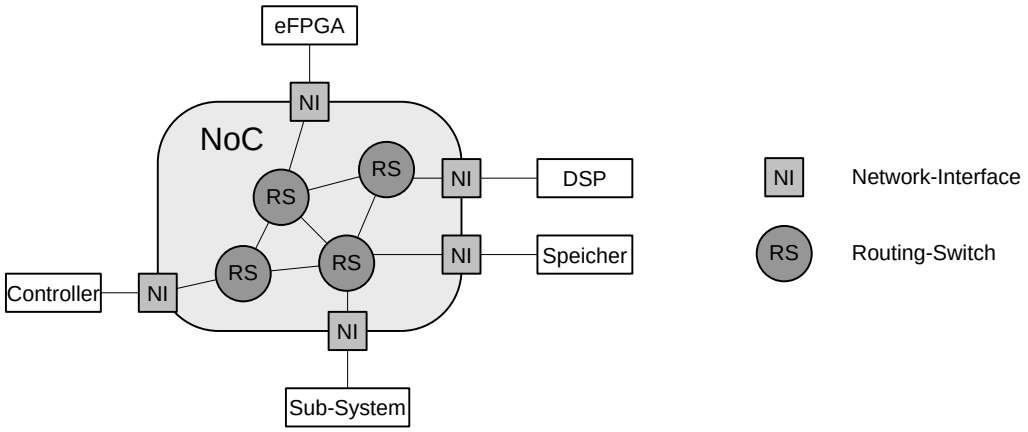


Abbildung 1.2: Exemplarisches NoC das funktionale Einheiten (eFPGA, DSP, Kontroller, Speicher und Sub-System) miteinander verbindet.

NoC übertragen Daten von einer funktionalen Einheit zu einer anderen, wobei die Daten zunächst von einer funktionalen Einheit an ein sogenanntes Network-Interface (NI) übergeben werden. Anschließend werden die Daten von diesem Network-Interface über einen oder auch mehrere Routing-Switches (RS) an die empfangende funktionale Einheit übertragen, die über ein Network-Interface an das NoC angeschlossen ist (vgl. Abbildung 1.2).

Die Eigenschaften und Kosten von NoC werden durch eine Vielzahl an NoC-spezifischen Parametern bestimmt. Diese reichen von der Ausprägung der Verbindungsleitungen über Routing-Algorithmen bis hin zur Vermittlungstechnik und Topologie des NoC und spannen daher einen großen Entwurfsraum auf. Die Achsen dieses Entwurfsraumes bilden zum einen durch das NoC verursachte Kosten, aber auch Performance-Maße, die sich gut quantitativ erfassen lassen. Die Flexibilität, die Unterstützung von QoS-Aspekten oder auch der Entwurfsaufwand und die Entwurfszeit stellen weitere Dimensionen des Entwurfsraumes dar, die jedoch schwierig zu quantifizieren sind.

Da verschiedene Systeme unterschiedliche Anforderungen an die Kommunikationsstruktur stellen, kann und muss ein NoC an die jeweiligen Anforderungen angepasst werden [14]. Es wird dazu die Kombination NoC-spezifischer Parameter benötigt, welche die Anforderungen bei möglichst geringen Kosten erfüllt. Zur Identifizierung dieser Parameterkombination ist es notwendig, den Entwurfsraum für NoC zu untersuchen.

Für eine systematische Untersuchung des Entwurfsraumes eines gegebenen Systems und gegebenen funktionalen Einheiten wird eine Entwurfsmethodik benötigt, die zum Einen den Entwurfsaufwand minimiert und zum Anderen eine Bewertung des NoC bezüglich Kosten und Performance ermöglicht.

Voraussetzung für einen geringen Entwurfsaufwand ist ein modulares und generisches NoC, bei dem einzelne Parameter mit geringem Aufwand variiert werden können. Die

Beschreibung eines solchen NoC sollte möglichst abstrakt erfolgen. Mit Hilfe dieser abstrakten Beschreibung ist es dann möglich eine NoC-Implementierung zu generieren.

Bei der Performance-Bewertung wird überprüft, ob die durch das System definierten Kommunikationsanforderungen erfüllt werden. Diese Bewertung des NoC erfolgt auf unterschiedlichen Abstraktionsebenen. Zu einem frühen Zeitpunkt im NoC-Entwurf erfolgt die Bewertung zunächst auf einer hohen Abstraktionseben, die eine erste Kosten- und Performance-Schätzung mit geringem Aufwand ermöglicht. Die damit verbundene geringere Genauigkeit ist in diesem Entwurfsstadium akzeptabel. Am Ende des Entwurfsprozesses ist hingegen eine genaue Bestimmung der Kosten sowie der Performance des NoC notwendig. Um die Performance eines NoCs für ein System exakt bestimmen zu können, werden Testbenches benötigt, die den Datenverkehr realer Anwendungen abbilden.

Im Rahmen dieser Arbeit wurde eine solche Entwurfsmethodik für NoC entwickelt. Diese Entwurfsmethodik ermöglicht es auch Entwicklern mit wenig Erfahrung im Entwerfen und Dimensionieren von NoC, den Entwurfsraum für NoC systematisch zu untersuchen. Damit lässt sich für eine gegebene Applikation bzw. Klasse von Applikationen und ein gegebenes SoC ein NoC identifizieren, welches für diese Anforderungen (Pareto-)optimal ist.

Das Ergebnis dieser Entwurfsmethodik sind die für die Aufgabenstellung Pareto-optimalen NoC. Diese werden durch die NoC-Parameter und den daraus generierten VHDL-Code beschrieben. Für die Generierung wird eine VHDL-Bibliothek benötigt, die für diese Arbeit erstellt wurde. Die Parameter, die für ein NoC vorgegeben werden können, sind in Anhang A angegeben und umfassen unter anderem die Topologie, Vermittlungstechnik, Datenwortbreite und Routing-Algorithmus. Neben der Beschreibung in VHDL werden die Elemente des NoC (Network-Interfaces und Routing-Switches) mit Hilfe von Standard-Zellen und physikalisch optimierten Kernkomponenten synthetisiert.

Eine Bewertung von NoC hinsichtlich der Performance und Kosten auf unterschiedlichen Abstraktionsebenen ist elementarer Bestandteile dieser Methodik. Um die Performancebewertung durchführen zu können, wurden verschiedene Analysemethoden (statischen Analyse, Simulation mit Petri-Netzen, SystemC und VHDL sowie FPGA-basierte Emulation) für NoC auf implementiert, analysiert und bewertet.

Die Bestimmung der Kosten in einer frühen Entwurfsphase basiert auf mathematischen Kostenmodellen für die einzelnen NoC-Komponenten. Diese wurde in umfangreichen Untersuchungen für die in Anhang A angegeben Parameter erstellt. Die Modelle unterstützen eine Implementierung mit Standardzellen sowie mit Standardzellen in Kombination mit physikalisch optimierten Kernkomponenten. Die Kosten eines NoC werden durch die benötigte Fläche und die Verlustleistung beschrieben.

Diese Entwurfsmethode wurde automatisiert, d.h. die einzelnen Phasen werden automatisch durchgeführt (inkl. Emulation und Synthese). Durch eine manuelle Optimierung kann eine weitere Reduzierung der Kosten bzw. Steigerung der Performance erzielt werden.

Die in dieser Arbeit vorgeschlagene, automatisierte Methodik unterstützt symmetrische NoC-Topologien. Mit Hilfe der VHDL-Bibliothek lassen sich auch asymmetrische und hierarchische Topologien abbilden. Diese werden nicht in der automatischen Entwurfs-methodik berücksichtigt. Durch den modularen Aufbau des NoC und der NoC-Komponenten kann die VHDL Beschreibung um bisher nicht unterstützte Parameter und Funktionen erweitert werden.

Für die Kostenmodellierung und Synthese wird in der Entwurfsmethode nur eine Technologie verwendet. Für eine Übertragung der Methode auf weitere Technologien müssen die Kostenmodelle überarbeitet werden. In erster Näherung können diese skaliert werden, für eine höhere Genauigkeit müssen die Modelle jedoch neu erstellt werden. Die physikalisch optimierten Kernkomponenten sind ebenfalls technologiespezifisch und müssen dementsprechend bei einem Wechsel der Technologie neu erstellt werden.

Diese Arbeit ist wie folgt gegliedert: Zunächst wird eine Übersicht über den Stand der Forschung bei NoC gegeben. In Kapitel 3 wird die erarbeitete, emulatorgestützte Entwurfsmethodik vorgestellt. Dabei wird insbesondere auf die verschiedenen Möglichkeiten der Performance-Analyse eingegangen. Das modulare und generische NoC, auf dem diese Entwurfsmethodik basiert, wird in Kapitel 4 vorgestellt. Weiterhin wird anhand exemplarischer Implementierungen der Einfluss verschiedener NoC-spezifischer Parameter auf die Kosten und Performance eines NoCs quantitativ untersucht. Im Anschluss werden Implementierungen des Routing-Switches mit Referenz-Implementierungen aus der Literatur verglichen.

Kapitel 5 beschreibt die Modellierungsmethodik, anhand derer Modellfunktionen für die von NoC verursachten Kosten bestimmt wurden. Mit Hilfe der in diesem Kapitel angegebenen Modellfunktionen wird der Entwurfsraum bezüglich der Verlustleistung, Siliziumfläche und Performance für zufälligen Datenverkehr aufgespannt. Im Kapitel 6 werden anwendungs-spezifisch optimierte NoC mit Hilfe der zuvor beschriebenen Entwurfsmethodik für reale Systeme aus dem Bereich der Video-Signal-Verarbeitung erzeugt und anschließend mit optimierten NoC aus der Literatur verglichen. Weiterhin wird die Leistungsfähigkeit der Methodik an exemplarischen, synthetischen Kommunikationsanforderungen demonstriert. Die Anwendung der FPGA-basierten Emulation eines NoC für ein reales Beispiel (Histograms of Oriented Gradients Algorithmus für Fahrassistenz-Systeme) und der Vergleich mit einer Implementierung auf einem Mehrkern-Prozessor ist in Kapitel 7 beschrieben. Die Arbeit schließt mit einer Zusammenfassung.

2 Network-on-Chip (NoC)

Networks-on-Chip ermöglichen eine skalierbare, flexible Kommunikation auf einem Chip und werden aus unterschiedlichen Modulen aufgebaut: Network-Interfaces, welche die Anbindung an eine funktionale Einheit vornehmen, Routing-Switches, die Daten durch das NoC verteilen und Links, welche die physikalischen Verbindungen in einem NoC darstellen. Das NoC, wie auch die einzelnen NoC-Komponenten, besitzen dabei eine Vielzahl von Parametern.

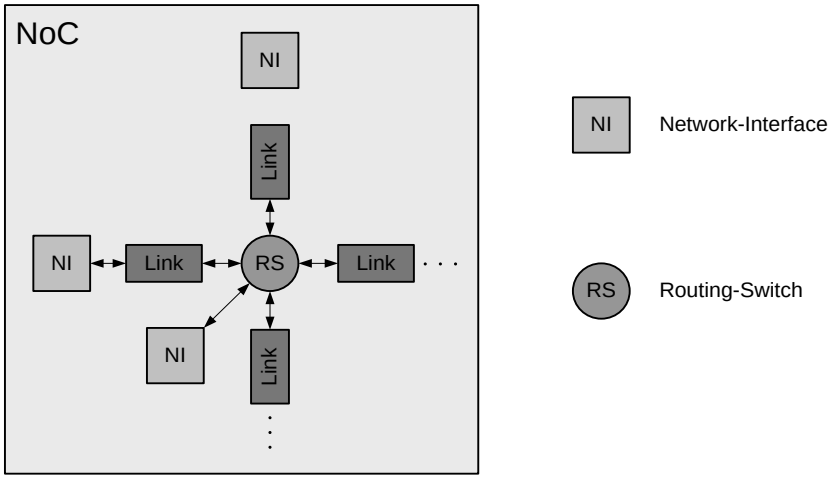


Abbildung 2.1: Allgemeiner Aufbau von NoC

Zur Strukturierung dieser Vielzahl von Parametern lassen sich die einzelnen NoC-Komponenten und deren Funktionen in das ISO/OSI-Referenzmodell einordnen [4,13]. Parameter, die einen signifikanten Einfluss auf die Leistungsfähigkeit bzw. die durch ein NoC verursachten Kosten haben, werden im Verlauf des Kapitels erläutert. Ein weiteres Unterkapitel behandelt Bewertungskriterien für NoC. Ausgehend von diesen Betrachtungen wird auf die Entwurfsproblematik von NoC eingegangen.

2.1 Einordnung in das ISO/OSI-Referenzmodell

Das ISO/OSI-Referenzmodell dient dazu, den Informationsaustausch zwischen verteilten Teilnehmern zu schematisieren und in einzelne, präzise definierte Schichten (Layer) zu unterteilen. Jede dieser Schichten bietet der nächsthöheren Schicht Dienste (Services) an. In Abbildung 2.2 ist die Kommunikation zwischen zwei Anwendungen, die in verschiedenen funktionalen Einheiten unterteilt werden, mittels des ISO/OSI-Referenzmodells beschrieben. Die Kommunikation erfolgt dabei über ein NoC unter Zuhilfenahme eines Routing-Switches. Die logische Informationsübertragung (gestrichelte Pfeile) erfolgt immer zwischen zwei Instanzen derselben Schicht über Protokolle. Dazu stehen der übertragenden Schicht lediglich

die Dienste der nächst niedrigeren Schicht zur Verfügung. Der Informationsfluss (grauer Pfeil) erfolgt unter Verwendung der darunter liegenden Schichten.

Es wird zwischen den netzabhängigen Schichten 1-3 und den netzunabhängigen Schichten 5-7 unterschieden. Die Schicht 4 stellt das Bindeglied zwischen den netzabhängigen und den netzunabhängigen Schichten dar. Dieses allgemeine Modell wird im Folgenden für NoC angepasst.

Um den Parameterraum von NoC zu strukturieren, wurden zunächst Dienste und die dazugehörigen Parameter den einzelnen netzabhängigen Schichten zugeordnet. Die Funktionen der netzunabhängigen Schichten werden im Folgenden skizziert.

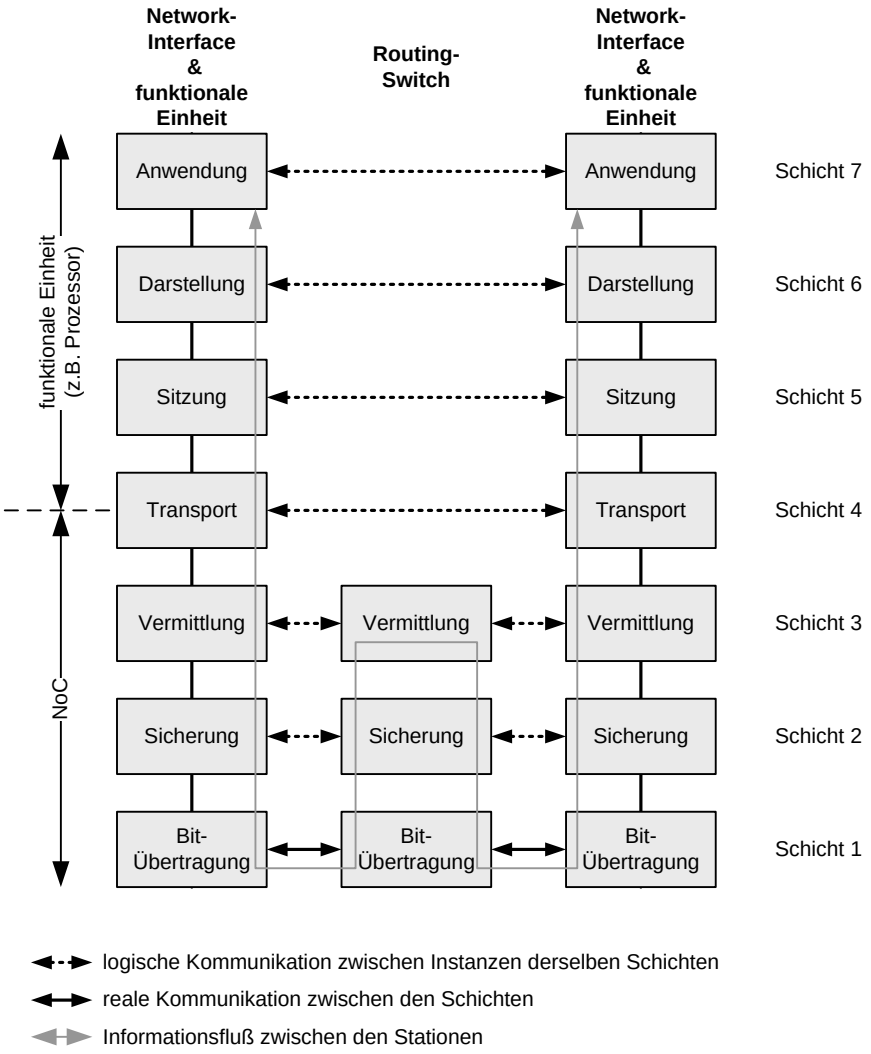


Abbildung 2.2: ISO/OSI-Referenzmodell

2.1.1 Physikalische Schicht

Die Bitübertragungsschicht, oder auch physikalische Schicht, ermöglicht die Übertragung von Bits über ein Kommunikationsmedium. Hier werden die funktionalen und elektrischen

Schnittstellen zum Übertragungsmedium definiert, wie z.B. die Buscodierung, die Taktfrequenz oder die Wortbreite der zu übertragenden Daten.

2.1.2 Sicherungsschicht

Die Aufgaben der Sicherungsschicht beinhalten sowohl Funktionen zum Verbindungsaufbau, das Zugriffsprotokoll auf das Medium sowie Maßnahmen zum Fehlerschutz.

Im Detail sind dies für NoC die Vermittlungstechnik, z.B. Leitungsvermittlung oder Paketvermittlung, der Datenfehlerschutz sowie weitere Dienste zur Unterstützung von QoS-Diensten höherer Schichten.

2.1.3 Vermittlungsschicht

Die Vermittlungsschicht ist für das Einrichten, den Betrieb und das Auflösen von Netzwerkverbindungen verantwortlich. Eine wichtige Aufgabe ist hier das Routing von Daten über das NoC.

2.1.4 Transportschicht

In der Transportschicht wird der End to End-Datentransport realisiert, d.h. es werden Datentransfers initiiert und beendet und somit der Datenfluss gesteuert. Die Fehlerbehandlung, die Realisierung von QoS und die Zuordnung von NoC-Adresse und virtueller Adresse finden in dieser Schicht statt.

2.1.5 Sitzungsschicht

Die Sitzungsschicht sorgt für die Prozesskommunikation zwischen zwei Systemen. In dieser Schicht werden Dienste für einen organisierten und synchronisierten Datenaustausch zur Verfügung gestellt.

2.1.6 Darstellungsschicht

Die Darstellungsschicht übersetzt Daten von einer systemabhängigen Darstellung in eine systemunabhängige Darstellungsform. Dabei können auch Dienste wie Verschlüsselung und Datenkompression verwendet werden.

2.1.7 Anwendungsschicht

Die Anwendungsschicht oder auch Verarbeitungsschicht ist die oberste der sieben hierarchischen Schichten und stellt der Anwendungssoftware eine Vielzahl von Funktionalitäten zur Verfügung. Die eigentliche Anwendung liegt über der Schicht 7 und wird vom ISO/OSI-Referenzmodell nicht erfasst.

2.2 NoC-Parameterraum

Den im vorherigen Abschnitt vorgestellten Schichten des ISO/OSI-Referenzmodells lassen sich NoC-spezifische Parameter zuordnen, die im weiteren Kapitel genauer erläutert werden. Die NoC-spezifischen Parameter beeinflussen nicht immer die Implementierung aller NoC-Komponenten. So hat beispielsweise der verwendete Routing-Algorithmus keinen Einfluss auf die Implementierung eines Network-Interfaces, da die Logik des Routing-Algorithmus komplett in den Routing-Switches implementiert ist.

2.2.1 Technologie

Ein wichtiger Parameter, der starken Einfluss auf die Leistungsfähigkeit und die durch NoC verursachten Kosten hat, ist die verwendete Technologie. Aktuell werden SoC, wie auch die zur Kommunikation verwendeten NoC in CMOS-Technologie gefertigt. Die Strukturgröße der verwendeten Technologien sinkt dabei stetig [15]. Die Technologie beeinflusst zum einen die Implementierung der Routing-Switches und Network-Interfaces, und damit insbesondere die erreichbare Taktfrequenz, die benötigte Fläche und die Verlustleistung. Zum anderen sind die Eigenschaften der elektrischen Verbindungsleitungen stark von der Technologie und der Strukturgröße abhängig.

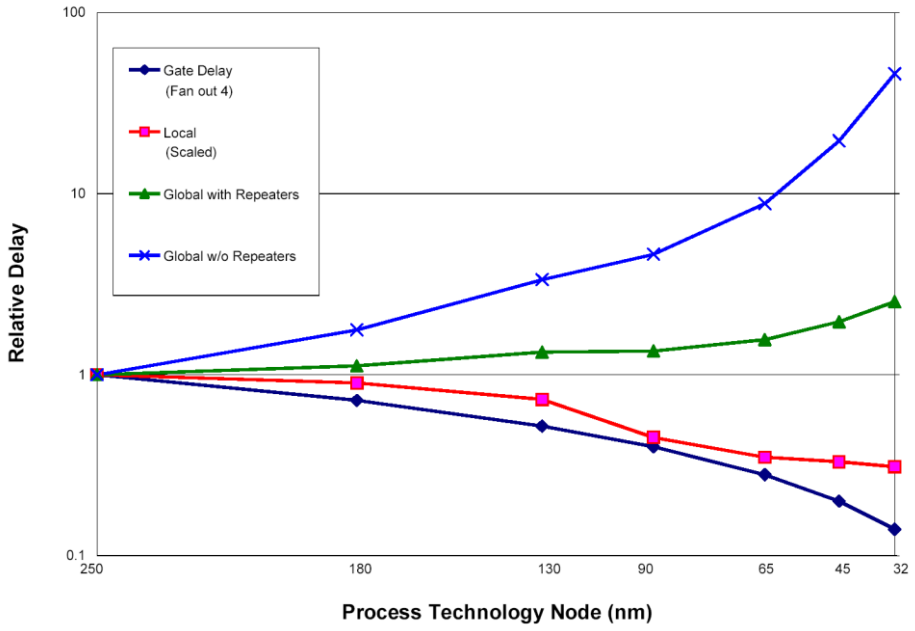


Abbildung 2.3: Relative Verzögerung auf Verbindungsleitungen für verschiedene Strukturgrößen [15]

Sinkende Strukturgrößen führen zu einer größeren Verzögerung auf den Verbindungsleitungen (vgl. [8] und Kapitel 1). Andererseits wird die Verzögerung durch logische Gatter (Gate Delay) immer geringer, wie in Abbildung 2.3 dargestellt ist. Daher wird

in Zukunft im Deep-Submicron-Bereich nicht mehr die Verzögerung durch logische Gatter, sondern die Verzögerung durch Verbindungsleitungen die Leistungsfähigkeit eines SoC bestimmen [16].

2.2.2 Implementierung der Links

Links verbinden NoC-Komponenten (Routing-Switches und Network-Interfaces) miteinander. Die Konnektivität der Links kann variiert werden. So kann ein Link, wie in Abbildung 2.4 abgebildet, aus ein oder zwei unidirektionalen Verbindungen bestehen, oder auch aus einer bidirektionalen Verbindung.

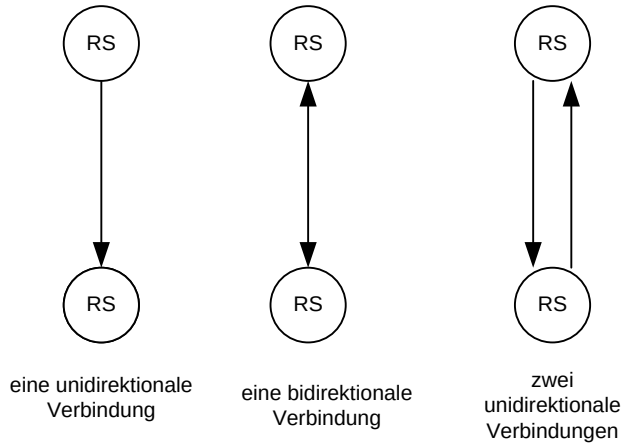


Abbildung 2.4: Konnektivität von Links

Die Verbindungen sind aus einem Satz von Verbindungsleistungen aufgebaut. Diese physikalischen Verbindungsleitungen besitzen verschiedene Parameter. Die Dimensionierung der für eine Verbindungsleitung benötigten Treiber ist abhängig von der Kapazität und dem elektrischen Widerstand der Verbindungsleitung. Diese Parameter werden zum einen durch die Länge der Verbindungsleitungen und deren Dimensionierung (Breite und Höhe der Leitung) wie auch durch den Abstand zu benachbarten Leitungen bestimmt. Die Kapazität C_{total} setzt sich dabei aus der Kapazität gegenüber dem Null-Potenzial C_{ground} und den Kapazitäten zu benachbarten Metallleitungen C_m zusammen (Abbildung 2.5). Die Kapazität C_m beeinflusst dabei nicht nur die die Kapazität der Leitung, sondern ist auch ein Maß für das Übersprechen zwischen den Leitungen (Cross-Talk). Je größer die Kapazität C_m ist, desto stärker beeinflussen sich die beiden Leitungen [18].

Wenn das Übersprechen zwischen den Leitungen minimiert wird, wird gleichzeitig die Verlässlichkeit der Datenübertragung erhöht. Dazu kann zum einen der Abstand zwischen den Leitungen erhöht, oder die Leitungen elektrisch entkoppelt werden. Die Entkopplung zweier Leitungen kann durch eine Leitung, die konstant auf dem Null-Potenzial liegt, erfolgen. Beide Maßnahmen resultieren jedoch in einem höheren Flächenbedarf für die Verbindungsleitungen.

Insbesondere bei langen Verbindungsleitungen und hohen Taktfrequenzen können zur Einhaltung der Timing-Vorgaben weitere Maßnahmen wie beispielsweise Repeater oder regenerative Booster eingesetzte werden [19]. Häufig werden auch Register vorgeschlagen, um so die Taktfrequenz des NoC weiter zu erhöhen [20].

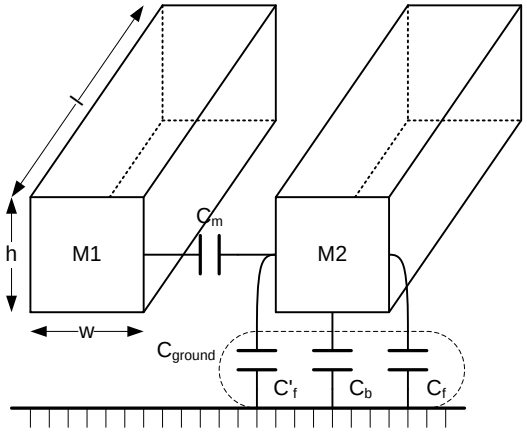


Abbildung 2.5: Einfaches Kapazitätsmodell [18]

2.2.3 Maximale Datenrate

Die maximale Datenrate kann auf eine einzelne NoC-Komponente oder auf das gesamte NoC bezogen sein. In beiden Fällen muss zwischen der Brutto- und Nettodatenrate unterschieden werden. Die Bruttodatenrate misst dabei alle über das NoC bzw. den Link übertragenen Daten pro Zeiteinheit, inklusive aller Kontrolldaten und hinzugefügter Redundanz, z.B. durch Fehlerschutzcodierung. Bei der Nettodatenrate werden nur die übertragenen Nutzdaten pro Zeiteinheit gezählt. Die maximal zu erreichende Nettodatenrate ist rechnerisch nicht exakt zu bestimmen. Sie hängt unter anderem von den hinzugefügten Kontrollinformationen, fehlgeschlagenen Verbindungsaufbauten bzw. Datenübertragungen, von der Auslastung des NoC und weiteren Faktoren ab. Zur Bestimmung der Nettodatenrate ist eine Simulation bzw. Emulation des NoC für einen definierten Datenverkehr notwendig. Dies ist in Kapitel 3.6 beschrieben.

Die maximale Bruttodatenrate ($r_{L,brutto,max}$) eines Links bestimmt sich hingegen aus der Bruttodatenwortbreite (w') eines Links multipliziert mit der maximalen Taktfrequenz (f_{max}), mit der die Datenworte über den Link gesendet werden. Die maximale Taktfrequenz wird neben der verwendeten Technologie auch von der Implementierung und Dimensionierung der NoC-Komponenten begrenzt.

$$r_{L,brutto,max} = f_{max} \cdot w' \tag{2.1}$$

Um die maximale Bruttodatenrate des gesamten NoCs zu bestimmen, wird davon ausgegangen, dass von jedem Network-Interface des NoC gleichzeitig mit der maximal möglichen Datenrate, der maximalen Bruttodatenrate einer NoC-Komponente, gesendet wird.

$$r_{brutto,max} = \sum_{i=1}^{NI} r_{L,brutto,max} \tag{2.2}$$

Die maximale Bruttodatenrate ist dabei jedoch ein Wert, der in der Regel nicht erreicht wird. Er wird zum einen durch zusätzlichen Aufwand für Kontrolldaten und zum anderen durch Überlastung einzelner NoC-Komponenten beschränkt.

2.2.4 Datenfehlerschutz

Durch die immer weitere Technologieskalierung werden Verbindungsleitungen immer sensitiver für Fehler [15]. Diese Fehler lassen sich in die Fehlerklassen transiente Fehler, permanente Fehler und alterungsbedingte Fehler aufteilen. Transiente Fehler werden beispielsweise durch Übersprechen zwischen Verbindungsleitungen, Rauschen der Versorgungsspannung oder durch Leckströme initiiertes Rauschen hervorgerufen. Permanente Fehler können durch Prozessvariationen verursacht werden. Durch Elektromigration und den Hot Carrier-Effekt werden Fehler durch Alterung verursacht [18]. Den größten Einfluss bei NoC haben transiente Fehler durch Übersprechen zwischen Verbindungsleitungen [21].

Um diese Fehler auf den Verbindungsleitungen zu vermeiden oder zu korrigieren, wurden in der Literatur verschiedene Verfahren vorgestellt. Sie lassen sich grundsätzlich in die Fehlervermeidung, Fehlererkennung und -korrektur sowie Fehlerbehandlung unterteilen.

Zur Vermeidung von Fehlern durch Übersprechen werden beispielsweise spezielle Formen von Datencodierung verwendet. Diese nutzen Datenmuster auf den Verbindungsleitungen welche ein starkes Übersprechen vermeiden [22, 23].

Tabelle 2.1: Übersicht Fehlerschutzcodes

Abkürzung	**Funktion**	**Fehlerschutzcode**
SED	Einzelfehlererkennung	Parity Code
SEC	Einzelfehlerkorrektur	Hamming Code
DED	Doppelfehlererkennung	Hamming Code
SEC/DED	Einzelfehlerkorrektur und Doppelfehlererkennung	Enhanced Hamming Code
TED	Dreifachfehlererkennung	Enhanced Hamming Code

Die Fehlererkennung und -korrektur wird mit Hilfe von Codes ermöglicht, die den Daten Redundanz hinzufügen. Durch das Hinzufügen von Redundanz lassen sich so einzelne oder auch mehrere Fehler erkennen und ggf. korrigieren. Die Anzahl der erkennbaren und korrigierbaren Fehler hängt von dem verwendeten Code ab. Bei NoC werden vielfach Parity und Hamming Codes verwendet [24, 25]. Zur Erkennung von 1-, 2- oder 3-Bitfehlern kann

ein Parity, Hamming bzw. Enhanced Hamming Code verwendet werden. Zur Fehlerkorrektur eignet sich der Hamming Code bei 1-Bitfehlern und der Enhanced Hamming Code bei 2-Bitfehlern, vgl. Tabelle 2.1. Die Anzahl der zusätzlich zu übertragenden Bits ist, sowohl vom Code als auch von der Wortbreite abhängig. Dies ist in Tabelle 2.2 dargestellt.

Die Fehlerschutzcodierung erfolgt im sendenden Network-Interface. Die Fehlererkennung und -korrektur kann sowohl im empfangenden Network-Interface wie auch in den Routing-Switches erfolgen.

Sollte eine Fehlerkorrektur nicht möglich sein, müssen die nichtkorrigierbaren Fehler behandelt werden. Dabei werden drei Möglichkeiten unterschieden: Send and Forget, bei der auftretende Fehler akzeptiert werden, sowie Send and Wait und Send and Wait mit Sliding Window Enhancement. Bei den letzteren Protokollen werden fehlerhaft übertragene Daten erneut übertragen. Bei dem Send and Wait-Protokoll wird nach dem Senden eines Datenwortes die Bestätigung des korrekten Empfangs dieses Datenwortes (*ackn*) abgewartet bevor das nächste Datenwort gesendet wird. Bleibt die Bestätigung innerhalb einer festgelegten Zeitspanne (*time to acknowledge*) aus, wird das Datenwort erneut übertragen.

Tabelle 2.2: Paritätsbits für Fehlerschutzcodes in Abhängigkeit von der Bruttodatenwortbreite

Bruttodatenwortbreite	**Anzahl der Paritätsbits**		
	Parity Code	**Hamming Code**	**Enhanced Hamming Code**
5 ... 11	1	5	6
12 ... 26	1	6	7
27 ... 57	1	7	8
58 ... 120	1	8	9
121 ... 246	1	9	10

Durch dieses Vorgehen wird die maximal erzielbare Datenrate stark reduziert. Um dies zu vermeiden, wird hier das Send and Wait-Protokoll mit Sliding Window Enhancement verwendet. Bei diesem Protokoll muss nicht auf die Bestätigung eines einzelnen Datenwortes gewartet werden, sondern die Bestätigung muss innerhalb einer Zeitintervalls (*Sliding Window Size*) erfolgen, bevor die Datenübertragung angehalten wird. Dadurch können die Daten kontinuierlich übertragen werden solange keine nichtkorrigierbaren Fehler auftreten. Tritt ein nichtkorrigierbarer Fehler auf, müssen das fehlerhaft übertragene Datenwort sowie alle auf dieses Datenwort folgenden Datenworte erneut übertragen werden. Aus diesem Grund müssen alle noch nicht bestätigten Datenworte im sendenden Network-Interface gespeichert werden.

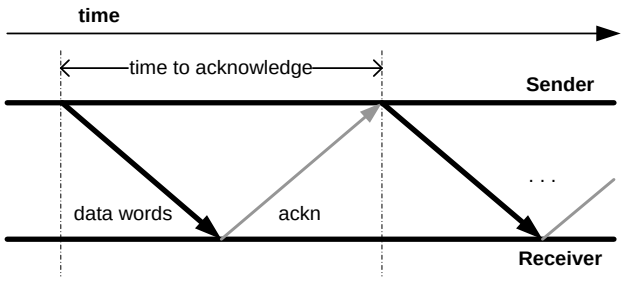


Abbildung 2.6: Ablauf des Send and Wait-Protokolls

Die Anzahl der Datenworte, die ohne den Erhalt einer Bestätigung gesendet werden können, wird durch das Zeitintervall Sliding Window Size definiert, welches auch die Größe des Speichers im Network-Interface bestimmt. Wenn die Sliding Window Size größer oder gleich der Zeit ist, die für eine Bestätigung benötigt wird, treten keine Leistungseinbußen bei einer fehlerfreien Datenübertragung auf.

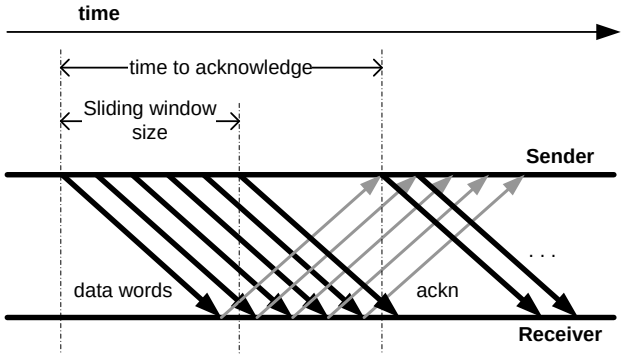


Abbildung 2.7: Ablauf des Send and Wait-Protokolls mit Sliding Window Enhancement

2.2.5 Vermittlungstechnik

Die Vermittlungstechnik definiert, wie Daten über das NoC transportiert werden. Sie hat einen großen Einfluss auf die Performance eines NoC wie auch auf die verursachten Kosten. Häufig in NoC verwendete Vermittlungstechniken sind Leitungsvermittlung und Paketvermittlung. Bei der Leitungsvermittlung wird eine physikalische Verbindung zwischen dem Sender und dem Empfänger aufgebaut, über welche anschließend die Daten übertragen werden [26]. Da bei der Leitungsvermittlung keine Daten innerhalb der Routing-Switches zwischengespeichert werden müssen, kann auf einen Speicher in den Routing-Switches verzichtet werden.

Bei der Paketvermittlung werden die Daten in Datenpakete fester Größe aufgeteilt und mit Steuerinformationen erweitert. Jedes dieser Datenpakete wird individuell über das Netz übertragen. Da die Datenpakete auf unterschiedlichen Wegen von einer Quelle zu einem Ziel übertragen werden können, ist die Reihenfolge, in der die Datenpakete beim Empfänger ankommen, nicht zwangsläufig die Reihenfolge, in der sie abgeschickt wurden. Diese muss im

Network-Interface wiederhergestellt werden. Bei Paketvermittlung werden drei Varianten unterschieden, die sich durch die Latenz und den benötigten Speicher innerhalb der Routing-Switches unterscheiden:

Beim Paketvermittlungsprotokoll *Store and Forward* [26, 27] wird ein Datenpaket vom Routing-Switch zunächst komplett empfangen, bevor es an den nächsten Routing-Switch weitergeleitet wird. Daraus resultiert eine Latenz in jedem Routing-Switch, die proportional zur Paketgröße ist. Außerdem benötigt jeder Routing-Switch einen Speicher, der mindestens ein Datenpaket zwischenspeichern kann.

Eine Weiterentwicklung ist das *Virtual cut through-Protokoll* [26]. Das in einem Routing-Switch empfangene Datenpaket wird, sobald der Kopf mit den Steuerinformationen empfangen wurde, an den nächsten Routing-Switch weitergeleitet. Dieser Routing-Switch empfängt das Datenpaket nur, wenn das komplette Datenpaket auch zwischengespeichert werden kann. Dadurch ist die Latenz, die innerhalb eines Routing-Switches verursacht wird, nur noch von der Größe der Steuerinformationen abhängig. Der Speicher des Routing-Switches muss aber immer noch ein komplettes Datenpaket speichern können.

Ein drittes, und bei NoC sehr verbreitetes Protokoll, ist das *Wormhole*-Protokoll [28]. Dabei wird das Datenpaket in sogenannte Flits aufgeteilt. Das erste Flit enthält dabei die Steuerinformationen, die benötigt werden, um das Datenpaket über das Netzwerk zu übertragen. Die restlichen Flits, die nur noch minimale Steuerinformationen enthalten, folgen dabei immer dem ersten Flit. Neben der, wie beim *Virtual cut through*-Protokoll, geringen Latenz wird hier zusätzlich der innerhalb der Routing-Switches benötigte Speicher reduziert, da minimal ein Flit gespeichert werden muss.

Die jeweiligen Vermittlungstechniken weisen verschiedene Vor- und Nachteile auf: Bei der Leitungsvermittlung ist die auftretende Latenz, sobald die Verbindung aufgebaut ist, minimal. Auch die Latenz, die durch den Verbindungsaufbau entsteht, ist bei kleinen bis mittleren Netzwerken vergleichbar mit der, die bei Verwendung des *Wormhole*- oder des *Virtual cut through*-Protokolls entsteht.

Der Speiche im Routing-Switch nimmt einen signifikanten Teil der Fläche des Routing-Switches ein. Die Größe des Speichers bestimmt auch die Leistungsfähigkeit. Neben der Größe des Speichers beeinflusst die Organisation und die Verteilung des Speichers die Leistungsfähigkeit des Routing-Switches [29, 30]. Allein durch eine Vergrößerung des Speichers ist eine Blockade des Routing-Switches durch Überlastung des Speichers nicht zu vermeiden [27]. Eine optimale Dimensionierung und Organisation des Speichers in den Routing-Switche ist daher unerlässlich [31].

Über ein Netzwerk, das Paketvermittlung verwendet, könne in der Regel mehr Daten pro Zeiteinheit übertragen werden, als über ein identisches Netzwerk, in dem Leitungsvermittlung

eingesetzt wird. Dies liegt an der besseren Ausnutzung der Netzwerkressourcen. Diese werden bei der Paketvermittlung nur belegt, wenn Daten übertragen werden. Im Gegensatz dazu ist eine Netzwerkressource bei Leitungsvermittlung belegt, sobald eine Verbindung aufgebaut ist, unabhängig davon, ob Daten übertragen werden [26].

Die Art des Datenverkehrs hat auch eine Auswirkung auf die Wahl einer optimalen Vermittlungstechnik. Wenn vornehmlich große, zusammenhängende Datenmengen übertragen werden sollen, ist die Leitungsvermittlung vorteilhaft, sind es jedoch viele kleine Datenblöcke, kann das Netzwerk besser durch Paketvermittlung ausgenutzt werden.

Bei Leitungsvermittlung können, wenn eine Verbindung bereits aufgebaut ist, Quality-of-Service-Dienste wie garantierte Bandbreite oder garantierte Antwortzeiten sicher eingehalten werden. Bei Paketvermittlung ist dies nur durch das Einführen weiter Maßnahmen, wie zum Beispiel virtueller Kanäle (Virtual Channels), möglich [32, 33]. Eine weitere Möglichkeit einen garantierten Service bei einer gleichzeitig hohen Auslastung sicher zu stellen, ist die Anwendung einer Kombination von Time-Division Multiplex (TDM) und Space-Division Multiplex, wie sie in [36] für NoC untersucht wurde.

Im Gegensatz zu Prozessornetzwerken, in denen sehr häufig das Store and Forward-Protokoll verwendet wird, wird es in NoC wegen der hohen auftretenden Latenz und des hohen Speicherbedarfs im Routing-Switches selten eingesetzt [27]. Um dennoch eine möglichst hohe Auslastung der NoC-Ressourcen zu erreichen, werden häufig Wormhole- oder Virtual Cut Through-Protokolle verwendet [12]. Leitungsvermittlung wird vornehmlich in NoCs verwendet, die Datenverkehr mit einer garantierten Dienstgüte übertragen müssen.

2.2.6 Priorisierung von Daten und präemptiver Zugriff auf NoC-Ressourcen

Damit die in Schicht 4 des ISO/OSI-Referenzmodells erwähnten QoS-Dienste realisiert werden können, müssen die darunterliegenden Schichten entsprechende Dienste bereitstellen, welche dies ermöglichen. Eine Möglichkeit garantierte Antwortzeiten als QoS-Aspekt zu realisieren, ist die Verwendung von priorisierten Daten.

Dabei besitzen die über das Netz zu übertragenden Daten eine Priorität, die über den Zugriff auf Netzwerkressourcen entscheidet. Diese Priorität wird bei der in Kapitel 2.2.5 vorgestellten Leitungsvermittlung an die Verbindung, die aufgebaut werden soll, bzw. die aus den Daten gebildeten Datenpakete, vererbt. Wenn eine Netzwerkressource bereits von einer Verbindung mit einer niedrigeren Priorität belegt ist, besteht bei einem präemptiven Zugriff auf diese Netzwerkressource die Möglichkeit, die bestehende Verbindung abzubrechen und den Zugriff auf diese Ressource der Verbindung mit der höheren Priorität zu gestatten.

2.2.7 Routing-Algorithmus

Der Routing-Algorithmus legt den Pfad im Netzwerk fest, auf dem die Daten von einer Quelle zu einem Ziel übertragen werden. Dabei muss der verwendete Routing-Algorithmus an die verwendete Netzwerktopologie angepasst sein. Es müssen Daten von jeder Quelle zu jedem Ziel im Netzwerk übertragen werden können. Dabei sollen die zur Verfügung stehenden Netzwerkressourcen bei minimalem Aufwand optimal ausgenutzt werden. Zusätzlich muss sichergestellt werden, dass der Routing-Algorithmus in Kombination mit der verwendeten Vermittlungstechnik keine Deadlock bzw. Livelock Situation erzeugt.

Ein Deadlock, der in [34] für Netzwerke definiert wurde, tritt dann auf, wenn zwei Datenpakete gegenseitig auf das Freigeben einer Netzwerkressource warten, die jeweils von dem anderen Datenpaket belegt wird. Ein spezieller Deadlock ist der Livelock [26]. Dieser tritt zum Beispiel dann auf, wenn ein Datenpaket in einem Netzwerk weitergeleitet wird, ohne jemals das Ziel zu erreichen.

Es existieren drei Strategien Deadlocks zu behandeln: Prävention, Vermeidung und Behandlung [35]. Prävention ist eine konservative Methode, um Deadlocks zu vermeiden. Dabei werden Netzwerkressourcen nur blockiert, wenn dadurch kein Deadlock entstehen kann. Ein Beispiel ist die Leitungsvermittlung, bei der nach dem versuchten Zugriff auf eine bereits belegte Netzwerkressource, alle zuvor reservierten Netzwerkressourcen freigegeben werden. Bei der Vermeidung von Deadlocks werden die benötigten Netzwerk-Ressourcen direkt belegt, auch wenn dadurch potentiell ein Deadlock verursacht werden kann. Dieser muss durch geeignete Maßnahmen vermieden werden. Ein Beispiel ist das Wormhole-Protokoll. Hier werden Deadlocks durch spezielle Routing-Algorithmen [37] oder durch das Einfügen virtueller Kanäle [38] vermieden.

Eine allgemeine Übersicht und Klassifikation von Routing-Algorithmen wurden in [39] vorgestellt und in [26] erweitert (siehe Abbildung 2.8). Diese Einteilung lässt sich auf NoC übertragen. Routing-Algorithmen werden zunächst bezüglich der Anzahl der Verbindungsziele (Number of Destinations) in Unicast- und Multicast Routing-Algorithmen unterschieden. Bei Unicast Routing-Algorithmen werden die Daten nur zu einem Ziel weitergeleitet, wohingegen bei Multicast Routing-Algorithmen die Daten an mehrere Ziele gleichzeitig gesendet werden können.

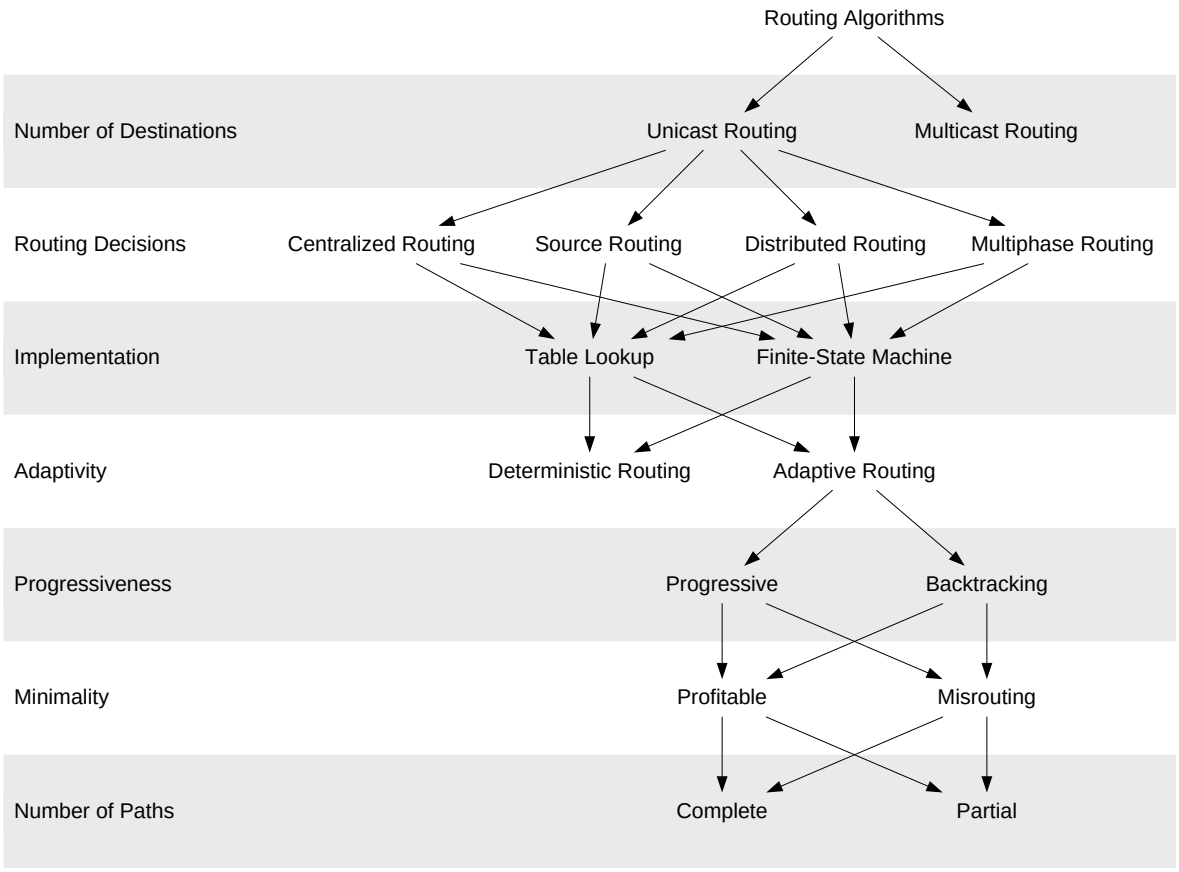


Abbildung 2.8: Allgemeine Klassifizierung von Routing-Algorithmen [26]

Im nächsten Schritt wird eine Unterteilung nach dem Ort vorgenommen, in dem die Routing-Entscheidung getroffen wird (Routing-Decisions). Beim Centralized Routing trifft eine zentrale Stelle die Routing-Entscheidung, beim Source Routing werden die Routing-Entscheidungen bereits beim Sender getroffen, wohingegen beim Distributed Routing die Routing-Entscheidung verteilt in jedem Routing-Switch getroffen wird. Das Multiphase Routing ist eine Mischform aus den zuvor genannten Möglichkeiten. Jede dieser Möglichkeiten kann beim Unicast-Routing verwendet werden.

Das dritte Klassifizierungskriterium ist die Art der Implementierung (Implementation) des Routing-Algorithmus. Er kann in Form einer Tabelle (Table Lookup) oder als endliche Zustandsmaschine (Finite-State Machine) realisiert werden.

Bei der Adaptivität (Adaptivity) eines Routing-Algorithmus unterscheidet man zwischen deterministischem Routing (Deterministic Routing), bei dem die Daten immer den gleichen Weg von einer Quelle zu einem Ziel durch das NoC nehmen. Bei einem adaptiven Routing-Algorithmus (Adaptive Routing) hingegen ist das Routing-Ergebnis auch vom aktuellen Zustand des Netzwerkes abhängig. Bei den deterministischen Routing-Algorithmen ist keine weitere Unterteilung möglich.

Ein weiteres Unterscheidungsmerkmal ist die Reaktion auf blockierte Netzwerkressourcen (Progressiveness). Bei progressiven Algorithmen wird entweder so lange gewartet, bis die benötigte Ressource wieder verfügbar ist (Delay Model), oder die bisher gesendeten Daten werden verworfen (Loss-Model) und müssen erneut übertragen werden (siehe auch Kapitel 2.2.9). Bei Routing-Algorithmen, die Backtracking unterstützen, wird der Weg, den die Daten genommen haben, zurückverfolgt. Bei jedem Routing-Switch, der auf dem Rückweg besucht wird, wird nach einem alternativen Weg zum Ziel gesucht.

Routing-Algorithmen, die Daten immer auf direktem Weg zu ihrem Ziel leiten, werden mit Profitable oder auch Minimal Path-Routing bezeichnet. Wenn aufgrund der Routing-Entscheidung die Daten auch auf Umwegen das Ziel erreichen, spricht man von Misrouting oder Non Minimal Path-Routing.

Das letzte Differenzierungsmerkmal bezieht sich auf die Anzahl der Pfade von einer Quelle zu einem Ziel, die durch einen Routing-Algorithmus erreicht werden können. Können alle möglichen Pfade erreicht werden, handelt es sich um einen Complete- oder auch Fully Adaptive Routing-Algorithmus. Ansonsten spricht man von einem Partial Adaptive Routing-Algorithmus.

2.2.8 Topologie

Ein wichtiger NoC-Parameter ist die gewählte Topologie, d.h. die Anordnung der Netzwerkressourcen untereinander. Sie hat großen Einfluss auf die Implementierungskosten und die Performance des NoC. Die meisten in NoC verwendeten Topologien wurden aus den Topologien von Multiprozessornetzwerken abgeleitet und für NoC adaptiert. Grundsätzlich lassen sich die Topologien in homogene Topologie wie z.B. Mesh, Torus, Stern (vgl. Abbildung 2.9, [40, 41, 42, 43]) und applikationsspezifische Topologien [44] unterscheiden.

Applikationsspezifische NoC-Topologien sind häufig hierarchisch aufgebaut, d.h. das NoC ist aus mehreren Sub-NoCs aufgebaut, die eine homogene Topologie besitzen. In Abbildung 2.10 ist eine exemplarische hierarchische NoC-Topologie dargestellt, die aus einem Mesh in der obersten Hierarchiestufe und einem Stern in der unteren Hierarchiestufe besteht.

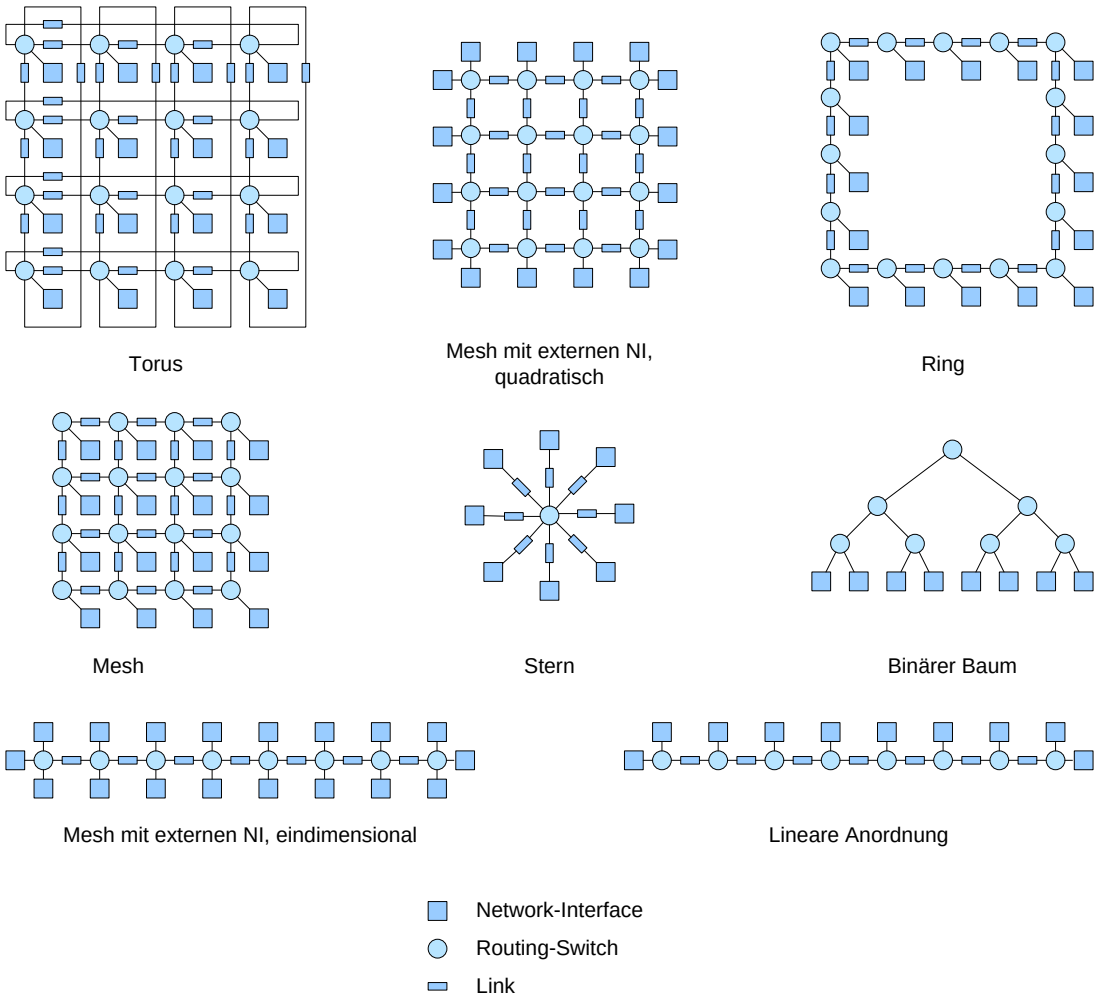


Abbildung 2.9: Homogene Topologien für NoC

Die Länge der Links hängt neben der Topologie maßgeblich von der Geometrie der funktionalen Einheiten, also dem resultierenden Layout des SoC, ab und muss bei der Topologie-Wahl des NoC berücksichtigt werden [40].

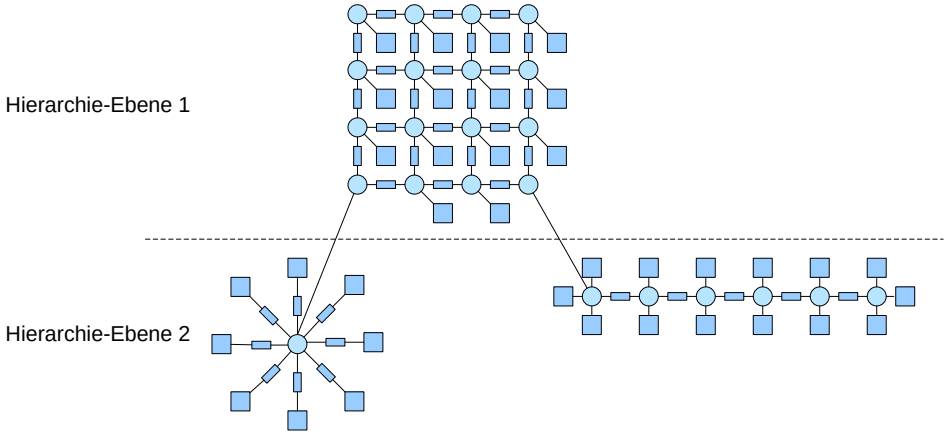


Abbildung 2.10: Exemplarische hierarchische Topologie

2.2.9 Verbindungsfehlerbehandlung

Verbindungsfehler treten immer dann auf, wenn eine Verbindung bei Leitungsvermittlung nicht aufgebaut bzw. ein Datenpaket nicht übertragen werden kann. Dies ist der Fall, wenn eine benötigte Netzwerkressource bereits belegt oder nicht bereit ist, Daten zu übernehmen. In diesem Kapitel wird erläutert, wie auf diese Verbindungsfehler reagiert wird. Wenn ein Verbindungsfehler erkannt wird, muss zunächst direkt entschieden werden, ob gewartet werden soll, bis die nicht zur Verfügung stehende Netzwerkressource wieder zur Verfügung steht (Wait-Model), oder ob der Verbindungsaufbau bzw. die Datenübertragung abgebrochen werden soll, und bereits gesendete Daten verworfen und blockierte Netzwerkressourcen wieder freigegeben werden sollen (Loss-Model).

Bei dem Wait-Model wird der Verbindungsaufbau/die Datenübertragung direkt an der Stelle fortgesetzt, an der der Fehler aufgetreten ist. Dadurch können der Verbindungsaufbau bzw. die Datenübertragung sehr schnell wieder aufgenommen werden. Allerdings sind bereits belegte Netzwerk-Ressourcen auch während der Wartezeit belegt und die Wahrscheinlichkeit von Deadlocks steigt.

Wenn das Loss-Modell angewendet wird, werden die belegten Netzwerkressourcen sofort wieder freigegeben. In dem sendenden Network-Interface kann dann entschieden werden, wie auf diesen Verbindungsfehler reagiert werden soll. Der Verbindungsaufbau bzw. die Datenübertragung können direkt, nach einer festen Wartezeit oder nach einer zufälligen Wartezeit neu initiiert werden. Währen der Wartezeit können die zunächst belegten Netzwerkressourcen auch für andere Datenübertragungen genutzt werden, so dass die Wahrscheinlichkeit von Deadlocks gering ist. Eine Vermeidung von Deadlocks lässt sich durch diese Maßnahmen nicht erreichen.

Für NoC, welche die in Kapitel 2.2.6 eingeführten Prioritäten unterstützen, besteht zusätzlich die Möglichkeit, die Priorität der Daten, die übertragen werden sollen, mit jedem erfolglosen Übertragungsversuch zu erhöhen. So lässt sich die maximal in einem NoC für eine Anwendung auftretende Latenz minimieren.

2.2.10 Quality-of-Service (QoS)

QoS wird als Dienst definiert, der von einem NoC einer anfordernden funktionalen Einheit bereitgestellt wird. Dies umfasst zwei Aspekte: die Definition des Dienstes und die Aushandlung des Dienstes. Die Aushandlung besteht dabei aus dem Ausgleichen der von der funktionalen Einheit angeforderten Dienste mit den, vom NoC bereitgestellten, Diensten. Ein Dienst kann dabei Aspekte wie garantierten Durchsatz, minimale Latenz, geringe Verlustleistung, Grenzen bezüglich Jitters, Datenfehlerfreiheit etc. umfassen [17].

Für NoC werden in der Literatur meistens zwei Klassen von QoS unterschieden, best effort services (BE) für Datenübertragungen die keine speziellen QoS-Anforderungen haben und guaranteed services (GS) für Übertragungen mit speziellen Anforderungen [45, 46]. Die speziellen QoS-Anforderungen werden in Korrektheit des Ergebnisses, Abschluss der Datenübertragung und Grenzen für Performance-Werte definiert. BE-Datenverkehr erhöht dabei die Auslastung des NoCs. GS-Datenverkehr steigert die Vorhersagbarkeit auf Kosten der Ausnutzung des NoC. Eine hohe Vorhersagbarkeit wird insbesondere bei Echtzeitsystemen benötigt [47]. Daher wird in [48] eine Kombination von BE- und GS-Datenverkehr gefordert.

Häufig wird GS-Datenverkehr aufgrund von statistischen Verteilungen garantiert. Diese werden als *soft* oder *statistischer* GS-Datenverkehr bezeichnet. Soll dagegen eine absolute Vorhersagbarkeit realisiert werden, spricht man von hartem GS-Datenverkehr.

Um GS-Datenverkehr zu realisieren, werden Dienste wie zum Beispiel der in Kapitel 2.2.4 vorgestellte Datenfehlerschutz, oder die in Kapitel 2.2.6 eingeführte Priorisierung von Daten benutzt. Auch die verwendete Vermittlungstechnik (Kapitel 2.2.5) hat Einfluss auf den QoS: paketbasierte Ansätze werden häufig bei BE-Datenverkehr verwendet, leitungsbasierte Techniken für GS-Datenverkehr. Eine weitere Möglichkeit, um Performance-Anforderungen zu erfüllen, insbesondere von hartem GS-Datenverkehr, sind virtuelle Kanäle. Sie können verwendet werden, um mit Hilfe von time division multiplexing eine verbindungsorientierte Paketvermittlung zu realisieren.

Beispiele für NoC, die soft GS-Datenverkehr unterstützen, sind QNoC [49], [32] und [50]. Eine Sonderstellung nimmt SoCBUS [51] ein. Hier werden Verbindungen aufgebaut, die grundsätzlich harten GS-Datenverkehr unterstützen. Jedoch erfolgt der Verbindungsaufbau als soft GS-Datenverkehr.

NoC die harten GS-Datenverkehr unterstützen sind AETHERAL [52], NOSTRUM [53], MANGO [54], SONICS [55], aSoC [56], sowie die in [57], [58] und [59] vorgestellten NoC.

2.3 Bewertungskriterien

Ein einheitliches Bewertungsschema, mit dem NoC verglichen werden können, ist nur in Grundzügen vorgeschlagen worden [60]. Um ein NoC bewerten zu können, müssen zunächst verschiedene Bewertungsmaße definiert werden. Diese Bewertungsmaße lassen sich grundsätzlich in Bewertungsmaße, welche die Leistungsfähigkeit eines NoC beurteilen, und Bewertungsmaße, welche die Kosten bewerten, die von dem NoC verursacht werden, unterteilen. Weitere Aspekte sind die Verlässlichkeit des NoC und die Einhaltung von QoS-Anforderungen. Je nach Anwendung und Vorgaben für das NoC können diese Kriterien unterschiedlich gewichtet werden.

2.3.1 Kosten

Die durch ein NoC verursachten Kosten werden durch die benötigte Fläche des NoC, die Verlustleistung und die benötigte Entwurfszeit ausgedrückt. Die Fläche eines NoC lässt sich aus dem Layout des NoC bestimmen. Die Verlustleistungsaufnahme ist jedoch dynamisch und hängt auch von der Auslastung des NoC und den übertragenen Daten ab. Neben diesen Kosten, die sich quantitativ erfassen lassen, gibt es weitere, "weiche" Kostenfaktoren. Dazu zählen beispielsweise die Entwicklungszeit oder die Erfahrung, die ein Entwickler für die Implementierung eines NoCs benötigt.

2.3.2 Performance

Um zu belegen, dass die von einer Anwendung an das NoC gestellten Kommunikationsanforderungen erfüllt werden, muss die Performance des NoC bestimmt werden. Dies kann auf unterschiedliche Arten erfolgen, welche sich sowohl in ihrer Genauigkeit wie auch in dem Aufwand zur Bestimmung der Performance unterscheiden.

Die genaueste Möglichkeit, die Performance eines NoC zu bestimmen, ist die Ausführung einer Anwendung auf einem SoC. Dazu müssen sowohl die funktionalen Einheiten wie auch gegebenenfalls die Software für die funktionalen Einheiten auf dem SoC implementiert, und das SoC gefertigt sein.

Eine weitere Möglichkeit das NoC innerhalb des SoC zu testen, ohne das SoC zu fertigen, ist die Emulation bzw. Simulation des gesamten SoC. Die Simulation des gesamten SoC ist in der Regel zu langsam, um komplette Anwendungen auszuführen. Bei der Emulation eines SoC werden schnell die Kapazitätsgrenzen der meist zur Emulation verwendeten FPGAs erreicht, oder es werden aufwändige Emulationsumgebungen [61] benötigt. Ein weiteres Problem bei diesem Vorgehen ist, dass das komplette SoC, wie auch die benötigte Software, sowohl bei der Emulation wie auch bei der Simulation bereits implementiert sein muss.

Um dies Problem zu lösen, lassen sich die funktionalen Einheiten des SoC abstrahieren. So kann anstelle des SoC lediglich das NoC als Teil des SoC untersucht werden. Der Datenverkehr, der von den funktionalen Einheiten erzeugt wird, muss in diesem Fall modelliert werden. Dadurch lässt sich das NoC simulieren bzw. auf einem FPGA emulieren. Die verschiedenen Möglichkeiten zur Performance-Bestimmung werden in Kapitel 3.4 und Kapitel 3.6 vorgestellt und bewertet. Ein weiterer Vorteil der Abstraktion der funktionalen Einheiten ist, dass sowohl die funktionalen Einheiten des SoC wie auch die Software, die auf den funktionalen Einheiten des SoC ausgeführt werden soll, noch nicht implementiert sein müssen. Dadurch kann bereits in einem frühen Entwurfsstadium die Performance von NoC bestimmt werden.

Der auf einem NoC durch die funktionalen Einheiten erzeugte Datenverkehr einer Anwendung kann in Form eines sogenannten Task-Graphen beschrieben werden [62]. Dieser, in Abbildung 2.11 exemplarisch dargestellte, gerichtete Graph $G_{TG}(V,A)$ besteht aus einer Menge Knoten V, die Prozesse repräsentieren, und aus einer Menge Kanten A, die den Datenverkehr zwischen zwei Prozessen darstellen. Jeder Knoten v_i besitzt eine definierte Ausführungszeit $t_{A,i}$. Optional kann einem Knoten eine Deadline zugewiesen werden. Dabei handelt es sich um einen Zeitpunkt, zu dem der zugehörige Prozess abgearbeitet sein muss. Die Kante $a_{i,j}$ vom Knoten v_i zum Knoten v_j werden mit einer Datenmenge D gewichtet, die von dem Startknoten zu dem Zielknoten übertragen werden soll. Mit Hilfe eines solchen Task-Graphen lassen sich die Kommunikations-Anforderungen darstellen, die eine auf ein SoC abgebildete Anwendung besitzt.

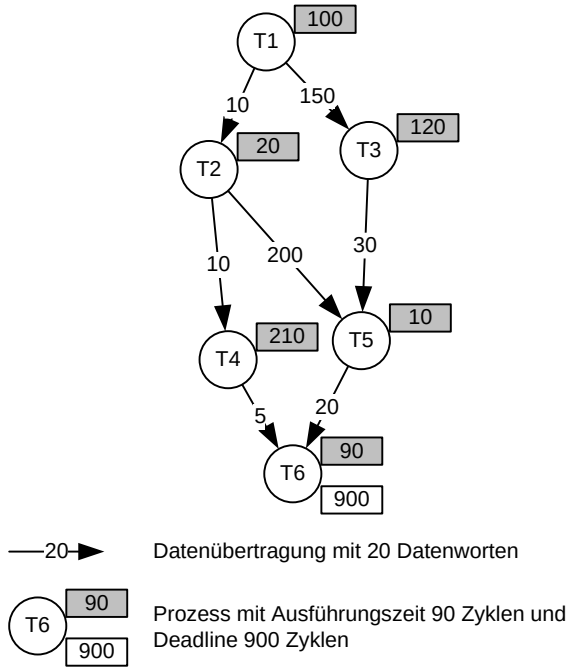


Abbildung 2.11: Exemplarischer Task-Graph

Zur Analyse der Performance verschiedener NoC-Implementierungen für nicht definierten Datenverkehr werden Benchmarks benötigt. Diese können den Datenverkehr realer, repräsentativer Anwendungen und/oder auch synthetischen Datenverkehr auf einem NoC beschreiben. Allgemeine Benchmarks, wie sie zum Beispiel bei der Bewertung von Prozessoren [63] oder auch Multiprozessoren eingesetzt werden, existieren für NoC nicht [60]. In der Literatur werden häufig Benchmarks, die auf synthetischem Datenverkehr oder repräsentative Anwendungen wie Videocodierung [56, 64, 65] oder WLAN-Baseband Processing [66] beruhen, verwendet.

Ein häufig benutzter synthetischer Datenverkehr ist Random Traffic. Dabei werden Datenquelle, Datensenke, sowie die zu übertragende Datenmenge und Datenrate für jede

einzelne Datenübertragung zufällig ausgewählt. Es wird zwischen uniform Random Traffic [43, 67, 68] und Localized Random Traffic [69, 70, 71] unterschieden. Bei Uniform Random Traffic ist die Wahrscheinlichkeit, mit der eine Datensenke für eine Datenübertragung ausgewählt wird, für alle Datensenken gleich. Die Wahrscheinlichkeit, dass eine Datensenke in der Nähe einer Datenquelle liegt, ist bei lokalem Random Traffic erhöht. Dies wird häufig durch Poisson- oder Exponentialverteilungen realisiert [26].

Eine Alternative ist die Bestimmung der Datensenke nach einem festgelegten Algorithmus, wie zum Beispiel beim Bit Reversal, Perfect Shuffle, Butterfly, Matrix Transpose oder Complement Traffic (vgl. Tabelle 2.3 und [26]).

Tabelle 2.3: Synthetische Datenverkehrsarten

	Binäre Koordinaten		**Bemerkung**
	Datenquelle	**Datensenke**	
Bit Reversal	$a_{n-1}, a_{n-2},...,a_1,a_0$	$a_0, a_1,...,a_{n-2},a_{n-1}$	
Perfect Shuffle	$a_{n-1}, a_{n-2},...,a_1,a_0$	$a_{n-2}, a_{n-3},..., a_0, a_{n-1},$	Linksrotation um 1 Bit
Butterfly	$a_{n-1}, a_{n-2},...,a_1,a_0$	$a_0, a_{n-2},...,a_1, a_{n-1}$	Austauschen des höchst- und niederwertigsten Bits
Matrix Transpose	$a_{n-1}, a_{n-2},...,a_1,a_0$	$a_{n/2-1},...,a_0,a_{n-1},...,a_{n/2}$	
Complement	$a_{n-1}, a_{n-2},...,a_1,a_0$	$\overline{a_{n-1}, a_{n-2},...,a_1,a_0}$	

Sowohl durch Task-Graphen beschriebener Datenverkehr, als auch synthetischer Datenverkehr, besteht aus einzelnen Datenübertragungen. Diese werden durch eine Datenmenge D_n, eine Datenblockgröße d_n und eine angeforderte Datenrate r_n definiert. Dabei muss D_n ein ganzzahliges Vielfaches von d_n sein.

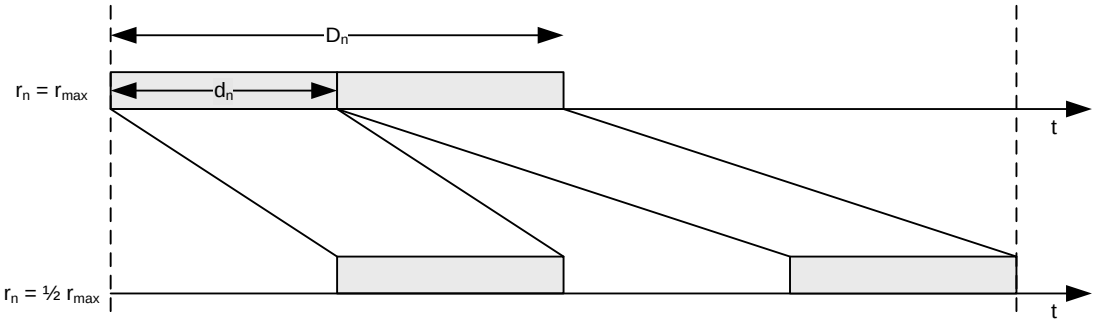


Abbildung 2.12: Format einer Datenübertragung

Die Performance eines NoC kann durch verschiedene Bewertungsmaße dargestellt werden. Ein verbreitetes und wichtiges Maß ist dabei die Latenz [5], die bei einer Datenübertragung auftritt. Diese berechnet sich aus der Zeit, die für die Datenübertragung insgesamt benötigt wird (t_{real}), und der Zeit, die für eine sofortige Übertragung (ohne NoC) benötigt worden wäre

(siehe Abbildung 2.13). Diese Zeit t_{opt} lässt sich aus der Datenrate r_n und der übertragenen Datenmenge D_n einer Datenübertragung berechnen

$$t_{opt} = \frac{D_n}{r_n}. \tag{2.3}$$

Die Latenz lässt sich in zwei Teilzeiten aufteilen: die erste Zeit wird durch die Verzögerung der Network-Interfaces, Routing-Switches und Links verursacht. Verzögerungen, die durch belegte Netzwerkressourcen verursacht werden, sind nicht berücksichtigt. Die zweite Teilzeit berücksichtigt diese dynamischen Effekte. Die Größe dieses Anteils ist stark von der Auslastung des NoC abhängig. Für Datenverkehr, der aus mehreren Datenübertragungen besteht, lässt sich so eine minimale, durchschnittliche und maximale Latenz bestimmen.

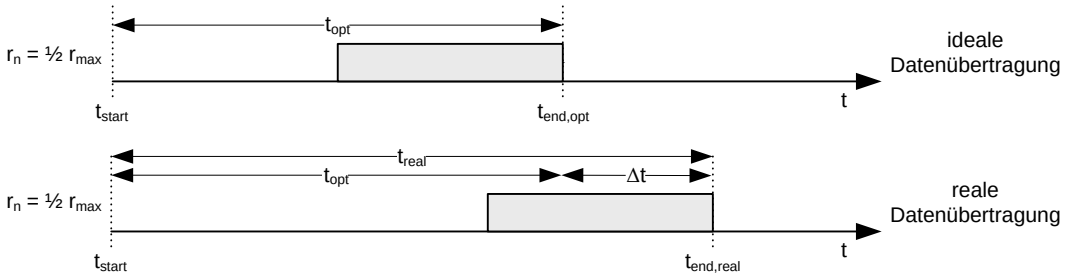


Abbildung 2.13: Definition der Latenz einer Datenübertragung

Die Datenrate r_n wird in diesem Fall von der Datenquelle vorgegeben und entspricht dem Intervall, in dem Datenpakete versendet werden. Die realisierte Datenrate lässt sich wie folgt berechnen:

$$r_{real} = \frac{D_n}{\Delta t + \frac{D_n}{d_n} \cdot t_{opt}} \tag{2.4}$$

2.4 Entwurfsproblematik

NoC-spezifische Parameter spannen einen großen Entwurfsraum auf. Die Dimensionen dieses Entwurfsraums repräsentieren dabei zum einen die in Kapitel 2.3.1 vorgestellten Kosten (Verlustleistung, Siliziumfläche, Entwicklungszeit und -aufwand) wie auch die in Kapitel 2.3.2 präsentierten Performance-Maße. Zusätzlich existieren noch weitere Dimensionen des Entwurfsraums wie beispielsweise QoS-Anforderungen, Effizienz (beispielsweise Energie/Bit), Testbarkeit (Testability) oder Flexibilität des NoC. Testbarkeit bezeichnet hier die Möglichkeit zur Verifikation der korrekten Funktion des NoC auf dem gefertigten Chip.

Innerhalb dieses Entwurfsraumes kann durch Variation von NoC-Parametern manövriert werden, wobei jeder Punkt des Entwurfsraums eine NoC-Implementierung repräsentiert. Um eine optimale NoC-Implementierung für ein System zu identifizieren, müssen die einzelnen

Implementierungen bewertet werden. Dazu muss eine Gewichtung der einzelnen Kostenfaktoren erfolgen.

Insbesondere die Entwicklungszeit und der Entwicklungsaufwand hängen neben den gewählten NoC-Parametern von dem verwendeten Entwicklungsprozess ab. Die Entwicklungszeit umfasst neben der Zeit zur Identifizierung der geeigneten NoC-Parameter auch die Zeit zur Implementierung, Bewertung und Verifizierung der NoC-Implementierung.

Um die Entwicklungszeit so gering wie möglich zu halten, ist eine systematische Entwurfsmethodik notwendig, um in dem komplexen Entwurfsraum die NoC-Implementierung zu identifizieren und zu realisieren, welch die durch das System gegebenen Anforderungen bei minimalen Kosten erfüllt. Eine solche Entwurfsmethodik wird im folgenden Kapitel vorgestellt.

Voraussetzung für eine systematische Untersuchung des Entwurfsraumes ist eine modulare und generische Beschreibung des NoC. Auf Basis dieser Beschreibung soll eine Implementierung eines NoCs unter Minimierung des dazu erforderlichen Entwurfsaufwands ermöglicht werden. Weiterhin muss eine Änderung einzelner NoC-Parameter möglich sein, ohne das NoC komplett neu zu beschreiben. Eine NoC-Beschreibung, die diesen Anforderungen genügt, wird in Kapitel 4.1 eingeführt.

Eine Untersuchung des Entwurfsraums für SoC wurde bereits in [75] eingeführt. Dabei wurde jedoch die Kommunikation zwischen den einzelnen funktionalen Einheiten des SoC nicht berücksichtigt. Diese hat jedoch eine nicht unerhebliche Auswirkung auf die Performance sowie die Kosten eines SoC, insbesondere bei SoC mit vielen funktionalen Einheiten. Daher wurde im Rahmen dieser Arbeit diese Ansätze aus [75] auf die Untersuchung des Entwurfsraums von SoC mit NoC erweitert.

3 Effiziente, emulatorgestützte NoC-Entwurfsmethodik

Durch die Vielzahl von NoC-spezifischen Parametern ergibt sich ein umfangreicher Entwurfsraum für NoC. Daher ist ein systematischer NoC-Entwurf notwendig, um in einer möglichst kurzen Entwurfszeit ein NoC zu implementieren und zu verifizieren, das die gestellten Anforderungen optimal erfüllt. Diese Anforderungen sind von Anwendung zu Anwendung bzw. von Anwendungsklasse zu Anwendungsklasse unterschiedlich. Daher muss jeweils individuell ein passendes NoC entworfen werden. Eine solche Entwurfsmethodik, die im Rahmen dieser Arbeit entwickelt wurde, wird in diesem Kapitel vorgestellt: Sie umfasst alle Schritte von der Systemanalyse über die automatisierte NoC-Generierung, Verifikation und Performance-Bewertung bis hin zur physikalisch optimierten VLSI-Implementierung. Dieser automatisierte Entwurfsablauf ist in Abbildung 3.1 schematisch dargestellt.

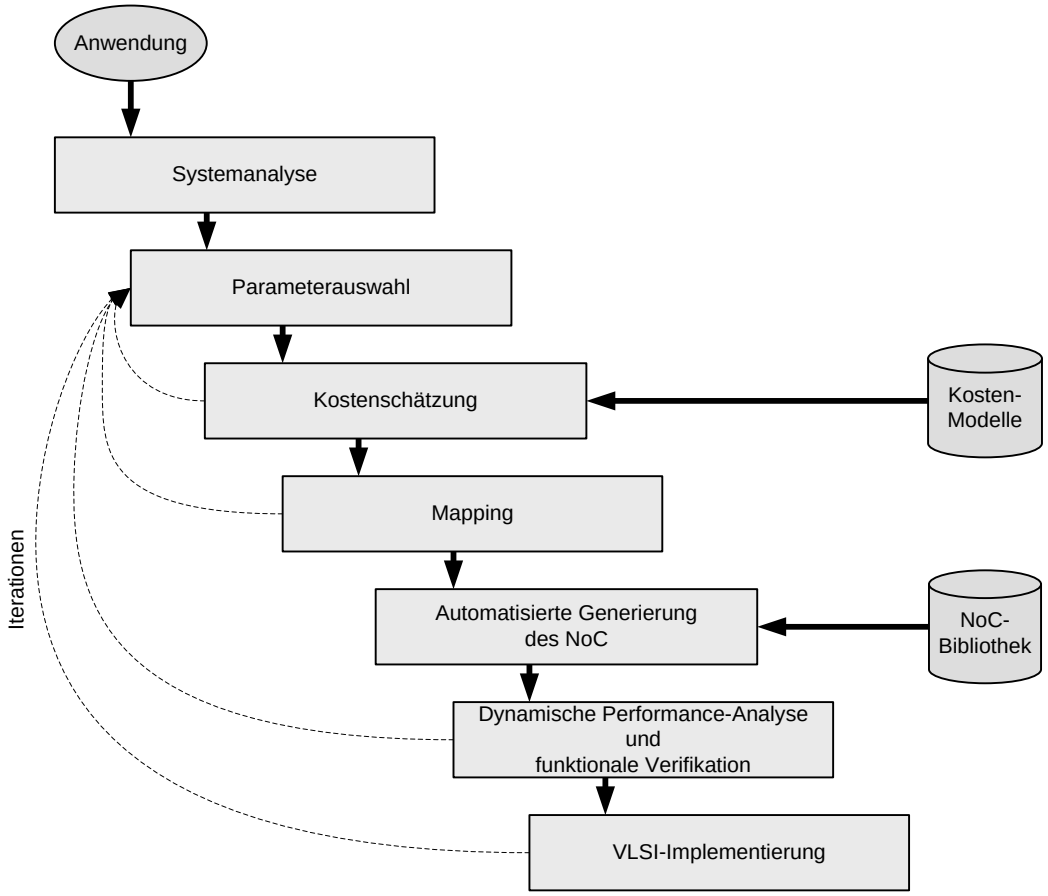


Abbildung 3.1: Entwurfsmethodik für NoC

Das Identifizieren des passenden NoC bzw. der NoC-Parameterkombination, die dieses NoC beschreibt, ist ein iterativer Prozess. Sobald während des Entwurfsprozesses festgestellt wird, dass die Kosten zu hoch sind, oder die Performance nicht ausreichend ist, wird diese Iteration abgebrochen und ein neuer Versuch mit veränderten NoC-Parametern gestartet. Um die Entwurfszeit zu minimieren muss sowohl die Anzahl der Iterationen, wie auch die

Iterationsdauer, minimiert werden. Durch eine Schätzung der, durch die Implementierung verursachten Kosten, in einer frühen Phase im Entwurfsprozess können attraktive Parameterkombinationen identifiziert und näher untersucht werden. Parameterkombinationen, die ein NoC beschreiben, das zu hohe Kosten verursacht, müssen nicht weiter untersucht werden.

Im weiteren Verlauf des NoC-Entwurfs wird ein Mapping, das heißt eine Zuordnung der funktionalen Einheiten des Systems zu den Network-Interfaces des NoC durchgeführt. Dieses Mapping hat erheblichen Einfluss auf die Performance des NoC. Im Anschluss an das Mapping werden verschiedene Beschreibungen des NoC beispielsweise für eine VLSI-Implementierung oder für eine Performance-Analyse automatisch generiert. Mit Hilfe dieser Beschreibungen wird im nächsten Schritt eine dynamische Performance-Analyse, beispielsweise mit einem FPGA-basierten Emulator durchgeführt. Anschließend, wenn sowohl Performance wie auch Kosten den Anforderungen genügen, wird eine VLSI-Implementierung durchgeführt. Diese umfasst eine Implementierung mit Hilfe von Standardzellen und physikalisch optimierten Kernkomponenten.

Diese einzelnen, in Abbildung 3.1 dargestellten Entwurfsphasen werden im Folgenden detailliert vorgestellt.

3.1 Systemanalyse

In der Systemanalyse wird eine gegebene Anwendung, die auf einem SoC implementiert werden soll, in mehrere Systemblöcke aufgeteilt. Diese Systemblöcke werden anschließend auf Architekturblöcke, wie zum Beispiel digitale Signalprozessoren (DSP), Mikrokontroller, oder eingebetteten FPGAs, abgebildet [72]. Bei dieser Abbildung müssen zum einen die benötigte Performance und zum anderen die verursachten Kosten berücksichtigt werden [2].

Mit dieser System- und Architekturbeschreibung des SoC kann die Kommunikation zwischen den einzelnen Systemblöcken analysiert und modelliert werden. Der zwischen den Systemblöcken entstehende Datenverkehr kann in Form eines Task-Graphen dargestellt werden, wie er in Kapitel 2.3.2 vorgestellt wurde. Neben der Beschreibung des Datenverkehrs muss auch das Budget für die Kommunikation auf dem SoC festgelegt werden. Es muss unter anderem bestimmt werden, welche Fläche und welche Verlustleistung zur Verfügung stehen. QoS-Anforderungen, die an das NoC gestellt werden, müssen definiert werden.

3.2 Parameterauswahl

Ausgehend von den für das NoC zulässigen Kosten, den Kommunikationsanforderungen, die das System an das NoC stellt (z.B. Anzahl und Layout der funktionalen Einheiten, Technologie), muss manuell ein Satz von NoC-Parameter ausgewählt werden, die das NoC vollständig beschreiben.

Aus diesem Parametersatz wird eine abstrakte NoC-Struktur erzeugt. Diese besteht aus Objekten (Routing-Switch-Objekten, Network-Interface-Objekten, und Link-Objekten), denen NoC-Parameter zugewiesen werden. Die einzelnen Objekte sind untereinander gemäß der Topologie verbunden. Diese Struktur ist Grundlage für die späteren Schritte Kostenschätzung, Mapping und Performance-Analyse sowie die automatisierte Generierung des NoC. Diese Struktur unterstützt beliebige, auch hierarchische und/oder heterogene, Topologien.

Bei der Parameterwahl sollten zunächst die Parameter festgelegt werden, die einen großen Einfluss auf die Performance und die Kosten des NoC besitzen (z.B. Topologie, Wortbreite, Taktfrequenz). In weiteren Iterationen des Entwurfsprozesses können anschließend die übrigen Parameter optimiert werden.

Über eine grafische Oberfläche können die NoC-Parameter ausgewählt werden, und die weiteren Entwurfsphasen, wie beispielsweise das Mapping, die dynamische Performance-Analyse oder auch die VLSI-Implementierung gestartet werden.

3.3 Kostenschätzung

Eine Bestimmung der durch eine NoC-Implementierung verursachten Kosten in einem frühen Entwurfsstadium ist essenziell, um NoC, die zu hohe Kosten verursachen, zu identifizieren und zu verwerfen. Mit Hilfe einer Schätzung, die auf mathematischen Modellfunktionen basiert, können diese Kosten direkt nach der NoC-Parameterwahl bestimmt werden. Dadurch kann die Iterationsdauer für die Untersuchung von NoC, die zu hohe Kosten verursachen würden, erheblich verkürzt werden.

Die Kostenfunktionen, die zur Schätzung benötigt werden, müssen in einer Bibliothek abgelegt sein. Zur Bestimmung dieser Funktionen wurden repräsentative NoC implementiert. Auf Basis dieser Implementierungen wurden anschließend die Modellfunktionen erstellt. Die Modellierungsmethodik sowie die ermittelten Modellfunktionen werden in Kapitel 5.1 vorgestellt.

3.4 Mapping

Die Zuordnung einer funktionalen Einheit, bzw. des auf der funktionalen Einheit ausgeführten Prozesses, zu einem bestimmten Network-Interface in einem NoC wird als Mapping bezeichnet. Dies kann auch als Abbildung eines Task-Graphen auf ein NoC interpretiert werden (vgl. Abbildung 3.2). Im Folgenden wird das Mapping als das Abbilden eines Tasks des Task-Graphen auf ein Network-Interface interpretiert.

Das Mapping hat dabei erheblichen Einfluss auf die Performance eines NoC. Insbesondere bei größeren NoC mit n Network-Interfaces ergibt sich eine sehr große Anzahl von $n!$

Möglichkeiten die funktionalen Einheiten eines Systems auf Network-Interfaces eines NoC abzubilden. Ziel des Mapping ist es, eine Abbildung zu finden, die eine hohe Performance bei minimalen Kosten ermöglicht. Für die quantitative Bewertung der Abbildung werden Bewertungsfunktionen benötigt. Diese sollen mit geringem Rechenaufwand eine objektive Bewertung der Abbildung ermöglichen. Hier wird diese Bewertung mit Hilfe einer statischen Performance-Analyse durchgeführt. Der genaue Ablauf dieser Analyse und der Bewertungsfunktionen werden im Folgenden erläutert. Falls das Ergebnis des Mappings ergibt, dass die geforderte Performance auf dem spezifizierten NoC nicht erreicht werden kann, muss eine neue Iteration mit einer angepassten NoC-Parameterwahl begonnen werden.

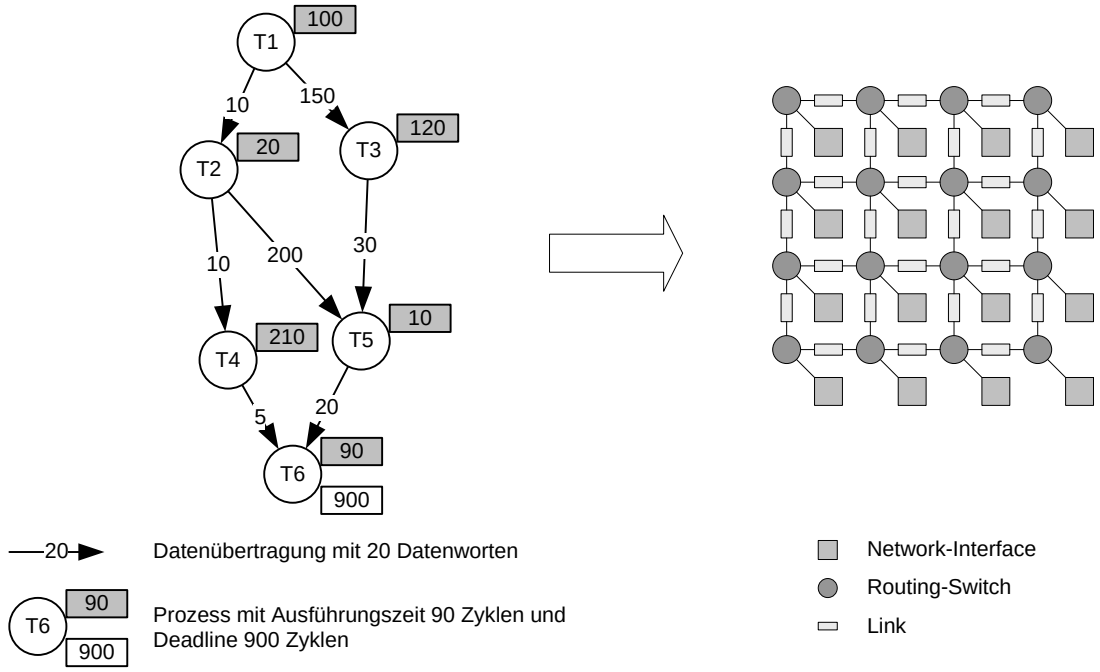


Abbildung 3.2: Abbildung eines Task-Graphen auf ein NoC

Um aus dieser Vielzahl von Abbildungsmöglichkeiten eine optimale, oder zumindest eine in vertretbarer Rechenzeit gute Abbildung zu bestimmen, wurden in [73] verschiedene heuristische wie auch exakte Ansätze vorgestellt. Im Rahmen dieser Arbeit wurde ein Branch & Bound-Algorithmus A*-Algorithmus nach [74] implementiert und optimiert. Diese beiden Algorithmen werden in den folgenden Abschnitten beschrieben und die Leistungsfähigkeit der Algorithmen sowie der Optimierungen anhand eines Beispiels untersucht und bewertet.

3.4.1 Mapping-Algorithmus

Mapping-Algorithmen lassen sich als sogenannter Suchbaum beschreiben und visualisieren. Exemplarische Suchbäume sind in Abbildung 3.3 dargestellt. Dabei steht jeder Knoten des Suchbaums für eine (Teil-)Abbildung eines Task-Graphen auf ein NoC. Die Kanten des

Suchbaums repräsentieren die Abbildung eines Tasks auf ein Network-Interface. Vereinfachend wird hier angenommen, dass auf ein Network-Interface des NoC maximal ein Task abgebildet werden kann. Die Knoten in einer Ebene des Suchbaums besitzen eine konstante Anzahl abgebildeter Tasks, die von Ebene zu Ebene steigt. Bei dem Knoten in der ersten Ebene des Suchbaums ist noch kein Task auf das NoC abgebildet worden, bei den Knoten in der 2. Ebene ist jeweils ein Task auf das NoC abgebildet worden und so weiter. Bei einem Task-Graphen mit beispielsweise sechs Tasks besitzt der Suchbaum für eine vollständige Abbildung des Task-Graphen auf ein NoC entsprechend sieben Ebenen.

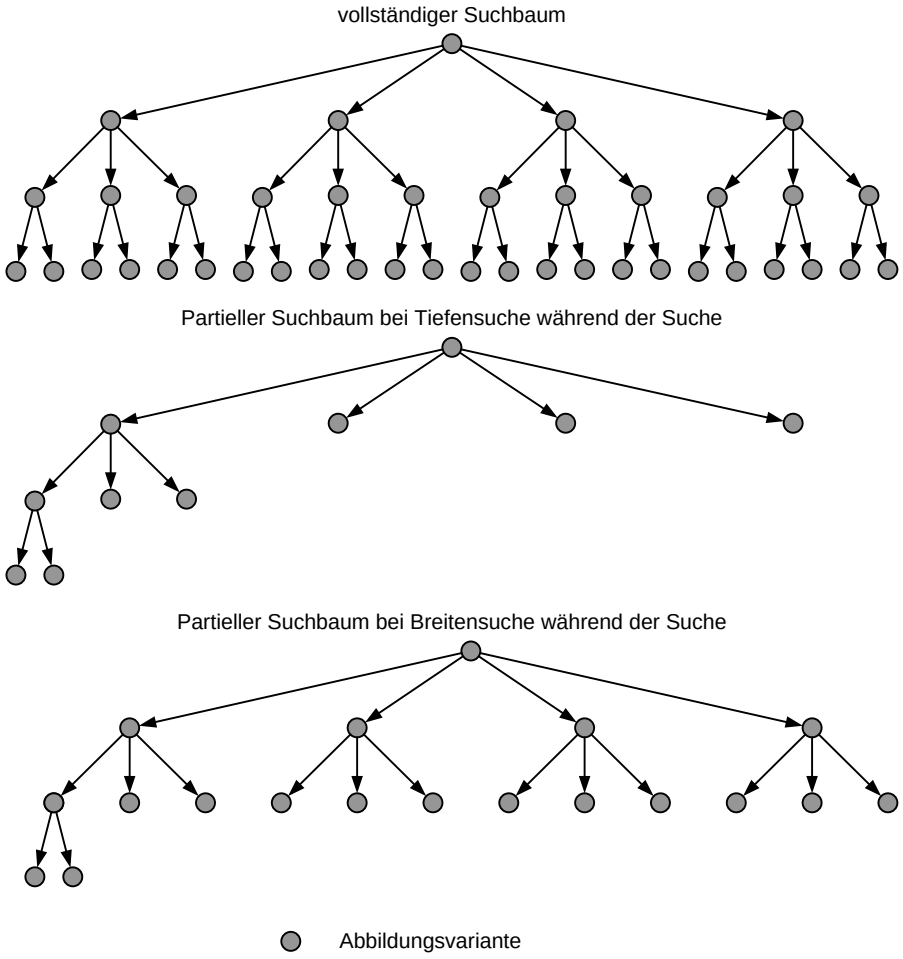


Abbildung 3.3: Suchbaum beim Branch & Bound-Algorithmus

Ein Suchbaum wird im Rahmen eines Mappings aufgebaut. Begonnen wird dies mit dem Wurzelknoten, der eine initiale Abbildung repräsentiert, die in der Regel leer ist. Dieser Knoten wir expandiert, d.h. es werden Nachfolgeknoten erzeugt, in dem ein Task auf verschiedene Network-Interfaces abgebildet wird. Anschließend wird aus allen noch nicht expandierten Knoten ein weiterer Knoten gewählt und expandiert. Die Wahl des zu expandierenden Knotens sowie die Art der Expansion werden durch den Mapping-

Algorithmus vorgegeben. Es werden so lange Knoten expandiert, bis der Mapping-Algorithmus terminiert.

Zur Bewertung einer (Teil-)Abbildung werden Kosten dieser Abbildung mithilfe der in Kapitel 3.4.2 beschriebenen statischen Performance-Analyse bestimmt. Hier können, je nach Optimierungsziel, unterschiedliche Kostenmaße wie beispielsweise die Anzahl der Hops (HOPS) oder die mit der jeweils übertragenen Datenmenge gewichteten Anzahl der Hops (WEIGHTED_HOPS) verwendet werden. Auch eine Verwendung anderer Kostenmaße, wie Latenz oder Verlustleistung ist möglich.

Bei dem ersten implementierten Suchalgorithmus, dem Branch & Bound-Algorithmus, werden zwei Ausprägungen unterschieden, das Breiten- und das Tiefensuchverfahren. Bei dem ersten Verfahren wird der zu expandierende Knoten des Suchbaums so gewählt, dass immer eine komplette Ebene des Suchbaums aufgebaut wird. Bei der Tiefensuche hingegen wird ein Ast des Suchbaumes bis zum Endknoten expandiert, anschließend ein weiterer (Teil-) Ast bis zum nächsten Endknoten usw. Bei diesem Verfahren werden nur diejenigen Knoten weiter expandiert, deren Abbildungskosten geringer sind als die der besten vollständigen Abbildung. Somit können einige Teil-Äste ausgeschlossen und die Anzahl der zu expandierenden Knoten und somit die Rechenzeit minimiert werden.

Eine Weiterentwicklung des Branch & Bound-Algorithmus ist der A*-Algorithmus [74]. Bei diesem Algorithmus wird immer derjenige Knoten, unabhängig von der Ebene im Suchbaum, expandiert, der die geringsten Kosten verursacht. Damit die Knoten in den oberen Ebenen des Suchbaums nicht überproportional häufig zur Expansion ausgewählt werden, müssen die Kosten für die noch nicht abgebildeten Tasks mit Hilfe einer Schätzfunktion vorläufig bestimmt werden. Die geschätzten Kosten dürfen dabei nicht größer als die bei einer Abbildung tatsächlich entstehenden Kosten sein. Zum Schätzen der Kosten der noch nicht abgebildeten Tasks wird hier angenommen, dass Start- und Zielknoten direkt nebeneinander liegen und somit minimale Kosten verursachen. Sobald ein Knoten in der untersten Ebene des Suchbaums die geringsten Kosten im Suchbaum besitzt, ist eine optimale Abbildung des Task-Graphen auf das NoC gefunden worden.

Da die Anzahl der möglichen Lösungen auch bei relativ kleinen NoC sehr groß ist und in einer sehr langen Rechenzeit resultiert, wurden im Rahmen dieser Arbeit Optimierungen der Mapping-Algorithmen erarbeitet. Dabei können Optimierungen unterschieden werden, die den Lösungsraum einschränken und solche, die den Lösungsraum nicht einschränken. Durch eine Einschränkung des Lösungsraumes ist die durch den Algorithmus gefundene Lösung nicht mehr zwangsläufig auch die optimale Lösung.

In einer ersten Optimierung werden die Knoten des Task-Graphen sortiert und nicht in einer zufälligen Reihenfolge auf das NoC abgebildet. Dabei wird jeder Knoten mit der Summe der

ein- und ausgehenden Datenmenge gewichtet. Der Knoten mit der höchsten Bewertung wird als erstes auf das NoC abgebildet.

Eine weitere Möglichkeit, das Mapping zu beeinflussen, ist, Abbildungen, bei denen die Auslastung einer NoC-Komponente deren Kapazität überschreitet, nicht zu berücksichtigen.

Eine drastische Reduktion der möglichen Abbildungen, und somit auch der Rechenzeit, ergibt sich, wenn die Anzahl der Folgeknoten beschränkt wird. Durch diese Maßnahme lassen sich der Lösungsraum und damit die Rechenzeit erheblich einschränken.

Um die Effektivität der implementierten Algorithmen und den Einfluss der vorgenommenen Optimierungen auf die Rechenzeit und Abbildungsqualität zu demonstrieren, wurde ein exemplarischer Task-Graph mit 23 Knoten auf ein NoC mit 25 Network-Interfaces abgebildet. Als Bewertungsmaß wurde die mit der Datenmenge gewichtete Anzahl der besuchten Routing-Switches für alle Datenübertragungen des Datenverkehrs verwendet. Insgesamt ergeben sich ca. $7{,}7 \cdot 10^{24}$ Möglichkeiten, die Knoten des Task-Graphen auf das NoC abzubilden. In Abbildung 3.4 ist die Rechenzeit, die bei Verwendung des Branch & Bound-Algorithmus (B&B) und bei Verwendung des A*-Algorithmus benötigt wird, sowie die Kosten der besten Abbildung, aufgetragen. Durch die verschiedenen Optimierungsmaßnahmen wird nicht mehr zwangsläufig die optimale Abbildung gefunden, dafür jedoch die Rechenzeit für die Suche nach der Abbildung erheblich verkürzt (vergleiche Abbildung 3.4). Als Kostenmaß wurde exemplarische die Anzahl der Hops gewählt. Andere Untersuchungen haben gezeigt, dass das verwendete Kostenmaß einen vernachlässigbaren Einfluss auf die Rechenzeit hat.

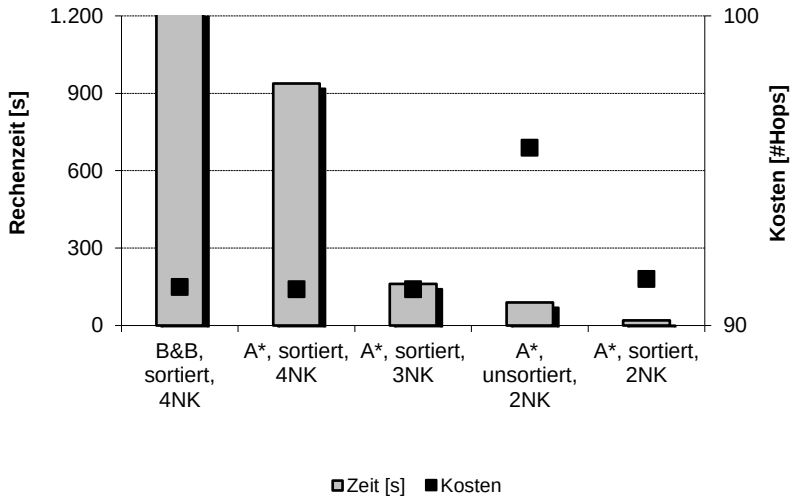


Abbildung 3.4: Rechenzeit und Kosten für das Mapping mit verschiedenen Mapping-Algorithmen und Optimierungsvarianten

Für die große Anzahl möglicher Abbildungen würden die untersuchten Algorithmen ohne Optimierungen eine zu hohe Rechenzeit benötigen. Daher wurde hier die Zahl der Nachfolgeknoten (NK) bei der Expansion eines Knoten beschränkt. Durch diese Einschränkung des Lösungsraumes ist jedoch nicht mehr sichergestellt, dass die optimale Lösung des Abbildungsproblems gefunden wird. Die Messungen wurden auf einem PC mit einem Pentium Core Duo Prozessor (3 GHz) und 1 GByte DDR RAM durchgeführt.

Der Branch & Bound-Algorithmus terminiert bei maximal vier Nachfolgeknoten nach 5500 s, wenn die Tasks des Task-Graph vor dem Mapping bezüglich der ein- und ausgehenden Datenmenge sortiert wurden. Durch die Verwendung des A*-Algorithmus können, bei ebenfalls maximal vier Nachfolgeknoten, die Rechenzeit deutlich und die Kosten leicht gesenkt werden. Durch eine weitere Reduktion der möglichen Nachfolgeknoten kann hier die Rechenzeit weiter reduziert werden. Die erreichte Abbildungsqualität sinkt dagegen nur leicht. Wenn ein umsortierter Task-Graph für das Mapping verwendet wird, steigt die benötigte Rechenzeit und die Abbildungsqualität sinkt (vgl. Abbildung 3.4).

Die in Abbildung 3.4 dargestellten Ergebnisse bezüglich der Rechenzeit und der Abbildungsqualität, ausgedrückt durch die Kosten, hängen von der Größe des NoCs (Anzahl der Network-Interfaces) und des Task-Graphen (Anzahl der Knoten und Kanten) ab. Daneben beeinflusst aber auch die Topologie des NoCs die Rechenzeit. Diese Ergebnisse können qualitativ auf andere Abbildungen von Task-Graphen auf NoC übertragen werden.

Für homogene NoC kann eine weitere Optimierung durch die Ausnutzung der vorhandenen Symmetrien des NoCs erreicht werden. Darauf wurde hier bewusst verzichtet, da die implementierten Algorithmen auch bei heterogenen NoC und hierarchischen NoC mit beliebigen und unsymmetrischen Topologien eingesetzt werden können.

Durch die Einschränkung des Lösungsraumes wird nicht zwangsläufig die optimale Lösung des Abbildungsproblems gefunden. Durch die Anwendung von sogenannten Nachbarschaftssuchverfahren (z.B. Simulated Annealing [76], Tabu Search [77]) kann versucht werden, die Abbildung weiter zu verbessern. Dazu kann die gefundene Abbildung mit den geringsten Kosten als Ausgangslösung verwendet werden.

Im Rahmen dieser Arbeit wurden die Algorithmen A* sowie Branch & Bound zum Abbilden eines Task-Graphen auf ein NoC bei minimalen Kosten implementiert. Diese Implementierungen wurden bezüglich der Performance des NoC optimiert. Durch die Verwendung des A*-Algorithmus mit einer sortierten Liste und zwei Nachfolgeknoten konnte die Ausführungszeit, bei nahezu gleicher Abbildungsqualität, um den Faktor 281 reduziert werden. Sie unterstützen beliebige Topologien und Kostenmaße (Hops, gewichtete Anzahl Hops, Verlustleitung, Energie, Latenz, Auslastung). Dies wird durch die Verwendung der in

Kapitel 5 vorgestellten Kostenmodellierung von NoC und den daraus resultierenden Kostenmodellen ermöglicht.

3.4.2 Statische Performance-Analyse

Bei einer statischen Performance-Analyse wird die Auslastung einzelner NoC-Komponenten bestimmt und Bewertungsmaße werden für Datenverkehr auf einem NoC erstellt. Es werden jedoch keine dynamischen Effekte wie zusätzlicher Datenverkehr durch Kontrollinformationen, wiederholter Verbindungsaufbau nach einem Verbindungsabbruch etc. berücksichtigt. Um die statische Performance-Analyse durchführen zu können, wird neben dem NoC und einer Datenverkehrsbeschreibung in Form eines Task-Graphen eine Abbildung des Task-Graphen auf das NoC benötigt.

Für jede Kante des Task-Graphen, deren Quell- und Zielknoten auf das NoC abgebildet wurden, werden die verursachten Kosten ermittelt. Diese Kante repräsentiert eine Datenübertragung mit definierter Datenmenge und Datenrate. Für diese Datenübertragung wird der Weg der Daten durch das NoC bestimmt. Dabei werden verschiedene Bewertungsmaße berechnet. Dies kann die Anzahl der für diese Datenübertragung besuchten Routing-Switches sein, die mit der Datenmenge gewichtete Anzahl der besuchte Routing-Switches, die benötigte Latenz oder auch die benötigte Verlustleistung. Zur Bestimmung der Latenz und der benötigten Verlustleistung werden die in Kapitel 5.1.1 vorgestellten Modellfunktionen für das zeitliche Verhalten bzw. der Verlustleistung von NoC-Komponenten verwendet.

Für den gesamten durch den Task-Graph beschriebenen Datenverkehr lassen sich durch eine Kombination dieser Ergebnisse entsprechende Bewertungsmaße, wie beispielsweise die minimale, durchschnittliche und maximale Latenz, oder die zur Datenübertragung benötigte Energie oder Verlustleistung berechnen.

Neben der Berechnung der Bewertungsmaße wird auch die Auslastung der benutzten NoC-Komponenten durch die Datenübertragungen bestimmt. Durch diese Analyse können potentiell überlastete NoC-Komponenten bereits in dieser frühen Phase des NoC-Entwurfs identifiziert werden. Daraufhin können Maßnahmen getroffen werden, um diese Komponenten zu entlasten, wie zum Beispiel angepasste Routing-Algorithmen oder eine veränderte Topologie.

Ein Problem dieser Analysetechnik ist die Berücksichtigung von dynamischen Effekten, wie sie zum Beispiel durch adaptive Routing-Algorithmen oder bereits belegte Netzwerk-Ressourcen verursacht werden. Eine dynamische Performance-Analyse, welche diese Effekte berücksichtigt, ist in dieser Entwurfsphase jedoch zu aufwändig.

Es wurde eine Software zur statischen Performance-Analyse implementiert. Mit Hilfe dieser Software können die auf einem NoC entstehenden Kosten sowie die Auslastung der einzelnen NoC-Komponenten bestimmt werden. Durch Einbindung der in Kapitel 5 vorgestellten Modellfunktionen kann auch die entstehende Verlustleistung und Latenz bestimmt werden. Es werden NoC mit beliebiger Topologie und verschiedenen Routing-Algorithmen unterstützt.

3.5 Automatisierte Generierung von NoC

Die Implementierung eines NoC nimmt einen großen Teil der Entwicklung von NoC ein. Diese Zeit ist zudem von der Erfahrung des Entwicklers abhängig. Zur Minimierung dieser Zeit wurden NoC-Bibliotheken erstellt. Mit Hilfe dieser Bibliotheken können automatisch NoC-Beschreibungen für die VLSI-Implementierung sowie für die dynamische Performance-Analyse erstellt werden. Diese Bibliotheken basieren auf einer modularen und generischen Struktur des NoC, sowie der einzelnen NoC-Komponenten, wie sie in Kapitel 4.1 vorgestellt wird.

Die Beschreibung des NoC für die VLSI-Implementierung sowie für den FPGA-basierten Emulator erfolgt in VHDL. Dazu enthält die VHDL Bibliothek parametrisierbare VHDL Beschreibungen der einzelnen funktionalen Einheiten wie Routing-Switch und Network Interface. Anhand der vorgegebenen NoC Parameter wird eine VHDL-Datei erstellt, in der die einzelnen NoC Komponenten instanziiert werden, mit den entsprechenden, für die jeweilige Komponente individuellen Parametern. Durch diesen Aufbau unterstütz die VHDL Bibliothek auch Topologien, die nicht in Kapitel 2.2.8 beschrieben sind, wie z.B. mehrere NoC mit Mesh Topologie, die mit einem Ring verbunden sind oder unregelmäßige NoC. Für diese NoC muss die VHDL-Beschreibung, in der die NoC Komponenten instanziiert und verbunden werden, manuell erstellt werden.

Für eine Analyse mit Hilfe einer SystemC basierten Simulation (vgl. Kapitel 3.6.2) wurde eine Bibliothek in C++ implementiert. Diese enthält die zyklengenauen Modelle der NoC-Komponenten. In dem SystemC Simulator werden die NoC anhand der vorgegebenen NoC-Parameter zur Laufzeit erstellt. Bei einer Untersuchung der Performance des NoC mit Hilfe von Colored Petri Nets (CPN, vgl. Kapitel 3.6.3) wurde eine weitere Bibliothek erstellt, die entsprechende Modellierungen des NoC für CPN enthält und automatisch XML-Dateien erstellt.

Für die VLSI-Implementierung des NoC mit Hilfe von physikalisch optimierten Kern-Komponenten wurden eine Bibliothek für die benötigten Kernkomponenten in einer 90 nm Technologie [109] implementiert. Die Beschreibung für den Datenpfad-Generators (DPG) [91] wurde erstellt. So lassen sich diese Komponenten ebenfalls für eine Vielzahl an NoC-Parameter Kombinationen automatisch generieren.

Neben diesen Beschreibungen werden zusätzlich Synthese-Skripte und eine Simulationsumgebungen für die VLSI-Implementierung der einzelnen NoC-Komponenten (Kapitel 3.7), sowie Synthese-Skripte für den FPGA-basierter Emulator (Kapitel 3.6.1) generiert. Mit Hilfe dieser Skripte lassen sich die einzelnen NoC-Komponenten aus Standardzellen synthetisieren und verifizieren.

3.6 Dynamische Performance-Analyse und funktionale Verifikation

Die dynamische Performance-Analyse wird benötigt, um Aussagen über die Leistungsfähigkeit eines NoC treffen zu können, wobei insbesondere dynamische Effekte berücksichtigt werden. Diese Analyse ist umso belastbarer, je größer die Datenmenge ist, die im Rahmen dieser Analyse über das NoC übertragen wird. Eine weitere wichtige Aufgabe ist die zyklengenaue, funktionale Verifikation des NoCs bzw. der einzelnen NoC-Komponenten.

Neben der Simulationsgenauigkeit ist die Simulationsgeschwindigkeit eine wichtige Eigenschaft der Analysemethode. In den folgenden Unterkapiteln werden verschiedene Methoden vorgestellt, die sich bezüglich der Simulationsgenauigkeit bzw. dem Abstraktionsgrad und der Simulationsgeschwindigkeit unterscheiden. Bei der Simulations-geschwindigkeit muss zusätzlich eine Vorbereitungszeit berücksichtigt werden, die pro untersuchtem NoC einmalig anfällt und, je nach Analysemethode, einen nicht unerheblichen Anteil an der Simulationsdauer hat. Diese Analysemethoden werden in diesem Kapitel quantitativ verglichen, bewertet und einem Einsatzbereich zugeordnet.

3.6.1 FPGA-basierte Emulation

Bei der FPGA-basierten Emulation wird das NoC auf einem FPGA nachgebildet. Selbst auf modernen FPGAs können komplexe SoCs aufgrund der begrenzten Anzahl an Logikelementen nicht auf einem einzelnen FPGA emuliert werden. Daher werden bei diesem Emulator die funktionalen Einheiten durch weniger komplexe, abstrakte Datenquellen und -senken ersetzt.

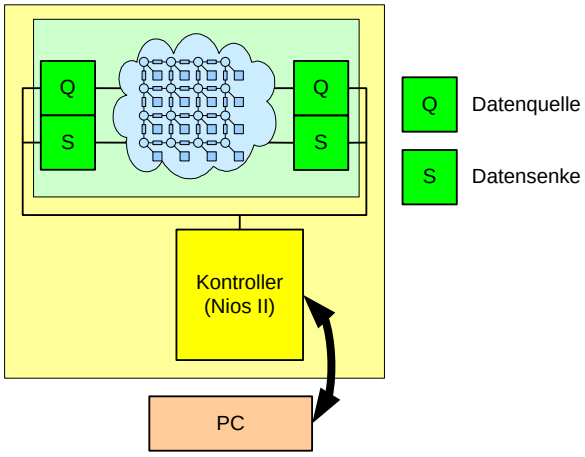


Abbildung 3.5: FPGA-basierte Emulationsumgebung

3.6.1.1 Emulationsumgebung Altera-FPGAs

Die hier verwendete Emulationsumgebung, die in Abbildung 3.5 dargestellt ist, besteht aus einem FPGA und einem PC. Der PC wird benötigt, um die FPGA-Konfiguration zu synthetisieren sowie Experimente zu starten und auszuwerten. Auf dem FPGA werden das zu simulierende NoC sowie die Datenquellen und –senken emuliert. Der Kontroller, ein Soft-Core-Prozessor, steuert die Datenquellen an und wertet die von den Datensenken erzeugten Rohdaten aus. Der Soft-Core-Prozessor dient zusätzlich zur Kommunikation mit dem PC. Bei dem hier verwendeten Kontroller handelt es sich um einen Nios II-Prozessor von Altera [78]. Dieser wurde mit speziellen Befehlen erweitert, um eine möglichst effiziente Anbindung an das NoC zu ermöglichen.

Die Performance eines NoC wird auf dem Emulator mit Hilfe von Experimenten bestimmt. Im Rahmen dieser Experimente werden in dem Kontroller Datenverkehrsanforderungen gemäß der vom Benutzer bestimmten Vorgaben erzeugt. Die Datenverkehrsanforderungen werden an die Datenquellen weitergeleitet, die entsprechenden Datenverkehr erzeugen.

Auf dem NoC-Emulator kann sowohl Datenverkehr, der in Form von Task-Graphen beschrieben wird, oder synthetischer Datenverkehr verwendet werden. Synthetischer Datenverkehr wird durch folgende Parameter charakterisiert, die bereits in Kapitel 2.3.2 vorgestellt wurden:

- Datenverkehrsart (vgl. Kapitel 2.3.2),
- Anzahl der Datenverkehrsanforderungen,
- durchschnittliche Auslastung auf dem NoC,
- minimale und maximale Datenrate,
- minimale und maximale Datenblockgröße,

- minimale und maximale Anzahl Datenblöcke pro Datenverkehrsanforderung,
- Anzahl der parallel sendenden Datenquellen,
- eine Liste mit zulässigen Datenquellen und Datensenken.

In den Datenquellen werden gemäß dieser Vorgaben Daten erzeugt, die anschließend über das NoC übertragen werden. Diese bestehen aus einem Datenkopf, in dem Informationen wie die Adresse der Datenquelle und der Datensenke, die Größe des Datenblocks und die Anzahl der zu sendenden Datenblöcke sowie die Startzeit der Datenübertragung enthalten ist. Neben dem Datenkopf bestehen die Datenpakete aus Pseudozufallszahlen. In der Datensenke wird, nachdem Daten empfangen wurden, überprüft, ob der Datenkopf korrekt empfangen worden ist. Beispielsweise wird kontrolliert, ob die Zieladresse mit der Adresse der Datensenke übereinstimmt. Außerdem wird die Empfangszeit protokolliert. Zusätzlich werden im Empfänger ebenfalls, mit dem Sender synchronisiert, Pseudozufallszahlen erzeugt, und diese mit den empfangenen Daten verglichen. An den Kontroller werden die Anfangs- und Endzeit einer Datenübertragung, sowie mögliche Fehler gemeldet.

Die Ergebnisse, die vom NoC-Emulator erzeugt werden, sind minimale, durchschnittliche und maximale Latenz der Datenübertragungen sowie die erreichte Auslastung. Bei der Verwendung eines Task-Graphen wird zusätzlich noch die Anzahl der erreichten und verfehlten Deadlines angegeben. Die Bestimmung der Auslastung einzelner NoC-Komponenten ist nur mit großem Hardwareaufwand möglich, da entsprechende dedizierte Zähler in den zu überwachenden NoC-Komponenten implementiert werden müssten. Daher wird dies nicht unterstützt.

Bevor ein Experiment gestartet werden kann, muss zunächst für jedes NoC einmalig der Emulator synthetisiert werden. Diese Vorbereitungszeit für ein Experiment ist nicht zu vernachlässigen. Die Änderung des Datenverkehrs eines Experimentes benötigt jedoch wenig Zeit, da die Datenverkehrsanforderungen im Kontroller erzeugt werden. Das Programm, das auf dem Kontroller ausgeführt wird, kann eine Vielzahl synthetischer Datenverkehrsarten sowie auf Task-Graphen basierenden Datenverkehr auf dem NoC initiieren (vgl. Absatz 2.3.2).

Für die Synthese des NoC eines FPGA-basierten Emulators wird der VHDL-Code verwendet, der auch die Basis für eine VLSI-Implementierung bildet. Bei der Implementierung des NoC konnte auf eine FPGA-spezifische Optimierung des NoC verzichtet werden. Die zur Synthese verwendeten Tools haben bei dem VHDL-Code bereits Speicher und Ähnliches erkannt und auf optimierte Speicherblöcke abgebildet.

Durch den Emulator mit der hohen Emulationsgeschwindigkeit ist eine zyklengenaue Emulation für Experimente mit großen Datenmengen, wie sie beispielsweise bei der Video-signalverarbeitung auftreten, in vertretbarer Zeit möglich.

3.6.1.2 Emulationsumgebung BEEcube

Für die Emulation großer NoC ist die Kapazität des verwendeten FPGAs (Altera Stratix II S60 [79]) nicht ausreichend. Daher wurde der Emulator auf die BEEcube Prototyping Platform der Firma CMC Microsystems portiert, der 4 FPGAs des Typs Xilinx Virtex-6 FPGA (XC6VLX550T). Dieses „Framework for NoC Evaluation by FPGA-based Emulation (FNOCEE)“ wurde in [80] zum ersten Mal präsentiert.

Neben der geänderten Hardware-Plattform gibt es weitere funktionale Änderungen, die in Abbildung 3.6 dargestellt sind. Die Steuerung eines Experimentes erfolgt nicht mehr durch einen Soft-Core Prozessor auf dem FPGA, sondern durch ein Host System mit deutlich größerer Leistungsfähigkeit. Die Eingabe für ein Experiment ist eine Beschreibung des Datenverkehrs in Form eines Task Graphen, sowie die Beschreibung des NoC (HW-Config). Das Mapping kann wie in dem, in Kapitel 3.6.1.1 beschriebenen Emulator, vorgegeben werden, als Teil der HW-Config. Alternativ bestehet die Möglichkeit, auf dem FPGA eine Optimierung des Mapping mit Hilfe von genetischen Algorithmen durchzuführen.

Die Funktion der zuvor beschriebenen Datenquellen und -senken ist in virtuellen Cores (VC) zusammengefasst worden. Diese werden über einen Bus angesteuert. Über diesen werden auch die Ergebnisse an das Host System zurück übermittelt.

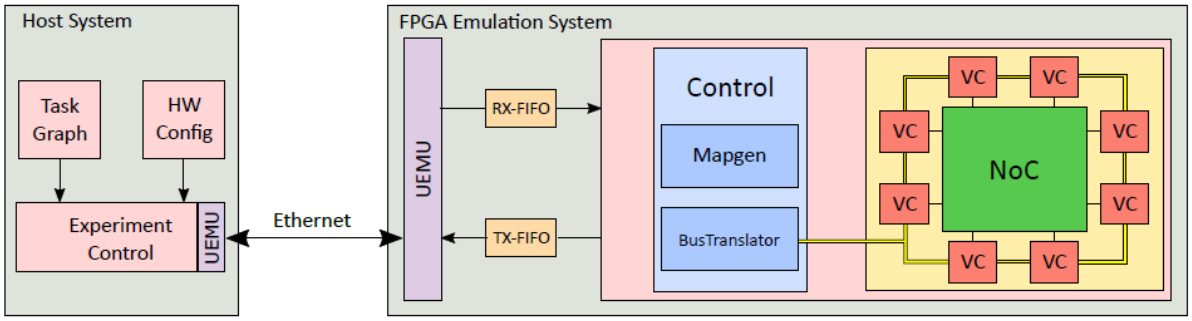


Abbildung 3.6: Allgemeiner Überblick über „Framework for NoC Evaluation by FPGA-based Emulation (FNOCEE)“

3.6.2 SystemC-basierte Simulation

SystemC ist eine C++ Klassenbibliothek die Hardwarebeschreibungssprachen, wie zum Beispiel VHDL oder VerilogHDL, mit der objektorientierten Programmiersprache C++ kombiniert. Zusätzlich stellt diese Open-Source Bibliothek einen Simulationskern zur Verfügung.

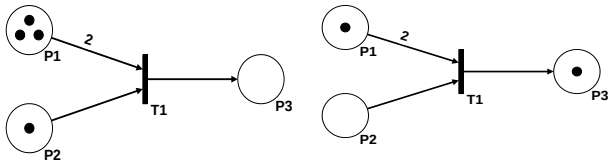
Im Rahmen der Forschungen zu NoC wurden viele SystemC-basierte Simulatoren auf verschiedenen Abstraktionsebenen vorgestellt [81]. Im Rahmen dieser Arbeit wurde ein zyklengenauer Simulator erstellt. Die Struktur des Simulators ist vergleichbar der Struktur des in Kapitel 3.6.1 vorgestellten FPGA-basierten Emulators. Der Simulator besteht aus den in

Kapitel 4.1 vorgestellten NoC-Komponenten und aus Datenquellen und -senken. Die Ansteuerung dieser Datenquellen und -senken erfolgt mit Hilfe des gleichen Programm-Codes, der auch auf dem Kontroller der FPGA-basierten Emulators verwendet wird.

Dieser Simulator ermöglicht die Simulation von NoC ohne eine Vorbereitungszeit, wie sie zum Beispiel beim FPGA-basierten Emulator anfällt. Die Auslastung der einzelnen Komponenten kann genau bestimmt werden.

3.6.3 Analyse mit Hilfe von Colored Petri Nets

Petri-Netze [82, 83] bieten eine graphische Beschreibungsmöglichkeit, die auf der Automatentheorie basiert und in verschiedenen Feldern des Ingenieurwesens Anwendung findet [84, 85]. Petri-Netze haben zwei Basiskomponenten: Plätze, die als Kreise, und Transitionen, die als Balken dargestellt werden. Plätze können mit Punkten markiert sein, die als Token bezeichnet werden. Plätze und Transitionen sind über Pfeile miteinander verbunden. Abhängig von der Richtung des Pfeils können ein oder auch mehrere Token von einem Platz durch eine Transition entfernt bzw. in einem Platz generiert werden. Eine Transition kann jedoch nur aktiviert werden (feuern), wenn jeder angeschlossene Platz die ausreichende Anzahl von Token besitzt. Diese Anzahl wird durch die Zahl an dem ausgehenden Pfeil bestimmt. Dadurch, dass in einem Petri-Netz mehrere Plätze gleichzeitig durch Token markiert werden können, eignen sie sich sehr gut zum Modellieren nebenläufiger Prozesse.



(a) (b)

Abbildung 3.7: Petri-Netz vor (a) und nach (b) dem Feuern der Transition T1.

In dem Beispiel, das in Abbildung 3.7 dargestellt ist, entfernt die Transition T1 zwei Token aus dem Platz P1 und einen einzelnen Token aus dem Platz P2. Anschließend wird ein neuer Token in Platz P3 generiert. Die Transition T1 kann nur feuern, wenn in Platz P1 mindestens zwei Token und in Platz P1 ein Token vorhanden ist.

Eine Erweiterung zu Petri-Netzen sind Colored Petri Nets (CPN) [86]. In klassischen Petri-Netzen besitzen die Token keine Eigenschaften. In CPNs hingegen können Token Daten enthalten. Diese Token-Daten werden in Datenstrukturen gespeichert, die *colorset* genannt werden, und Token und Plätzen zugeordnet sind. Das Format dieser colorsets reicht von einfachen Datentypen wie Zahlen bis hin zu komplexen Datentypen, die eine Kombination mehrerer einfacher Datentypen sind.

```
colset control: with data | request | ack;
colset packet: int, control;
```

Abbildung 3.8: CPN colorset.

Abbildung 3.8 zeigt zwei exemplarische colorset. Das colorset *control* definiert einen Datentyp, mit dem unterschiedliche Datenpakete unterschieden werden können. Das zweite colorset *packet* besteht aus zwei Teilen: einem Integer-Datentyp (*int*) und einem Teil, welcher das Format des zuvor definierten colorset *control* besitzt.

Die Daten eines Tokens können von einer Transition gelesen und modifiziert werden. Zusätzlich kann in CPN-Modellen die Zeit berücksichtigt werden. So können Token mit Zeitstempeln versehen werden. Zusätzlich kann Transitionen eine Bearbeitungsdauer zugewiesen werden.

Für ein CPN-Modell eines NoCs können durch eine Simulation des Modells unterschiedliche Messungen vorgenommen werden: der Durchsatz einer Transition (wie oft feuert eine Transition), die Belegung eines Platzes (Anzahl und Art der Token, die einen Platz belegen) und die Auswertung der Token-Daten. Im Rahmen dieser Arbeit wurde der Simulator CPN Tools [86] (Version 2.2.0) verwendet [87]. Bevor die Simulation gestartet werden kann, muss zunächst ein Syntax-Check des CPN Modells vorgenommen werden, der je nach Komplexität des CPN-Modells bis zu einer Stunde dauern kann. Bei kleineren Änderungen an dem Modell, wie beispielsweise anderem angeforderten Datenverkehr, kann ein partieller Check durchgeführt werden, der eine geringere Rechenzeit hat, als ein kompletter Syntax-Check.

Auch für CPN wurden hierarchische Modelle von den in Kapitel 4.1 vorgestellten NoC-Komponenten erstellt und in einer Bibliothek zusammengefasst. Das Zusammenfügen dieser einzelnen Komponenten zu einem NoC-Modell erfolgt ebenfalls automatisiert auf Basis der abstrakten NoC-Beschreibung, wie in Kapitel 3.5 beschrieben.

Das Erzeugen von Datenverkehrsanforderungen sowie die Auswertung der Ergebnisse einer Datenübertragung erfolgt, anders als im FPGA-basierten Emulator, nicht in einem zentralen Kontroller, sondern ist auf die einzelnen Datenquellen und Datensenken verteilt. Um eine bestimmte Datenverkehrsart zu realisieren, muss jede Datenquelle individuell konfiguriert werden.

Gemessen werden die durchschnittliche Latenz, die an jeder Datensenke auftritt, sowie die Auslastung jedes Network-Interfaces und jedes Routing-Switches. Diese Verteilung lässt sich leicht grafisch darstellen und ermöglicht so eine Identifikation von Bereichen des NoCs mit hoher Auslastung.

Dieser Simulator arbeitet, im Gegensatz zu den andern in diesem Kapitel vorgestellten Analyse-Techniken, ereignisgesteuert. Dadurch erreicht er eine höhere Simulationsgeschwindigkeit bei der Übertragung von großen Datenblöcken.

3.6.4 Simulation des VHDL-Codes

Neben der Emulation eines durch VHDL-Code beschriebenen NoCs besteht die Möglichkeit, diesen VHDL-Code zu simulieren. Auch diese Simulation des NoCs ist zyklengenau. Ein weiterer Vorteil ist, dass im Gegensatz zum FPGA-basierten Emulator, auf jedes Signal und Register des NoC zu Analyse-Zwecken zugegriffen werden kann. Für die Bewertung dieser Analyse-Technik wurde die Software ModelSim [88] verwendet.

3.6.5 Quantitativer Vergleich der Analysemöglichkeiten

Für einen quantitativen Vergleich der Simulationsgeschwindigkeit und Simulationsgenauigkeit der unterschiedlichen Performance-Analyse-Methoden wurden verschiedene Experimente durchgeführt. Es wurde exemplarisch ein NoC mit folgenden Parametern verwendet:

- Mesh Topologie,
- statisches XY-Routing,
- 16-Bit Bruttodatenwortbreite,
- Leitungsvermittlung,
- kein Fehlerschutz.

Die Größe des NoC, die der Anzahl der an das NoC angeschlossenen funktionalen Einheiten entspricht, wurde im Rahmen der Experimente variiert.

Zur Evaluierung wurde synthetischer, zufälliger Datenverkehr verwendet. Die Parameter, die für verschiedene Experimente verändert wurden, sind die angeforderte Auslastung des NoC und die Datenblockgröße. Die Experimente wurden auf einem PC mit einem Pentium Core Duo Prozessor (3 GHz) und 1 GByte DDR RAM durchgeführt. Der für den FPGA-basierten Emulator verwendete FPGA ist ein Altera Stratix II EP2S60 [79].

Die Leistungsfähigkeit von NoC kann durch verschiedene Maße quantifiziert werden (vgl. Kapitel 2.3.2). Für diese Untersuchung wurde die durchschnittliche Latenz gewählt, die Daten erfahren, die über ein NoC übertragen werden. Um die Genauigkeit der verschiedenen Performance-Analyse-Methoden zu vergleichen, wurden die einzelnen Ergebnisse mit den Ergebnissen der FPGA-basierten Emulation verglichen. Diese Werte wurden als Referenz betrachtet, da die hohen Simulationsgeschwindigkeiten Experimente mit sehr großen Datenmengen zulassen und der zur Synthese des Emulators verwendete VHDL-Code des NoC identisch mit dem VHDL-Code der späteren VLSI-Implementierung ist.

Wenn die Simulationsdauer ausreichend lang gewählt wurde, erreicht die Genauigkeit für alle dynamischen Analysetechniken einen relativen Fehler der unter 2 % liegt. Lediglich bei der statischen Performance-Analyse war der relative Fehler für mittlere und große Auslastung des NoC deutlich größer.

Zur Bestimmung der Simulationsgeschwindigkeit muss neben der eigentlichen Simulationsdauer eine Vorbereitungszeit berücksichtigt werden, die in den hier verwendeten Analysetechniken einmal für jedes NoC aufgewendet werden muss. Diese Zeit variiert stark zwischen den unterschiedlichen Techniken und unterschiedlichen NoC-Größen. Bei der FPGA-basierten Emulation muss zunächst die FPGA-Konfiguration für jedes NoC synthetisiert werden. Für eine Analyse mit Hilfe von Petri-Netzen muss ein Syntax-Check durchgeführt werden, bei der Simulation des VHDL-Codes muss dieser zunächst kompiliert werden. Die Vorbereitungszeiten sind für ein NoC mit Mesh Topologie und 25 funktionalen Einheiten in Tabelle 3.1 angegeben.

Tabelle 3.1: Vorbereitungszeit der verschiedenen Analysetechniken für ein NoC mit 16 Bit Bruttodatenwortbreite und Mesh Topologie mit 5 x 5 Teilnehmern.

	Vorbereitungszeit
Statische Performance-Analyse	-
VHDL Simulation (Kompilation des VHDL-Codes)	< 1 min
SystemC Simulation	< 1 s
CPN Simulation (Syntax-Check)	< 10 min
FPGA-basierte Emulation (Synthese des NoC-Emulators)	< 1 h

Die eigentliche Simulationsgeschwindigkeit, bei der die Vorbereitungszeit nicht berücksichtigt wird, lässt sich in übertragenen Datenworten pro Sekunde messen. Andere Maße, wie zum Beispiel simulierte Taktzyklen pro Sekunde, wurden auch untersucht und führen zu ähnlichen Ergebnissen. Da bei der statischen Performance-Analyse keine Simulation durchgeführt wird, ist diese Technik im folgenden Vergleich nicht berücksichtigt worden.

Sowohl NoC-spezifische Parameter wie auch Parameter, die den Datenverkehr beschreiben, haben einen Einfluss auf die Simulationsgeschwindigkeit. Dieser Einfluss unterscheidet sich von Analysetechnik zu Analysetechnik. Zuerst wurde der Einfluss der NoC-spezifischen Parameter untersucht. Exemplarisch wurde hier die Anzahl der funktionalen Einheiten, die an das NoC angeschlossen sind, variiert. Es wurden NoCs mit Mesh Topologie der Größe 2x2 bis 6x6 untersucht. Diese ergeben NoC mit 4 bis 36 funktionalen Einheiten. Dazu wurde Datenverkehr mit einer Auslastung von 50 % und einer Datenblockgröße von 100 Datenworten gewählt. Die Ergebnisse dieser Experimente sind in Abbildung 3.9 dargestellt.

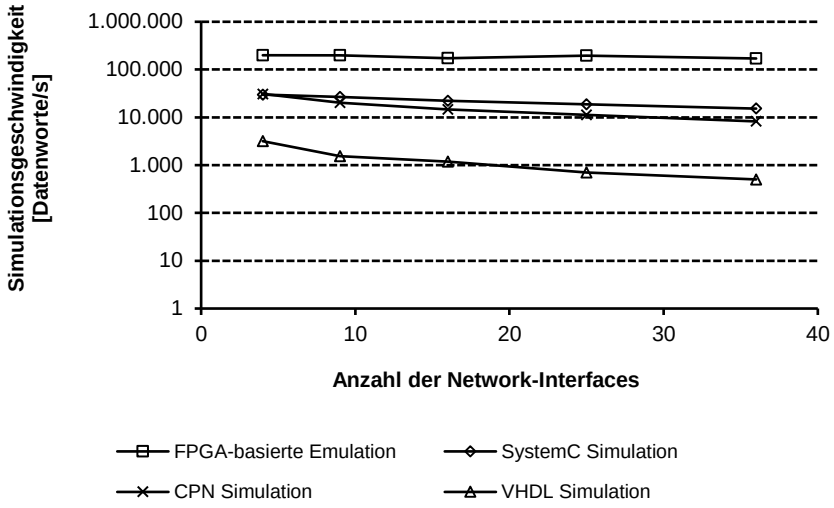


Abbildung 3.9: Simulationsgeschwindigkeit für NoCs mit variierter Größe

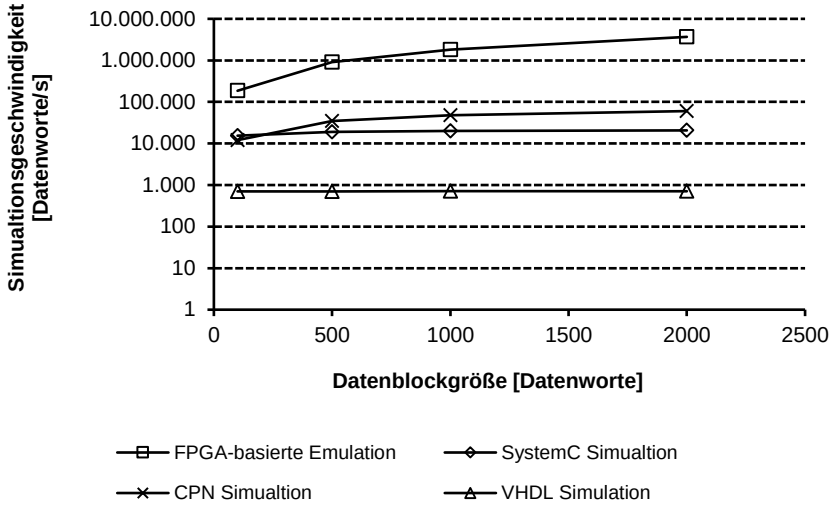


Abbildung 3.10: Simulationsgeschwindigkeit für NoC mit 25 funktionalen Einheiten und variierter Datenblockgröße

Die Anzahl der angeschlossenen funktionalen Einheiten hat einen vernachlässigbaren Einfluss auf die Simulationsgeschwindigkeit des Emulators, da die inhärente Parallelität des FPGAs genutzt werden kann. Bei den anderen Techniken nimmt die Simulationsgeschwindigkeit mit steigender NoC-Größe ab, da mit steigender NoC-Größe auch die Komplexität des zu simulierenden NoCs steigt. Dieser Effekt ist für den SystemC-basierten Simulator geringer als bei der VHDL-Simulation und der Analyse, die auf CPNs basiert.

Ein weiterer Parameter, der Einfluss auf die Simulationsgeschwindigkeit hat, ist die Datenblockgröße. Sowohl die Simulationsgeschwindigkeit des Emulators wie auch der CPN-basierten Simulation hat eine nahezu lineare Abhängigkeit von der Datenblockgröße. Bei dem Emulator wird die meiste Simulationszeit für die Generierung und Auswertung der

Datenverkehrsanforderungen benötigt. Bei mehr Datenworten pro Datenverkehrsanforderung steigt so die Simulationsgeschwindigkeit. Der Anstieg der Simulationsgeschwindigkeit der CPN-Simulation liegt an der Tatsache, dass der Simulator ereignisgesteuert und nicht zyklengenau arbeitet. Auch hier führt eine hohe Datenblockgröße zu weniger Ereignissen pro übertragenem Datenwort, was in einer gesteigerten Simulationsgeschwindigkeit resultiert. Die Simulationsgeschwindigkeiten der anderen Techniken bleiben von diesem Parameter unbeeinflusst.

Diese Experimente zeigen, dass der FPGA-basierte Emulator mindestens eine Größenordnung schneller ist, als die SystemC-basierte und die auf CPN basierte Simulation. Diese sind wiederum um eine Größenordnung schneller als die VHDL-Simulation. Die SystemC- und CPN-basierte Simulation haben eine vergleichbare Simulationsgeschwindigkeit. Je nach gewählten systemspezifischen Parametern (z.B. Datenblockgröße) hat die eine oder andere Technik weiter Vorteile.

Die Vorteile der statischen Analyse liegen in der geringen Analysezeit und den guten Beobachtungsmöglichkeiten für NoC beliebiger Größe. So kann beispielsweise die Auslastung einzelner Links und/oder Routing-Switches ausgegeben werden. Die Genauigkeit ist jedoch relativ gering, insbesondere bei hoher Auslastung des NoC. Zusätzlich können einige NoC-Parameter nur mit viel Aufwand berücksichtigt werden, wie beispielsweise adaptive Routing-Algorithmen. Durch die geringe Analysezeit ist eine Anwendung zu einem frühen Zeitpunkt im Entwurfsablauf sinnvoll. Hier werden aus einer Vielzahl von möglichen Parameterkombinationen wenige potentiell vorteilhafte Parameterkombinationen ausgewählt. Auch ist eine Anwendung im Rahmen der Abbildung von Task-Graphen auf NoC möglich (vgl. Kapitel 3.4).

Der VHDL-Simulator führt eine zyklengenaue Simulation des VHDL-Codes durch, der auch bei einer späteren VLSI-Implementierung verwendet wird. Gegebenenfalls. werden einzelne Module durch für die Zieltechnologie optimierte Module ersetzt. Die Funktion dieser Module bleibt jedoch unverändert. Da im Rahmen dieser Simulation jedes Signal und Register des NoC beobachtet werden kann, lässt sich eine optimale Überwachung des NoC realisieren. Aufgrund der geringen Simulationsgeschwindigkeit ist ein Einsatz zur Performance-Analyse, insbesondere bei großen NoC, nicht sinnvoll. Der VHDL-Simulator wird vielmehr zur Verifikation und zur Fehlersuche während der Implementierung einzelner Komponenten der VHDL-Bibliothek verwendet.

Der ereignisgesteuerte CPN-basierte Simulator hat eine relativ hohe Simulationsgeschwindigkeit, insbesondere für große Datenpakete bei synthetischem Datenverkehr. Auch bietet er gute Überwachungsmöglichkeiten. Jedoch muss für jedes NoC ein zeitintensiver Syntax-Check durchgeführt werden. Zudem steigt die Komplexität des CPN-Modells schon bei NoC

mittlerer Größe so an, dass der Syntax-Check bei den hier verwendeten Software-Werkzeugen nicht mehr durchführbar ist [89]. Außerdem ist die grafische Modellierung neuer Parametervarianten für ungeübte Entwickler aufwändig. Im Entwurfsprozess ist eine Anwendung im Rahmen der dynamischen Performance-Analyse sinnvoll.

Die Verwendung eines zyklengenauen, SystemC-basierten Simulators hat den Vorteil, dass keine Vorbereitungszeit benötigt wird, dynamische Effekte berücksichtigt werden und eine relativ hohe Simulationsgeschwindigkeit erreicht wird. Weitere Vorteile sind die guten Beobachtungsmöglichkeiten, die eine Identifikation von überlasteten NoC-Komponenten ermöglicht, sowie die nahezu unbegrenzte Größe des zu simulierenden NoC. Als Nachteil muss der erhöhte Modellierungsaufwand angesehen werden, da für eine spätere VLSI-Implementierung eine Beschreibung in einer Hardwarebeschreibungssprache, zum Beispiel VHDL, benötigt wird. Auch ist die Simulationsdauer für große Datenmengen, wie sie in realen Anwendungen vorkommen, zu groß und somit nicht praktikabel. Daher ist eine Anwendung bei der dynamische Performance-Analyse mit kleiner bis mittlerer Datenmenge sinnvoll.

Tabelle 3.2: Qualitativer Vergleich von Performance-Analysetechniken

	Statische Performance-Analyse	**VHDL Simulator**	**SystemC Simulator**	**CPN Simulator**	**FPGA-basierter Emulator**
Simulationsgeschwindigkeit	+ +	- -	+	+	+ +
Vorbereitungszeit	+ +	o	+ +	-	- -
Berücksichtigung dynamischer Effekte	- -	+ +	+ +	+ +	+ +
Beobachtungsmöglichkeiten	+	+ +	+ +	+	-
Genauigkeit	- -	+ +	o	o	+ +
Beliebige NoC-Größe	+ +	o	+ +	- -	- -
Modellierungsaufwand	+ +	o	+	-	o

Die höchste Simulationsgeschwindigkeit wird bei der FPGA-basierten Emulation erreicht. Sie ist ungefähr eine Größenordnung höher als die Simulationsgeschwindigkeiten von SystemC- und CPN-basierten Simulatoren. Die Taktfrequenz des NoC-Emulators ist circa eine Größenordnung kleiner als die Taktfrequenz eines SoC. Zusätzlich wird der identische Quell-Code sowohl bei der Synthese des NoC-Emulators wie auch bei der VLSI-Implementierung verwendet. Lediglich zur Steigerung der Performance des Emulators können im NoC

verwendete Speicher durch spezielle, auf den FPGA angepasste, Speicherblöcke beschrieben werden. Diese sind funktional identisch mit der Beschreibung für die VLSI-Implementierung. Dynamische Effekte werden auf zyklengenauer Ebene optimal berücksichtigt. Die maximale Größe eines NoC, das mit Hilfe eines FPGA-basierten Emulators untersucht werden kann, ist durch die Kapazität des FPGAs beschränkt. Dieses Problem kann jedoch durch die Kombination mehrere FPGAs gelöst werden. Die Synthese eines NoC-Emulators verursacht eine hohe Vorbereitungszeit. Daher ist eine Anwendung in einer frühen Entwurfsphase, in der viele NoC-Parameterkombinationen berücksichtigt werden müssen, nicht vorteilhaft. Diese Ergebnisse sind in Tabelle 3.2 qualitativ bewertet und zusammengefasst. Ein Maß für die Komplexität bei der Modellierung von NoC, die im Rahmen dieser Arbeit verwendet wurde, ist die Simulationszeit.

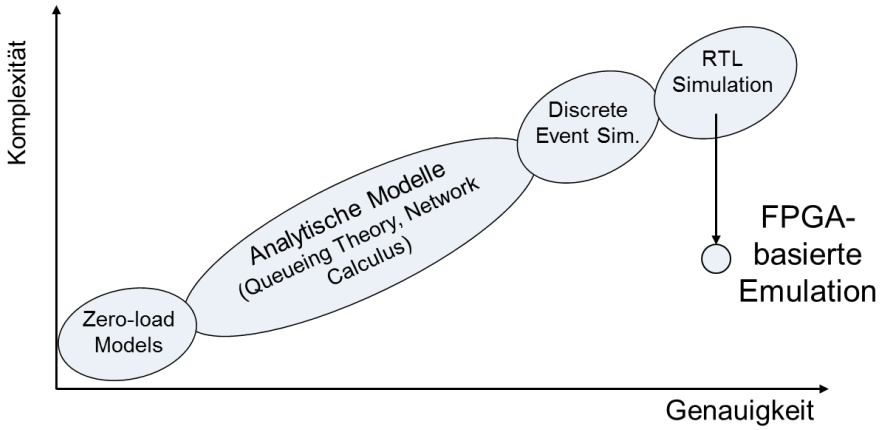


Abbildung 3.11: Schematische Einordung der Methoden zur Modellierung von NoC nach [90], erweitert um die Ergebnisse dieser Arbeit

In [90] wurde ebenfalls eine qualitative Einordnung unterschiedlicher Analyseverfahren vorgestellt, wie sie in Abbildung 3.11 dargestellt ist. Die in dieser Arbeit vorgestellten, quantitativen Ergebnisse zu den Simulationszeiten decken sich mit den qualitativen annahmen. Hierbei entsprechen die „Zero-load Models“ der statischen NoC-Analyse, die „Analytic Models“ werden durch den CPN-Simulator abgebildet, die „Discrete Event Simulator“ durch die SystemC Simulation und die RTL-Simulation durch die VHDL-Simulation.

Durch die Verwendung einer FPGA-basierten Emulation und die Kombination unterschiedlicher Analyseverfahren im Rahmen der Design Space Exploration kann die in Abbildung 3.11 dargestellte Systematik aufgebrochen, und eine hohe Präzision bei signifikant reduzierter Komplexität (Simulationszeit) während des gesamten Design Space Exploration erreicht werden.

3.7 VLSI-Implementierung

Die VLSI-Implementierung eines NoC lässt sich in die Implementierung der einzelnen NoC-Komponenten, Network-Interface, Routing-Switch und Link, sowie die Implementierung des gesamten NoC unterteilen. Bei der Implementierung des gesamten NoC wird die Platzierung der zuvor erstellten NoC-Komponenten innerhalb des SoC vorgenommen und die Komponenten werden untereinander verbunden. Die Platzierung der einzelnen Komponenten hängt stark von dem Layout und den Abmessungen der funktionalen Einheiten des NoC sowie der Topologie des NoC ab. Diese Informationen haben auch einen Einfluss auf die Implementierung der NoC-Komponenten, zum Beispiel die Länge der Links oder der zur Verfügung stehenden Fläche für einzelne Komponenten.

In dem hier vorgestellten Entwurfsprozess für NoC werden die einzelnen NoC-Komponenten mit einem in Kapitel 3.7.1 standardzellenbasierten Design-Flow erstellt. Dieser automatisierte Entwurfsprozess beinhaltet neben der Implementierung auch eine Verifikation und Bewertung der erstellten NoC-Komponente. Diese Implementierung kann, wenn Vorgaben bezüglich Fläche, Verlustleistung oder erreichbarer Taktfrequenz nicht erreicht werden, durch ein individuelles Eingreifen in den Synthese- bzw. Place and Route-Prozess optimiert werden. Eine weitere Möglichkeit, die Kosten zu senken bzw. die Performance zu steigern, ist die Verwendung von physikalisch optimierten Teil-Komponenten. Dies wird in Kapitel 3.7.2 beschrieben.

3.7.1 Standardzellenbasierte Implementierung

Die VLSI-Implementierung einer NoC-Komponente mit Standardzellen beginnt mit der Synthese. Dabei wird die VHDL-Beschreibung in eine Beschreibung mit Hilfe sogenannter Standardzellen überführt. Es können eine Vielzahl von Parametern vorgegeben werden, wie beispielsweise die zu erreichende Taktfrequenz, Lastkapazitäten an den Ausgängen der Komponente, die zulässige Fläche oder eine Flächenauslastung. Zusätzlich kann festgelegt werden, ob die Komponente bezüglich der Verlustleistung, der maximalen Taktfrequenz oder der Fläche optimiert werden soll.

Nach der Abbildung der VHDL-Beschreibung auf eine Beschreibung mit Standardzellen müssen diese Zellen platziert und miteinander durch Leitungen verbunden werden (Place and Route). Nach dem Platzieren und Verbinden wird eine statische Analyse des zeitlichen Verhaltens durchgeführt und die Setup- und Hold-Zeiten zwischen einzelnen Registern bzw. zwischen Ein- und Ausgangsports und Registern bestimmt und mit den Vorgaben verglichen. Anschließend werden gegebenenfalls die Treiberstärken einzelner Zellen durch Austauschen der Zellen erhöht oder verkleinert. Sollten die Vorgaben nicht erfüllt sein, wird iterativ der Place and Route-Prozess neu gestartet, bis alle Vorgaben erreicht sind.

Das so erstellte Layout der Komponente wird einem Design Rule Check unterzogen. Dabei wird überprüft, ob alle, durch die Technologie vorgegebene Regeln, wie zum Beispiel Mindestgrößen und Mindestabstände, eingehalten wurden. Wenn das Layout keine Design Rule-Fehler enthält, werden die vorhandenen Transistoren und parasitären Kapazitäten extrahiert und in eine Netzliste geschrieben.

Das analoge Verhalten des NoC-Layouts kann mit Hilfe dieser Netzliste simuliert werden. Mit einer Co-Simulation der VHDL-Beschreibung und einem Vergleich der Simulationsergebnisse kann die korrekte Funktion der einzelnen NoC-Komponenten unter Berücksichtigung des analogen Verhaltens verifiziert werden. Zusätzlich lässt sich im Rahmen dieser Simulation die Verlustleistung der Komponente in unterschiedlichen Betriebszuständen bestimmen.

Eine erste Optimierung dieser standardzellenbasierten Implementierung kann durch individuelles Anpassen der den Entwurfsprozess steuernden Parameter erfolgen. Dies kann sowohl während der Synthese als auch beim Place and Route erfolgen und ist ein iterativer Prozess.

3.7.2 Standardzellenbasierte Implementierung mit physikalisch optimierten Kernkomponenten

Die standardzellenbasierte Implementierung von NoC-Komponenten kann durch die Verwendung physikalisch optimierter Kernkomponenten weiter verbessert werden. Um einen möglichst großen Gewinn bei möglichst geringem Aufwand zu erhalten, müssen die zu optimierenden Komponenten bzw. Teilkomponenten einen signifikanten Anteil an der Fläche bzw. Verlustleistung haben. Sie sollten zusätzlich einen regelmäßigen Aufbau besitzen, damit der Entwurfsaufwand minimal ist. Gut geeignet für eine physikalische Optimierung sind diejenigen Teilkomponenten, die Daten transportieren, wie zum Beispiel der Switch des Routing-Switches. Dieser hat für jeden Eingangsport und jede Eingangsleitung die gleiche logische Funktion. Schlecht geeignet sind dagegen Teilkomponenten, die Steuerungs- und Kontrollfunktionen haben, wie beispielsweise der Router und Arbiter des Routing-Switches.

Der in Kapitel 3.7.1 vorgestellte Design-Flow wurde für die Verwendung von physikalisch optimierten Teilkomponenten erweitert. Dieser modifizierte Design-Flow ist in Abbildung 3.12 dargestellt. Ausgehend von einem VHDL-Code werden Teilkomponenten ausgewählt, für die eine physikalische Optimierung vorteilhaft sind. Anschließend wird für diese Teilkomponenten mit Hilfe des Datenpfad-Generators (DPG) [91] eine physikalisch optimierte Version der Teilkomponente erstellt, diese charakterisiert und in den standardzellenbasierten Design-Flow als weitere Bibliothek eingebunden.

Durch die Verwendung des DPG ist es möglich, reguläre Datenpfade, die aus wenigen Basiszellen bestehen, mit minimalem Entwurfsaufwand zu erstellen. Dazu werden einerseits

die Layouts der Basiszellen und andererseits (parametrisierbare) DPG-Beschreibung benötigt. Diese legt die Anordnung der Basiszellen und die Verbindungen der Basiszellen untereinander fest. Die DPG-Beschreibung beinhaltet eine Darstellung des Signalflussgraphen auf Wort- wie auch auf Bitebene. Für die Implementierung von Teilkomponenten für NoC müssen die Parameter der DPG-Beschreibung, wie zum Beispiel die Datenwortbreite, bestimmt werden, damit das Layout der Teil-Komponente vom DPG erstellt werden kann.

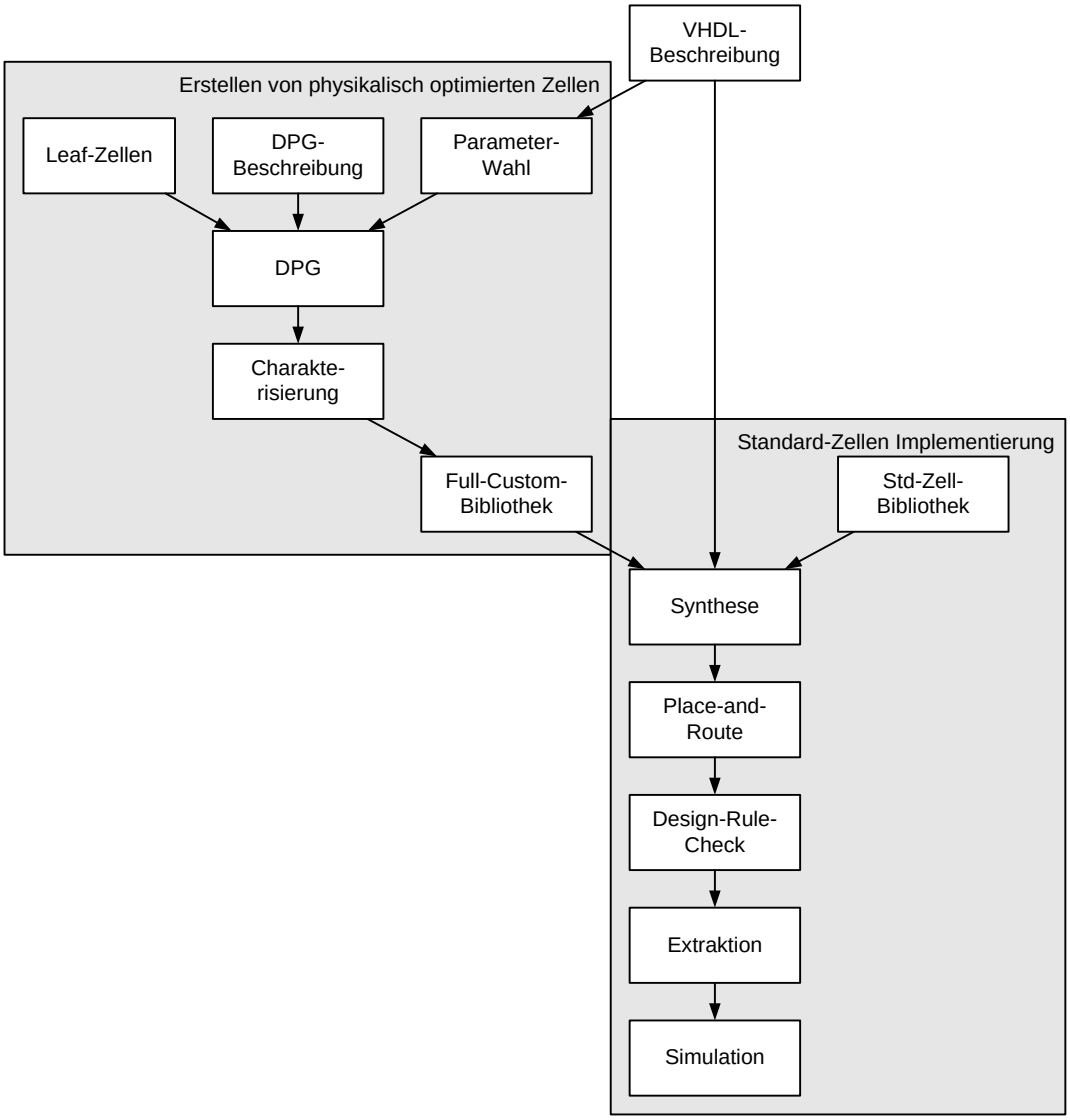


Abbildung 3.12: Design-Flow für die VLSI-Implementierung von NoC-Komponenten mit Standardzellen erweitert um physikalisch optimierte Zellen

Damit eine Teilkomponente in den Standardzellen Entwurfsprozess integriert werden kann, muss sie für die unterschiedlichen Arbeitspunkte der Transistoren (Slow, Typical und Fast) charakterisiert werden. Diese Ergebnisse werden in einer entsprechenden Bibliothek zusammengefasst. Die Charakterisierung umfasst die Abmaße und Fläche des Layouts, wie auch das zeitliche Verhalten. Für Eingangsports der Komponente, die direkt oder indirekt mit

einem Register verbunden sind, müssen die Setup- und Hold-Zeiten bezüglich des Taktsignals für verschiedene Flankensteilheiten des Takt- und des Datensignals bestimmt werden. Zur Bestimmung der Setup- und Hold-Zeiten wird die Teilkomponente wiederholt simuliert, wobei die Setup- bzw. die Hold-Zeit sukzessive verringert wird, bis die korrekte Funktion der Komponente nicht mehr sichergestellt ist. Die letzte Setup- bzw. Hold-Zeit, bei der die Schaltung noch funktional korrekt arbeitet, wird in die Charakterisierung eingetragen.

Das zeitliche Verhalten der Ausgangspins wird durch die Flankensteilheit des Ausgangssignals, sowohl für eine steigende wie auch für eine fallende Signalflanke, definiert. Diese werden für unterschiedliche Lastkapazitäten und unterschiedliche Eingangsflanken bestimmt. Neben der Charakterisierung des zeitlichen Verhaltens und der Fläche wird auch die Eingangskapazität des Eingangsports benötigt. Optional ist eine Bestimmung der Verlustleistung.

3.8 Spiral-Modell

Ein Nachteil der zu Beginn des Kapitels vorgestellten Entwurfsmethodik, die auf dem Wasserfall-Modell basiert, ist, dass die Anzahl der Iterationen im Rahmen des NoC-Entwurfs stark von der Erfahrung des Entwicklers abhängig. Daher ist der Entwurfsaufwand für ein NoC nur schwer abzuschätzen.

Eine Alternative, die auf eine systematische Untersuchung des Entwurfsraums setzt, ist das aus der Software-Entwicklung adaptierte Spiralmodell [92], das eine Weiterentwicklung des zu Beginn dieses Kapitels vorgestellten Wasserfall-Modells darstellt.

Ziel dieses Entwurfsmodells ist die systematische Exploration des Entwurfsraumes von NoC für ein gegebenes System, dessen Verkehrsanforderungen durch einen Task-Graphen beschrieben sind. Um dieses Ziel zu erreichen, werden mehrere Iterationen des Entwurfsprozesses für NoC, die aus den Phasen Parameterauswahl, Implementierung, Kosten- und Performance-Bestimmung sowie der Bewertung der untersuchten NoC besteht, durchlaufen.

Bei den ersten Iterationen werden diejenigen Parameter variiert, bei denen der Einfluss auf die Kosten und Performance am größten ist (vgl. Tabelle 4.7). Bei den folgenden Iterationen sinkt der Einfluss der Parameter auf Kosten und Performance von Iteration zu Iteration. Gleichzeitig wird die Abstraktion von der tatsächlichen Implementierung des NoC verringert. Bei der Parametervariation wird für den zu variierenden Parameter eine vollständige Suche durchgeführt, d.h. jede mögliche Parametervariante wird untersucht.

Im gleichen Maße wie die Abstraktion eines NoCs von Iteration zu Iteration sinkt, steigen die Genauigkeit der Performance- und Kostenschätzung und die für die Iteration benötigte Zeit.

Im Folgenden wird ein exemplarischer Ablauf des NoC-Entwurfs mit Hilfe dieser Entwurfsmethodik skizziert, der in Abbildung 3.13 dargestellt ist.

Der Prozess beginnt im Ursprung des Koordinatensystems mit der ersten Parameterwahl. Dabei werden alle für den gegebenen Task-Graphen möglichen und sinnvollen Topologien ausgewählt. Im nächsten Schritt werden für die gewählten NoC-Topologien NoC in der abstrakten C++-basierten Beschreibung erstellt (Implementierung). Anschließend wird ein Mapping des Task-Graphen auf die einzelnen NoC durchgeführt. Mit diesem Mapping lässt sich die Performance des NoCs mit Hilfe der statischen Performance-Analyse bestimmen. Die Performance wird in dieser Iteration durch die gewichtete Anzahl Hops ausgedrückt. Weiterhin werden die durch das NoC verursachten Kosten, in Form der benötigten Fläche, mit Hilfe der in Kapitel 5 vorgestellten Kostenfunktionen bestimmt. Basierend auf diesen Kosten- und Performance-Maßen werden die Pareto-optimalen NoC bestimmt. Diese bilden die Basis für die Parameterwahl der nächsten Iteration.

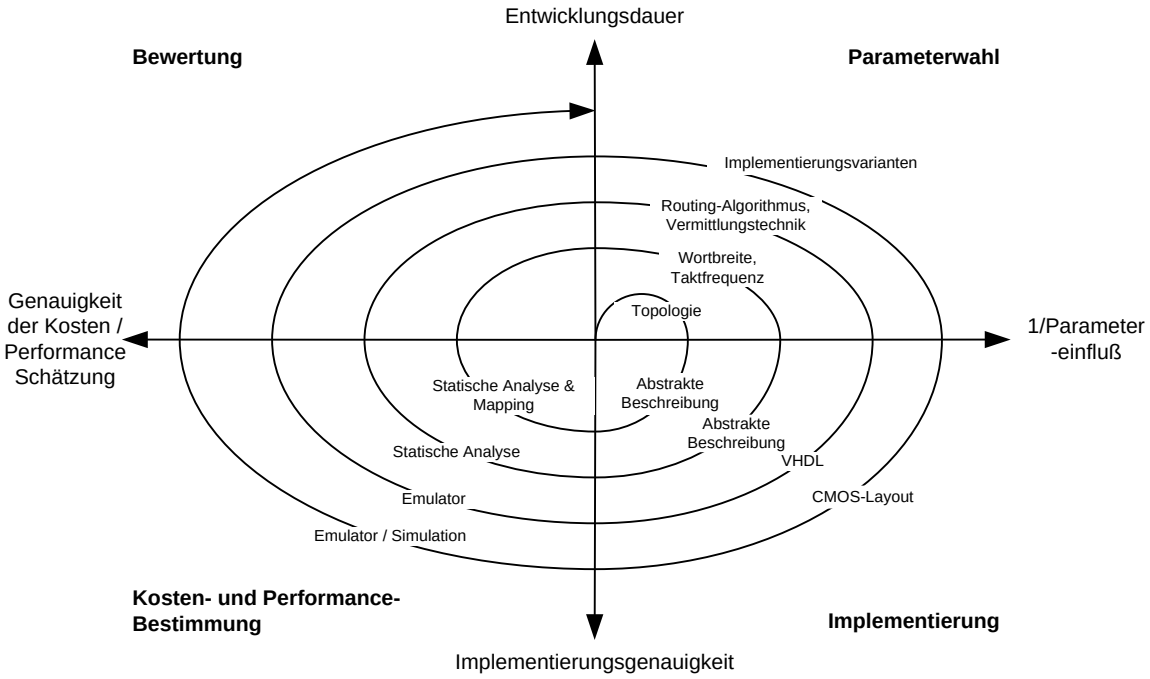


Abbildung 3.13: Spiralmodell der NoC-Entwicklung

In der nächsten Iteration werden die Wortbreite und die Taktfrequenz des NoC bestimmt. Die Implementierung erfolgt hier bereits in VHDL. Der für die Performance-Schätzung benötigte Emulator nutzt diese VHDL-Beschreibung. Als Performance-Bewertung wird lediglich ausgewertet, ob bei den Versuchen, die auf dem Emulator durchgeführt werden, die vorgegebenen Deadlines des Task-Graphen eingehalten werden. Als Kosten-Maß dienen die benötigte Fläche und die verursachte Verlustleistung. Beide Maße werden wiederum mit Hilfe von Kostenfunktionen bestimmt.

In den nächsten Iterationen werden für weitere Parameter wie beispielsweise Vermittlungstechnik, Routing-Algorithmus, und, in Abbildung 3.13 nicht dargestellt, die Verbindungsfehlerbehandlung mit dem gleichen Vorgehen optimale Parametervarianten bestimmt.

Sobald alle NoC-Parameter optimiert worden sind, werden in einer abschließenden Iteration unterschiedliche Implementierungsvarianten untersucht. Diese beeinflussen lediglich die durch das NoC verursachten Kosten. Die Implementierung erfolgt hier bereits als VLSI-Implementierung mit Hilfe von Standardzellen, gegebenenfalls erweitert um physikalisch optimierte Makros. Anschließend erfolgt eine Verifikation der VLSI-Implementierung durch die Simulation einzelner NoC-Komponenten. Die verwendeten Modellfunktionen werden durch die Ergebnisse der Implementierung (Fläche) und der Simulation (Verlustleistung) optimiert.

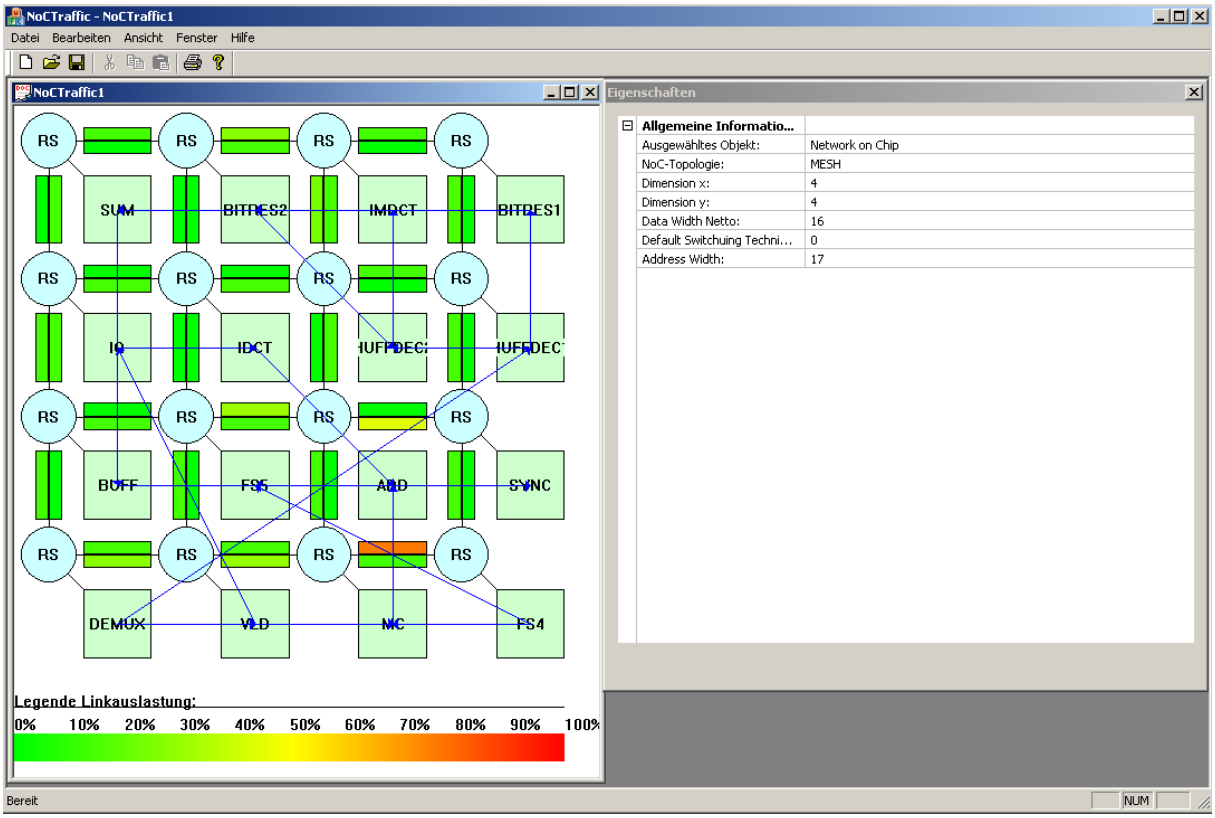


Abbildung 3.14: Grafische Benutzeroberfläche des Software-Werkzeugs zum automatischen NoC-Entwurf

Eine weitere, applikationsspezifische Optimierung des NoC kann durch das Entfernen nicht genutzter NoC-Komponenten erfolgen. Dadurch wird jedoch die Flexibilität des NoCs eingeschränkt, da evtl. nicht mehr alle an das NoC angeschlossenen funktionalen Einheiten miteinander kommunizieren können.

Durch die Systematik dieses Entwurfsprozesses ist eine Realisierung in einem Entwurfswerkzeug möglich, das im Rahmen dieser Arbeit erstellt worden ist. Dieses Entwurfswerkzeug umfasst die Parameterauswahl in den verschiedenen Iterationsstufen, das Mapping und die statische Performance-Analyse sowie die Kostenschätzung. Weiterhin werden die für die dynamische Performance-Analyse benötigten FPGA-basierten NoC-Emulatoren automatisch erzeugt und die Experimente zur Performance-Bestimmung des NoC auf einem FPGA durchgeführt. Nach den einzelnen Iterationen ist jeweils optional eine applikationsspezifische Optimierung des NoC möglich. Abschließend werden die benötigten NoC-Komponenten im Rahmen des implementierten Werkzeuges mit Hilfe von Standardzellen implementiert und verifiziert. Diese Implementierung kann dann die Basis für eine weitergehende, manuelle Optimierung des NoC bilden.

Teil dieses Entwurfswerkzeuges für NoC ist eine grafische Benutzeroberfläche, in der die Topologie des NoC und der auf das NoC abgebildete Task-Graphen dargestellt werden (Abbildung 3.14). Innerhalb der (quadratischen) Network-Interfaces wird der Name des auf das NoC abgebildeten Tasks dargestellt. Die dünnen Pfeile zeigen, welches Network-Interface mit welchem Network-Interface kommuniziert. Die Farben der Links repräsentieren die während der statischen Performance-Analyse ermittelte Auslastung des jeweiligen Links. In dem rechten Fenster werden die Eigenschaften und die verursachten Kosten der aktuell ausgewählten NoC-Komponente bzw. des gesamten NoC dargestellt.

Tabelle 3.3: Übersicht über die Iterationen des Spiral-Entwurfsmodells

Iteration	**NoC-Parameter**	**Implementierung / Beschreibung**	**Kosten und Performance Bestimmung durch**
Iteration 1	Topologie	Abstrakte Beschreibung	Mapping, Statische Analyse & Kostenmodelle
Iteration 2	Wortbreite, Taktfrequenz	Abstrakte Beschreibung	Statische Analyse & Kostenmodelle
Iteration 3	Routing-Algorithmus, Vermittlungstechnik	VHDL-Beschreibung	Emulation & Kostenmodelle
Iteration 4	Implementierungs-varianten	CMOS-Layout	Emulation, Simulation und CMOS Layout

In Tabelle 3.3 sind die einzelnen Iterationen des Entwurfswerkzeugs dargestellt. Der Parameterraum, der für die jeweiligen Parameter untersucht wird, ist in Anhang A beschrieben. Die Kosten, die in jeder Iteration mit wachsender Genauigkeit beschrieben werden, sind die benötigte Fläche und Verlustleistung. Zusätzlich wird verifiziert, dass die vorgegebenen Verkehrsanforderungen für die gewählten NoC-Parameter in den jeweiligen Iterationen erfüllt werden können. In den Iterationen 3 und 4 wird zusätzliche eine funktionale Verifikation des VHDL-Codes des NoC vorgenommen. In der Iteration 4 würde darüber hinaus sichergestellt, dass die angeforderte Taktfrequenz auch erreicht werden kann. Die

identifizierten Pareto-optimalen NoC sind hierbei in der Regel die Basis für eine weitere, individuelle Optimierung.

4 Implementierung und Bewertung ausgewählter NoC und NoC-Komponenten

Im folgenden Kapitel wird der modulare und generische Aufbau des NoCs und der NoC-Komponenten erläutert. Anschließend werden verschiedene Implementierungsvarianten für NoC miteinander bezüglich ihrer korrespondierenden Kosten sowie der Performance quantitativ untersucht.

4.1 Modulares und generisches NoC

Ein NoC besteht aus den drei Basis-Komponenten Network-Interface, Routing-Switch und Link. Durch definierte Schnittstellen können diese beliebig miteinander kombiniert werden. Die Anordnung der einzelnen NoC-Komponenten ist hier durch die Topologie definiert. NoC-Parameter können generisch sein, wie zum Beispiel die Datenwortbreite, die Anzahl der Ports eines Routing-Switches oder die Anzahl der an das NoC angeschlossenen funktionalen Einheiten. Eine Übersicht, über die im Rahmen dieser Arbeit untersuchten Parameter ist in den Tabellen in Anhang A gegeben.

Die einzelnen Komponenten sind in sich wiederum modular aufgebaut. Dadurch können unterschiedliche Funktionalitäten durch Austauschen einzelner Module in das NoC integrierte oder entfernt werden. Zum Beispiel können durch Austauschen eines Moduls unterschiedliche Fehlerschutzcodierungen verwendet werden.

Exemplarisch wird zunächst die Schnittstelle zwischen Routing-Switch, Network-Interface und Link erläutert. Diese besteht aus Datenleitungen und Steuerleitungen in Richtung der Datenübertragung (Datenkanal) sowie aus Steuerleitungen in entgegen gerichteter Richtung (Rückkanal), die beispielsweise den Verbindungsaufbau bestätigen. Daten, die über das NoC übertragen werden, können Nutz- oder Kontrolldaten sein. Kontrolldaten werden über eine Steuerleitung angezeigt. Eine weitere Steuerleitung (Valid), die zwischen den NoC-Komponenten existiert, zeigt an, ob die Daten (Nutz- und Kontrolldaten) gültig sind.

Kontrolldaten können aus einem oder mehreren Kontrollworten bestehen. Diese Kontrollworte werden verwendet, um zum Beispiel Verbindungen bei Leitungsvermittlung auf- oder abzubauen oder den Beginn und das Ende eines Datenpaketes beim Wormhole-Routing anzuzeigen. Ein solches Kontrollwort enthält dementsprechend Informationen über die Art des Kontrollwortes (z.B. Verbindungsauf- oder -abbau) sowie zusätzliche Informationen, wie beispielsweise die Zieladresse oder eine Priorität der zu übertragenden Nutzdaten.

Der korrekte oder auch fehlgeschlagene Verbindungsaufbau, sowie der fehlerfreie Empfang von Daten werden über den Rückkanal übertragen. Dieser ist, je nach gewählten NoC-Parametern, zwei oder drei Bit breit.

Der modulare Aufbau des Routing-Switches ist in Abbildung 4.1 dargestellt. Der Routing-Switch besteht aus einem Arbiter, der den Zustand der Ausgangsports verwaltet und den Zugriff auf die Ausgangsports gewährt oder verbietet. Der Router bestimmt gemäß dem Routing-Algorithmus, der Position des Routing-Switches im Netzwerk und der Zieladresse der zu übertragenden Daten, welcher Eingangsport mit welchem Ausgangsport verbunden werden soll. Diese Verbindung wird durch den Switch realisiert, der durch den Arbiter gesteuert wird. Dieser stellt auch die Verbindung des Rückkanals her. Zusätzlich befinden sich an jedem Eingangsport *i* noch optionale Module mit Registern (R_{IPi}), zur Busdecodierung (BDC_{IPi}) sowie zur Fehlerkennung und -korrektur (ECC_{IPi}).

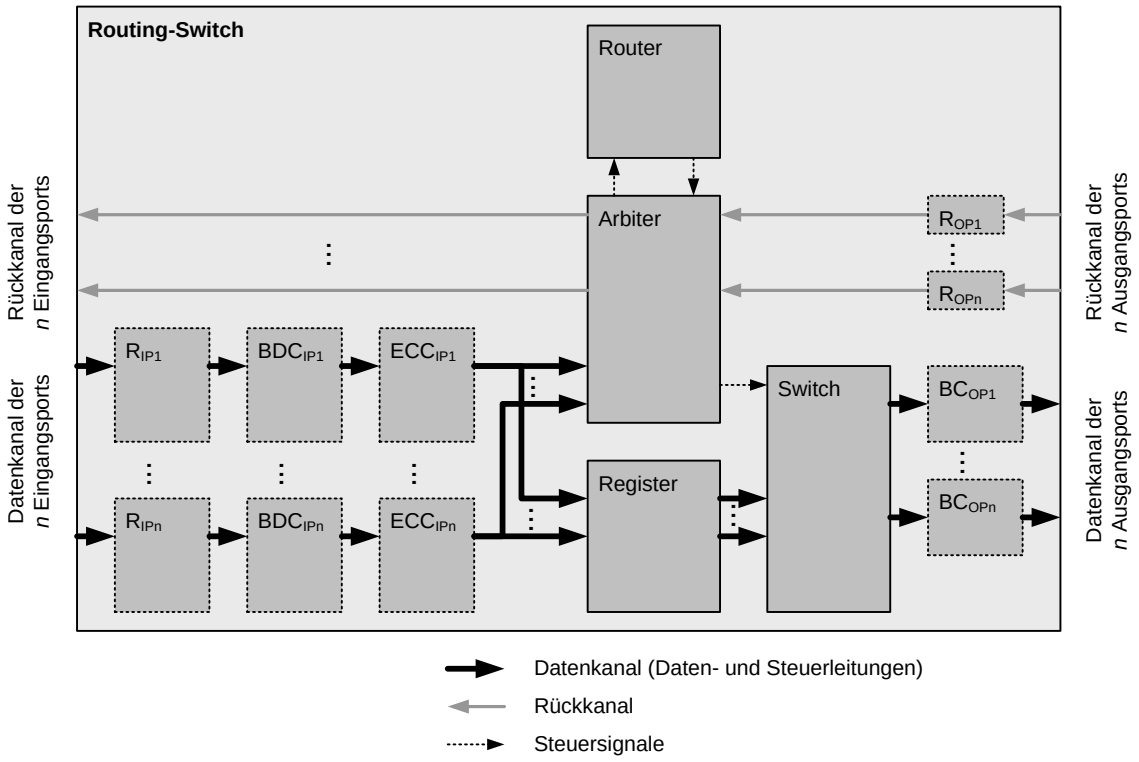


Abbildung 4.1: Modularer Aufbau von Routing-Switches mit n Ports. Optionale Module sind gestrichelt eingezeichnet

Ein Network-Interface besteht aus einem Modul zum Senden von Daten (*NI-Send*) und einem Modul zum Empfangen von Daten (*NI-Receive*), vgl. Abbildung 4.2. Der sendende Teil besteht aus einem Kontrollblock (*Control*), in dem die Vermittlungstechnik realisiert wird. Probleme beim Verbindungsaufbau werden von dem Block Connection Failure Handling (*CFH*) behandelt. Eine Fehlerschutzcodierung der Daten kann in dem Block *ECC/EDC* erfolgen bevor im Block Buscodierung (*BC*) die Daten zur Verlustleistungsreduktion erneut

codiert werden können. Datenfehler, die während der Übertragung über das NoC aufgetreten sind, werden im Block Error Handling behandelt. Der Speicher (*Send-Buffer*) wird für verschiedene Vermittlungstechniken oder den Datenfehlerschutz benötigt, um Daten erneut über das NoC senden zu können.

Im Modul *NI-Receive* werden zunächst im Block Busdecodierung (*BDC*) die über das NoC übertragenen Daten decodiert und eine Fehlererkennung und ggf. Fehlerkorrektur im Block *ECC/EDC* durchgeführt. Der Block *Error Handling* ist für die Fehlerbehandlung von Datenfehlern zuständig und im Block *Control* wird das empfängerseitige Protokoll implementiert.

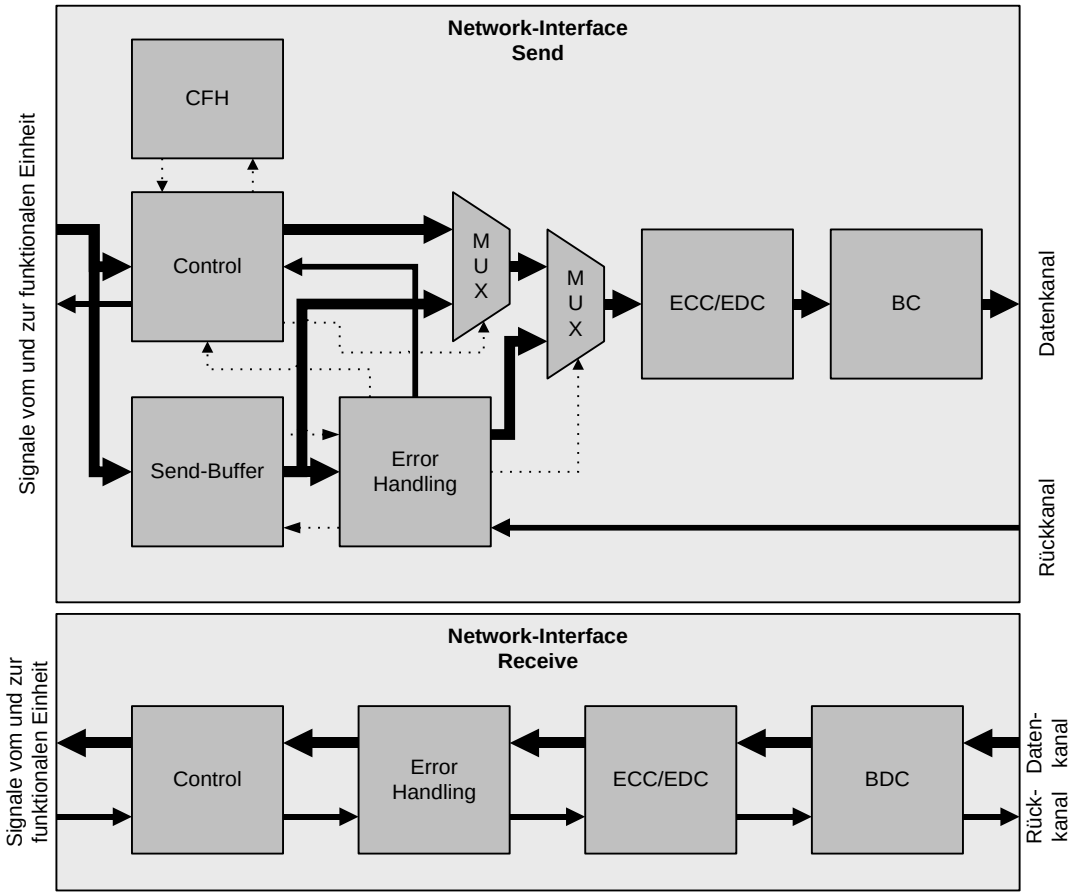


Abbildung 4.2: Modularer Aufbau des Network-Interfaces

Mit Hilfe dieser allgemeinen Beschreibung von NoC-Komponenten wurde eine VHDL-Bibliothek erstellt, welche die in Anhang A aufgeführten Parameter unterstützt. Durch Verwendung dieser Bibliothek und eines in Kapitel 3.5 vorgestellten Generators für NoC, kann automatisch eine VHDL-Beschreibung eines NoC erstellt werden, die als Basis für eine VLSI-Implementierung dient.

4.2 NoC-Protokoll

Das Protokoll der Datenübertragung in dem hier implementierten NoC wir durch die Vermittlungstechnik, den Routing-Algorithmus und die Behandlung von Verbindungsfehlern definiert. Weitere Funktionalitäten wie Datenfehlerschutz oder Bus-Codierung können in dieses Protokoll integriert werden. Die verwendeten Routing-Algorithmen unterstützen ein verteiltes Minimal Path-Routing für Unicast-Übertragungen. Auftretende Verbindungsfehler durch blockierte NoC-Ressourcen werden durch das Loss-Modell (vgl. Kapitel 2.2.7) gelöst. Durch dieses Protokoll können in diesem NoC keine Deadlocks auftreten. Eine genaue Beschreibung und Bewertung der implementierten Routing-Algorithmen und des Verhaltens bei Verbindungsfehlern sind in Kapitel 4.5.2 gegeben.

Im Folgenden werden die zwei implementierten Protokolle vorgestellt, die auf Leitungsvermittlung und auf Wormhole-Routing basieren. Grundsätzlich kann in einem NoC auch eine Kombination aus beiden Protokollen angewendet werden.

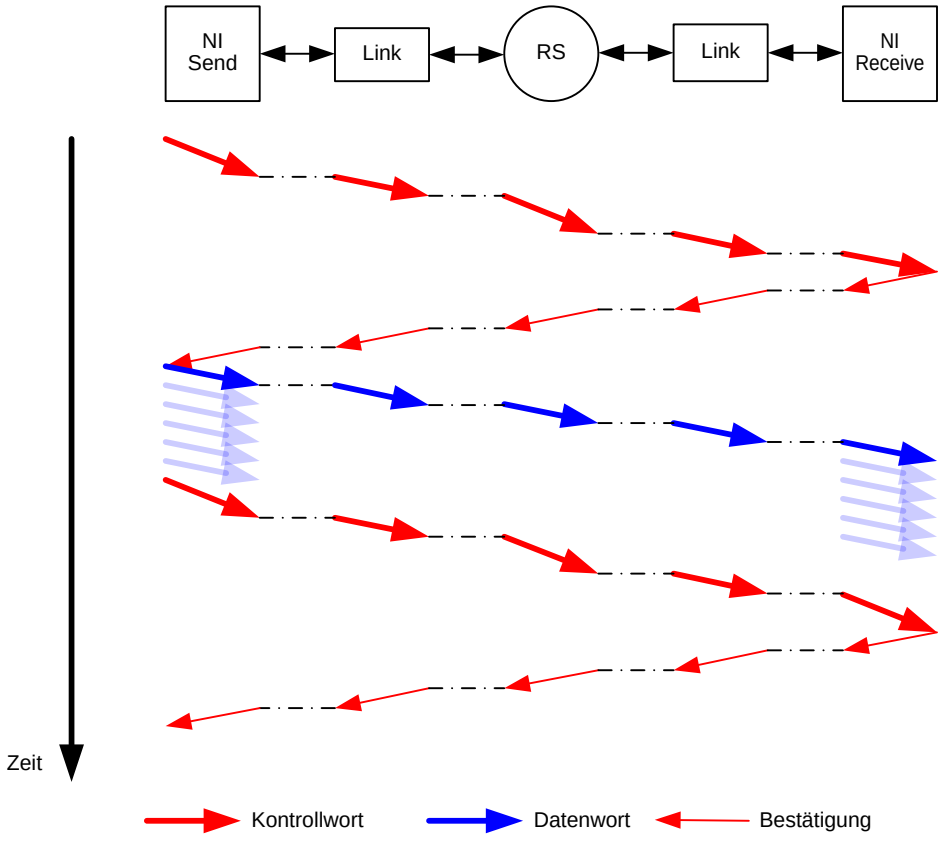


Abbildung 4.3: Timing-Diagramm für eine erfolgreiche Datenübertragung bei Leitungsvermittlung

Bei Leitungsvermittlung wird zunächst eine Verbindung aufgebaut. Dies ist in Abbildung 4.3 dargestellt. Dazu werden von dem sendenden Network-Interface (*NI-Send*) Kontrollworte erzeugt, die unter anderem die Zieladresse enthalten. Diese werden über einen Link zum

nächsten Routing-Switch übertragen, der anhand der Zieladresse und des Routing-Algorithmus das Kontrollwort von dem Eingangs- an einen entsprechenden Ausgangsport weiterleitet. Das wird so lange wiederholt, bis das empfangende Network-Interface (*NI-Receive*) das Kontrollwort empfängt. Das Modul *NI-Receive* signalisiert den erfolgreichen Verbindungsaufbau durch ein Bestätigungssignal über den Rückkanal, sobald es bereit ist, Daten zu empfangen. Das Bestätigungssignal wird zum Modul *NI-Send* übertragen, welches nach dem Empfang der Bestätigung mit der eigentlichen Datenübertragung beginnt. Wenn die Datenübertragung abgeschlossen ist, wird der Verbindungsabbau durch ein weiteres Kontrollwort vom Modul *NI-Send* initiiert. Die blockierten Netzwerk-Ressourcen werden dann durch die Bestätigung des Moduls *NI-Receive* wieder freigegeben.

Bei dem Verbindungsaufbau können Fehler auftreten, wenn zum Beispiel eine Netzwerk-Ressource, die benötigt wird, nicht zur Verfügung steht ist. Das kann beispielsweise ein belegter Routing-Switch oder Link sein. Auch ein Network-Interface, das nicht bereit ist, Daten zu empfangen, kann einen solchen Verbindungsfehler auslösen. In diesem Fall wird dieser Fehler durch ein entsprechendes Bestätigungssignal angezeigt und die bereits belegten Netzwerk-Ressourcen werden wieder freigegeben. Wie auf einen Fehler beim Verbindungsaufbau reagiert wird, ist in Kapitel 2.2.9 beschrieben. Beispielsweise kann der Verbindungsaufbau erneut gestartet werden (vgl. Abbildung 4.4).

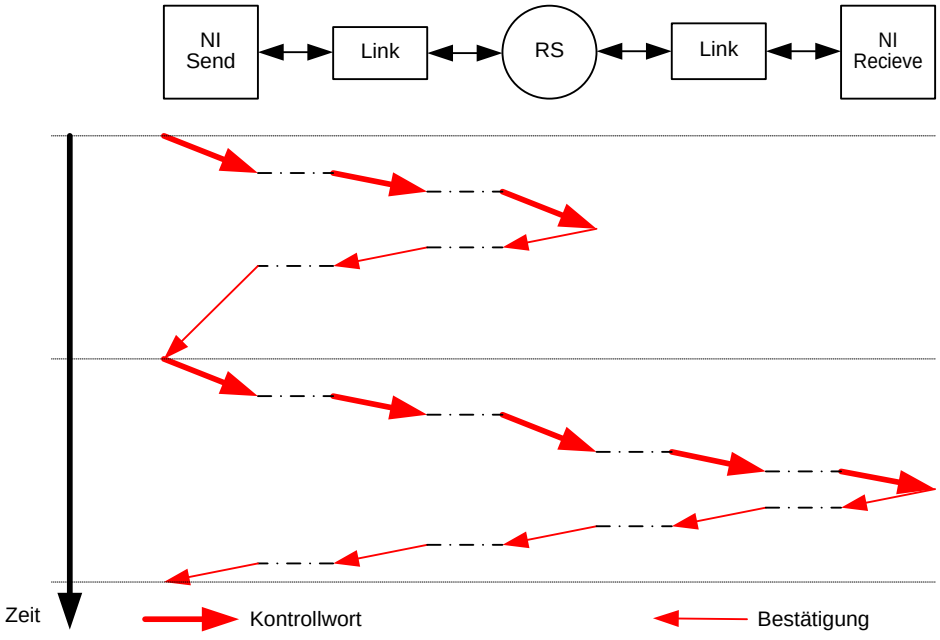


Abbildung 4.4: Timing-Diagramm für einen erfolglosen Verbindungsaufbau bei Leitungsvermittlung und Wormhole-Routing

Die zweite implementierte Vermittlungstechnik ist das Wormhole-Protokoll (Abbildung 4.5). Die Datenpakete, die über das Netz geschickt werden, können dabei eine beliebige Größe

haben. Es muss jedoch sichergestellt werden, dass das empfangende Network-Interface die empfangenen Daten aufnehmen kann. Ein Flit besteht hier aus einem einzelnen Datenwort ohne weitere Steuerinformationen.

Es wird bei diesem Protokoll im ersten Flit ein Kontrollwort und anschließend, ohne eine Bestätigung abzuwarten, die Daten-Flits übertragen. Die Übertragung der Daten-Flits erfolgt analog zur Übertragung der Daten bei Leitungsvermittlung. Der korrekte Empfang des Kontrollwortes wird auch hier bestätigt, sofern des NI-Receive bereit ist, Daten zu empfangen.

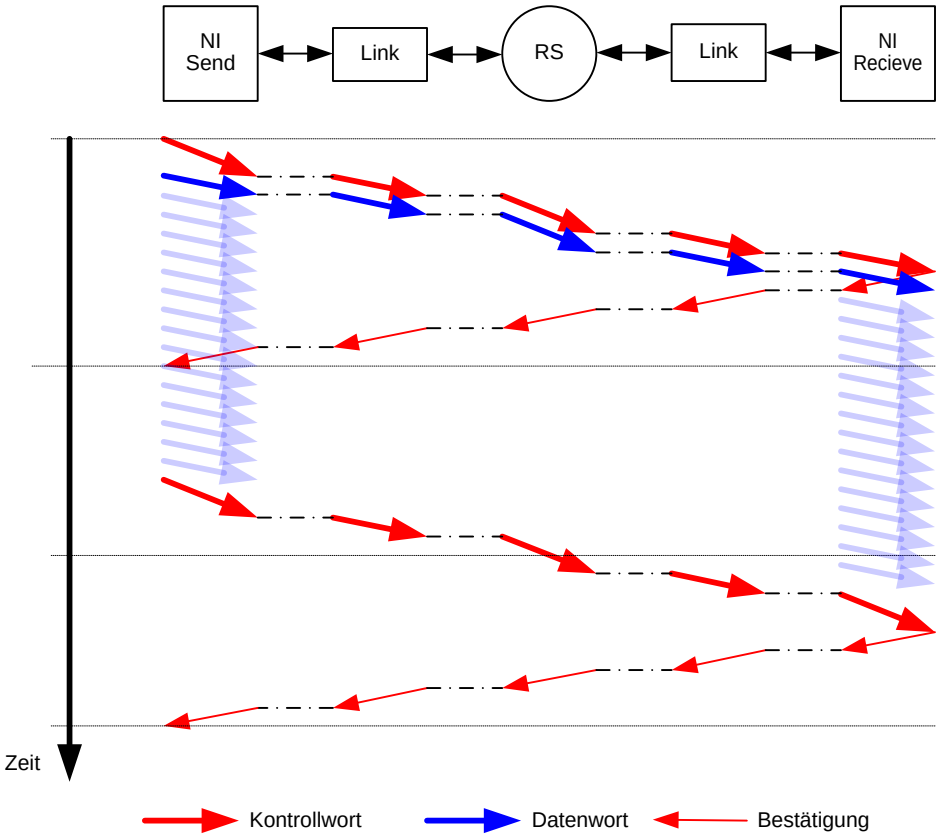


Abbildung 4.5: Timing-Diagramm für eine erfolgreiche Datenübertragung bei dem Wormhole-Protokoll

Bei einer erfolglosen Übertragung wird diese ebenfalls über ein entsprechendes Bestätigungssignal angezeigt, alle bisher vom NI-Send gesendeten Daten verworfen und die belegten Netzwerkressourcen wieder freigegeben. Bei einem erneuten Versuch, die Daten zu übertragen, müssen alle bereits gesendeten Daten erneut gesendet werden. Daher müssen sie in einem Speicher im Network-Interface vorgehalten werden. Die Größe des verwendeten Speichers beeinflusst die Performance des NoC. Ist der Speicher zu klein gewählt, können nicht alle Daten gespeichert werden, bevor die Bestätigung des Verbindungsaufbaus erfolgt ist. Dann müssen Wartezyklen eingefügt werden.

Die optimale Speichergröße bestimmt sich aus der maximal für ein sendendes Network-Interface auftretenden Latenz. Diese muss nicht für jedes Network-Interface eines NoC identisch sein. Eine weitere Vergrößerung des Speichers über diesen Wert hinaus resultiert nicht in einem weiteren Performance-Gewinn.

4.3 Kosten- und Performance-Bestimmung von NoC-Komponenten

Für die Performance-Untersuchung wurde das zu untersuchende NoC auf dem FPGA-basierten Emulator vermessen. Eine genaue Beschreibung des Emulators ist in Kapitel 3.6.1 gegeben. Der Datenverkehr, mit dem ein Performance-Vergleich der unterschiedlichen NoC durchgeführt wird, besteht aus zufälligem Datenverkehr mit einer Datenblockgröße von 100 Datenworten. Die angeforderte Auslastung des NoC wird zwischen 10 % und 80 % variiert. Zur Bewertung wurden jeweils acht Messungen durchgeführt, wobei jeweils 25.000 Datenblöcke übertragen wurden.

Um die Kosten für eine NoC-Implementierung zu bestimmen, wurden die einzelnen NoC-Komponenten (Network-Interface und Routing-Switch) exemplarisch in einer 90 nm Technologie implementiert. Zur Synthese wurde die Software Design Compiler von Synopsys [93] verwendet, der Place and Route-Schritt wurde mit Encounter [94] durchgeführt. Zur Extraktion und für den Design Rule Check wurde das Cadence Design-Framework [95] verwendet. Die abschließende Simulation zur Verifikation und Verlustleistungsbestimmung erfolgte mit NanoSim [96]. Die Kosten des gesamten NoC werden durch Superposition der einzelnen Implementierungs-Ergebnisse ermittelt.

Die Fläche einer NoC-Komponente bzw. der einzelnen Module einer Komponente werden aus dem platzierten und gerouteten Layout bestimmt. Die Aufteilung auf die einzelnen Module (beispielsweise des Switches eines Routing-Switches) einer Komponente kann nicht allein aus dem Layout erfolgen, da während des Place and Route-Schrittes die Hierarchiestufen aufgelöst werden. Die Fläche der einzelnen Module wird anhand des Flächenanteils des Moduls nach der Synthese aus der Gesamtfläche der Komponente bestimmt.

Die Bestimmung der Verlustleistung erfolgt durch eine Simulation einer einzelnen Datenübertragung mit 80 Datenworten. Diese Anzahl hat sich bei vorhergehenden Untersuchungen als guter Kompromiss aus Simulationszeit und Genauigkeit herausgestellt. Die übertragenen zufälligen Datenworte verursachen eine Schaltaktivität σ auf einer einzelnen Verbindungsleitung von ca. 25%. Für diesen Datenverkehr wird im Folgenden die durchschnittlich benötigte Verlustleistung angegeben. Die Verlustleistung wird immer für eine komplette NoC-Komponente (z.B. Routing-Switch) bestimmt.

Die Position einer NoC-Komponente innerhalb des NoCs hat, bei ansonsten identischen Parametern, einen vernachlässigbaren Einfluss auf die Kosten, die von dieser NoC-

Komponente verursacht werden. Daher wird zur Kostenabschätzung lediglich eine Komponente implementiert. Aufgrund dieser Implementierungen können die Kosten des gesamten NoCs extrapoliert werden. Für die folgenden Untersuchungen wurden die in Tabelle 4.1 angegebenen NoC-Parameter als Referenz verwendet.

Tabelle 4.1: Referenzparameter für Implementierungen von NoC-Komponenten

Globale Parameter	**Wert**
Vermittlungstechnik	Wormhole-Routing
Bruttodatenwortbreite	32
Nettodatenwortbreite	32
Topologie	Mesh
Größe	5x5 (25 Teilnehmer)
Maximale Taktrate	500 MHz
Technologie	TSMC90 (90 nm, 4 Metall-Layer)

Network-Interface	
Bus-Codierung	Keine
Fehlererkennung	Keine
Fehlerbehandlung	Send and Forget
Verbindungsfehlerbehandlung	Feste Anzahl an Wartezyklen (31)
Lastkapazität an den Ausgängen zum NoC	0,0018 pF (direkter Anschluss von Network-Interface an Routing-Switch, ohne Link)
Prioritätsunterstützung	Keine

Routing-Switch	
Implementierung	Seriell, mit zusätzlichen Registern an den Eingangsports
Bus-Codierung	Keine
Fehlerschutz	Keiner
Routing-Algorithmus	XY-Routing
Prioritätsunterstützung	Keine
Lastkapazität an den Ausgängen zum NoC	0,415 pF (abhängig vom Link)

Link	
Länge	1 mm
Abstand zwischen Verbindungsleitungen	einfacher minimaler Abstand
Konnektivität	2 uni-direktionale Links
Anzahl der Register auf einer Verbindungsleitung	0

4.4 Network-Interface

Zunächst wird in diesem Abschnitt eine exemplarische Implementierung eines Network-Interfaces vorgestellt. Abweichend zu den in Tabelle 4.1 gewählten Parametern wird hier eine einfache Fehlerkorrektur (Hamming Code) und zur Fehlerkorrektur das Send and Wait-Protokoll mit Sliding Window Enhancement verwendet, damit die Blöcke für den

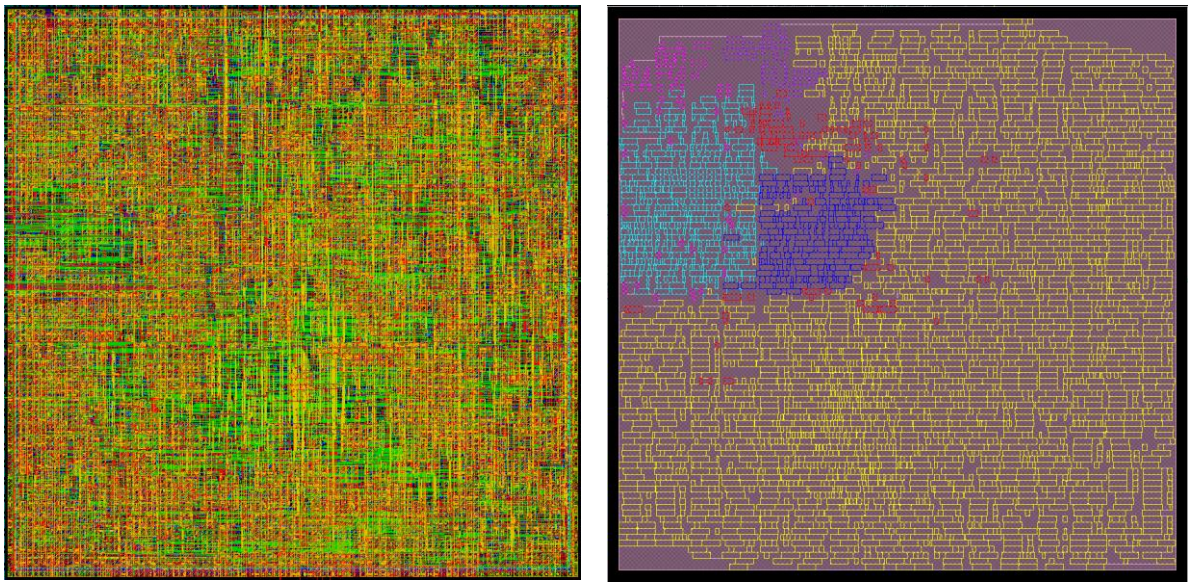
Datenfehlerschutz innerhalb des Network-Interfaces eine relevante und darstellbare Größe besitzen. Daraus resultiert bei einer Nettodatenwortbreite von 32 Bit eine Bruttodatenwortbreite von 39 Bit. Diese geänderten Parameter sind in Tabelle 4.2 zusammengefasst.

Das Layout des implementierten Network-Interfaces ist Abbildung 4.6 (a) dargestellt. In Abbildung 4.6 (b) sind die einzelnen Module des Network-Interface im Layout eingefärbt. Es ist zu erkennen, dass der Speicher einen Großteil der Fläche des Network-Interfaces einnimmt. Weitere große Anteile werden von den Modulen zur Fehlerschutzcodierung und -decodierung benötigt. Die Blöcke, die steuernde Funktionen haben wie die Controller im sendenden und empfangenden Teil, oder auch der Block *Connection Failure Handling* haben lediglich einen kleinen Anteil an der Fläche des Network-Interfaces. Die genaue Flächenverteilung ist in Abbildung 4.7 aufgetragen.

Tabelle 4.2: Abweichende Parameter für die exemplarische Implementierung des Network-Interfaces

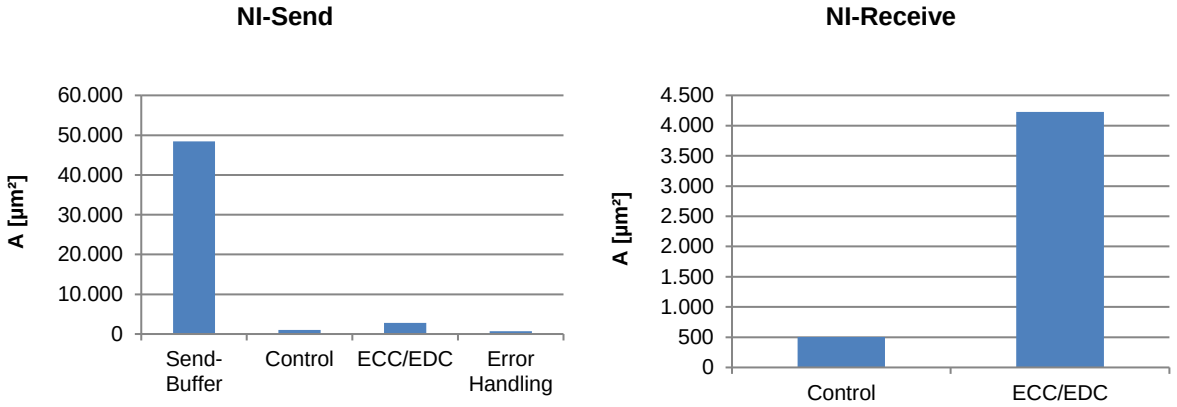
Parameter	**Wert**
Bruttodatenwortbreite	39
Fehlererkennung	Hamming Code (SEC)
Fehlerbehandlung	Send and Wait with Retransmission und Sliding Window Enhancement

Die Verlustleistung, die von einem Network-Interface verbraucht wird, hängt neben den gewählten NoC-Parametern auch von dem Zustand ab, in dem sich das Network-Interface befindet. Das Network-Interface kann sich in dem Zustand *Idle* befinden, wenn keine Daten übertragen und keine Verbindungen auf- oder abgebaut werden. Daneben können sich sowohl der sendende wie auch der empfangende Teil in den fünf in Tabelle 4.3 aufgeführten Phasen einer Datenübertragung befinden.



(a) (b)

Abbildung 4.6: Layout (a) eines Network-Interfaces. In (b) ist die Verteilung der Fläche auf den sendenden Teil (Send-Buffer (gelb), Controller (rot), ECC/EDC (blau) Error Handling (lila)) und den empfangenden Teil (Controller (pink), ECC/EDC und Error Handling (türkis)) dargestellt.



(a) (b)

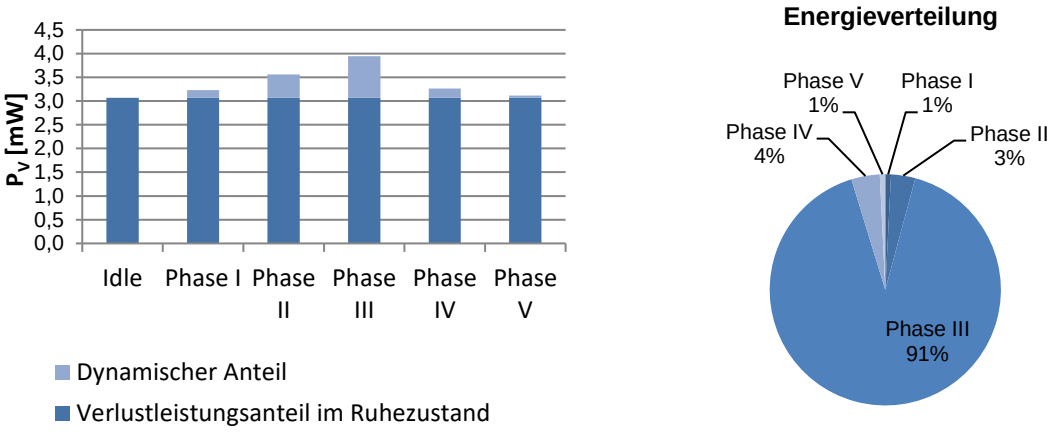
Abbildung 4.7: Fläche der einzelnen Module des exemplarischen Network-Interfaces für den sendenden Teil (a) den empfangenden Teil (b)

Für das exemplarische Network-Interface ergeben sich die in Abbildung 4.8 dargestellten Verlustleistungswerte für ein sendendes Network-Interface und in Abbildung 4.9 die Verlustleistung eines empfangenden Network-Interfaces. Die Verlustleistung wird dabei in zwei Anteile aufgeteilt: zum einen den Anteil, der im Zustand *Idle* verbraucht wird (Verlustleistungsanteil im Ruhezustand) und einen Anteil, der spezifisch für die jeweilige Phase der Datenübertragung ist (dynamischer Anteil).

Tabelle 4.3: Phasen einer Datenübertragung in einem Network-Interface

Übertragungs-Phase	**Beschreibung**
Idle	Kein Datenverkehr, lediglich Aktivität auf der Leitung des Taktsignals
I	Verbindungsaufbau bzw. Datenübertragung initiieren
II	Warten auf den Beginn der Datenübertragung
III	Datenübertragung
IV	Verbindungsabbau einleiten
V	Warten auf die Bestätigung des Verbindungsabbaus

Neben der Verlustleistung, die während der einzelnen Übertragungsphasen verbraucht wird, ist auch die Energie von Interesse, die benötigt wird, um Daten über ein NoC zu übertragen, bzw. die bei einer Datenübertragung in einem Network-Interface umgesetzt wird. Dazu wird eine exemplarische Datenübertragung über zwei Routing-Switches betrachtet, bei der 128 Datenworte übertragen werden. Die Energieverteilung auf die einzelnen Phasen der Datenübertragung ist, getrennt nach sendendem und empfangendem Network-Interface, auf der rechten Seite von Abbildung 4.8 bzw. Abbildung 4.9 aufgetragen.

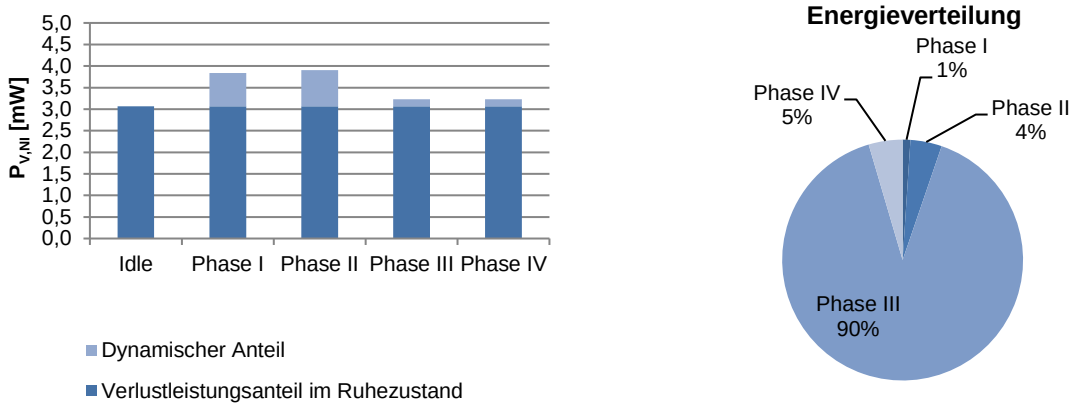


(a) (b)

Abbildung 4.8: Verlustleistung (a) und Energieverteilung (b) für die verschiedenen Phasen einer Datenübertragung eines Network-Interfaces beim Senden von Daten

Aus diesen Ergebnissen (Abbildung 4.8 (a) und Abbildung 4.9 (a)) lässt sich ableiten, dass ein Großteil der Verlustleistung (Takt-Netzwerk und Leakage) im Ruhezustand verbraucht wird. Der Anteil, der für Verbindungsauf- und -abbau, sowie die eigentliche Datenübertragung benötigt wird, ist vergleichsweise gering.

Der Großteil der Energie, die in einem Network-Interface benötigt wird, um Daten über ein NoC zu transportieren, wird während der eigentlichen Datenübertragung umgesetzt. Die Energie, für Verbindungsauf- und -abbau macht selbst bei kleinen Datenblöcken, wie sie hier verwendet wurden, nur einen Anteil von ca. 10 %, sowohl im Sende- wie auch im Empfangsbetrieb aus (vgl. Abbildung 4.8 (b) und Abbildung 4.9 (b)).

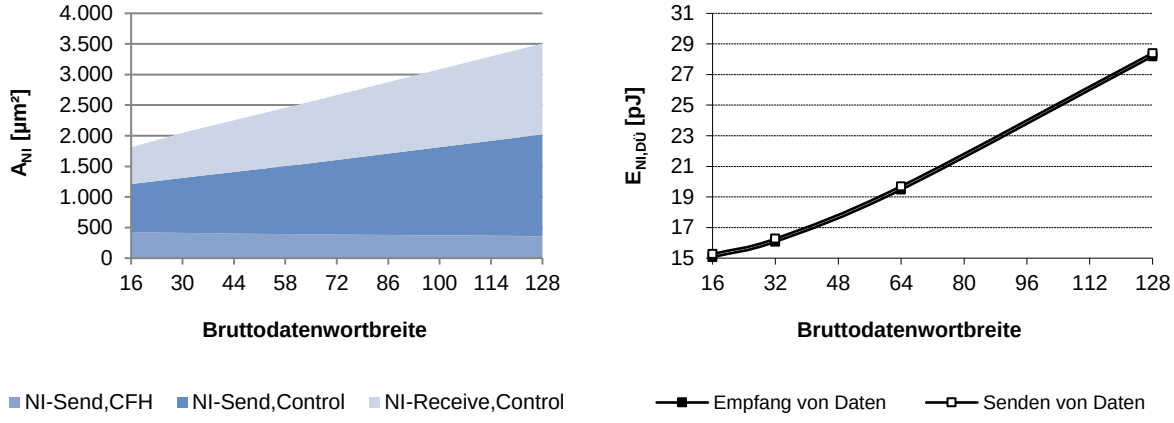


(a) (b)

Abbildung 4.9: Verlustleistung (a) und Energieverteilung (b) für die verschiedenen Phasen einer Datenübertragung eines Network-Interfaces beim Empfangen von Daten

4.4.1 Vermittlungstechnik

In diesem Abschnitt werden die beiden Vermittlungstechniken Leitungsvermittlung und Wormhole-Routing bezüglich der in einem Network-Interface verursachten Kosten verglichen. Neben der Vermittlungstechnik hat außerdem die Bruttodatenwortbreite einen erheblichen Einfluss auf die Kosten. In Abbildung 4.10 und Abbildung 4.11 ist die benötigte Fläche für die einzelnen Module des Network-Interfaces für beide Vermittlungstechniken dargestellt. Das Network-Interface besteht aus Modulen, die unabhängig von der Wortbreite sind (hier *NI-Send,CFH*), wie auch Modulen, die eine lineare Abhängigkeit von der Bruttodatenwortbreite (*NI-Send,Control*, *NI-Receive,Control*) besitzen.



(a) (b)

*Abbildung 4.10: Fläche und Energie einer Datenübertragung eines Network-Interfaces für verschiedene Wortbreiten bei **Leitungsvermittlung***

Die Energie für eine Datenübertragung im Network-Interface steigt ebenfalls mit der Datenwortbreite an. Der Unterschied zwischen einem Network-Interface im Sende- oder Empfangsbetrieb ist jedoch marginal (vgl. Abbildung 4.10 (b)).

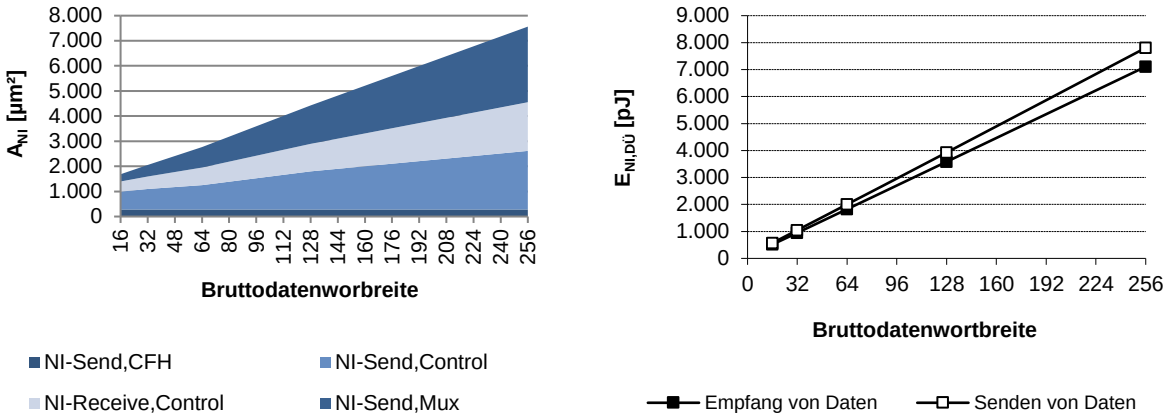


*Abbildung 4.11: Fläche und Energie einer Datenübertragung eines Network-Interfaces für verschiedenen Wortbreiten bei **Wormhole-Routing** (Die Fläche des benötigten Send-Buffers ist im Diagramm nicht dargestellt.)*

Der beim Wormhole-Routing benötigt Speicher ist in Abbildung 4.11 nicht dargestellt. Dieser dominiert, zumindest bei der hier verwendeten standardzellenbasierten Implementierung, die Fläche des Network-Interfaces, so dass die Abhängigkeiten der übrigen Module von der Wortbreite nicht mehr erkennbar sind.

Sowohl die Fläche als auch die für eine Datenübertragung benötigte Energie eines Network-Interfaces sind für die Leitungsvermittlung deutlich geringer als bei dem Wormhole-Routing. Dies wird durch den zusätzlich benötigten Speicher (*Send-Buffer*) verursacht. Die Performance, insbesondere für größere NoC, ist jedoch beim Wormhole-Routing besser (vgl. Kapitel 4.7). Eine Reduktion, sowohl der benötigten Fläche wie auch der benötigten Energie bzw. Verlustleistung, lässt sich durch die Verwendung von physikalisch optimierten Kernkomponenten erreichen. Insbesondere der Speicher im *Send-Buffer*, der beim Wormhole-Routing benötigt wird, stellt eine interessante Komponente für eine Optimierung dar. Dies wird ausführlich in Abschnitt 4.4.4 erläutert.

Der Speicher des *Send-Buffers*, kann neben der Verwendung für das Wormhole-Routing, auch noch für einen mögliche Fehlerbehandlung (Send and Wait mit Sliding Window Enhancement) verwendet werden. Je nach angeschlossener funktionaler Einheit kann auch auf eine Implementierung des Speichers innerhalb des Network-Interfaces verzichtet werden und der Speicher stattdessen mit der funktionalen Einheit geteilt werden (z.B. on-Chip Speicher, Cache eines Prozessorkerns).

4.4.2 Datenfehlerschutz

Im Rahmen dieser Arbeit wurden verschiedene Maßnahmen zur Fehlererkennung und -korrektur, sowie zur Fehlerbehandlung implementiert. Zur Fehlererkennung und -korrektur wurden die in Tabelle 2.1 vorgestellten Codierungen verwendet. Bei dem sendenden Network-Interface wird diese Codierung im Block *Error Correcting or Detecting Code* (*ECC/EDC*) vorgenommen und im empfangenden Network-Interface ebenfalls im Block *Error Detecting or Correcting Code* (*ECC/EDC*) ausgewertet.

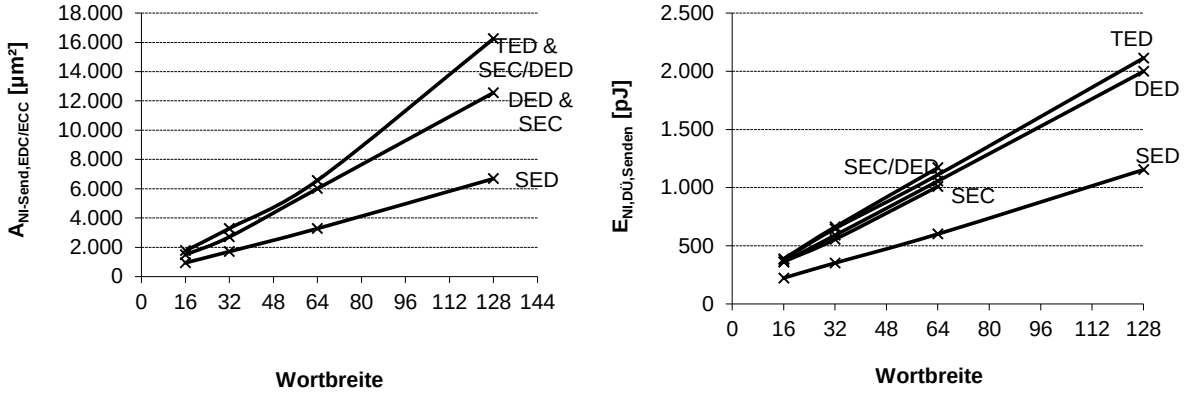
Tabelle 4.4: Abkürzungen zum Datenfehlerschutz

Abkürzung	**Beschreibung**	**Abkürzung**	**Beschreibung**
ECC	Error Correcting Code	SED	Single Error Detection
EDC	Error Detecting Code	DED	Double Error Detection
S&F	Send and Forget	TED	Triple Error Detection
S&W	Send and Wait	SEC	Single Error Correction
		DED	Double Error Correction

Zur Fehlerbehandlung wurden die in Kapitel 2.2.4 vorgestellten Mechanismen Send and Forget (S&F), Send and Wait (S&W) sowie Send and Wait mit Sliding Window Enhancement (S&W SWE) untersucht. Diese werden sowohl im sendenden als auch im empfangenden Network-Interface in einem entsprechenden Block *Error Handling* implementiert.

Der Einfluss der Parameter Fehlererkennung und -korrektur sowie Fehlerbehandlung kann getrennt voneinander untersucht werden. Die Fehlererkennung und -korrektur beeinflusst lediglich den Block *ECC/EDC* im sendenden und empfangenden Teil des Network-Interfaces, der Parameter Fehlerbehandlung entsprechend den Block *Error Handling*.

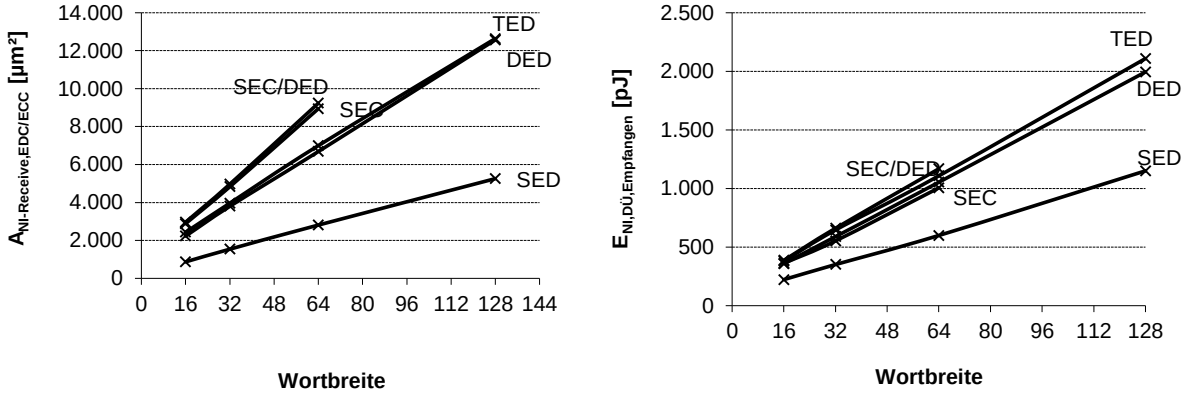
Der Block *ECC/EDC* hängt neben dem verwendeten Algorithmus zur Fehlererkennung auch von der Wortbreite der Daten ab. In Abbildung 4.12 sind die Kosten für den Block *ECC/EDC* im sendenden Teil eines Network-Interfaces und in Abbildung 4.13 des empfangenen Teils aufgetragen.



(a) (b)

*Abbildung 4.12: Fläche (a) und Energie für eine Datenübertragung (b) des Blocks ECC/EDC des **sendenden** Teils des Network-Interfaces für verschiedene Wortbreiten und Fehlerschutzcodierungen (Fehlerbehandlung: Send and Forget)*

Wie erwartet, ist die Fläche und Verlustleistung des Blocks *ECC/EDC* für den SED-Code, einem Paritätscode, beim sendenden Network-Interface am geringsten, wogegen Fläche und Verlustleistung für die übrigen Codes nahezu identisch sind. Der Mehraufwand des Enhanced Hamming Codes (TED, SEC/DED) gegenüber dem Hamming Code (DED, SEC) ist zu vernachlässigen. Sowohl die benötigte Fläche, als auch die Verlustleistung, zeigen eine nahezu lineare Abhängigkeit von der Wortbreite..

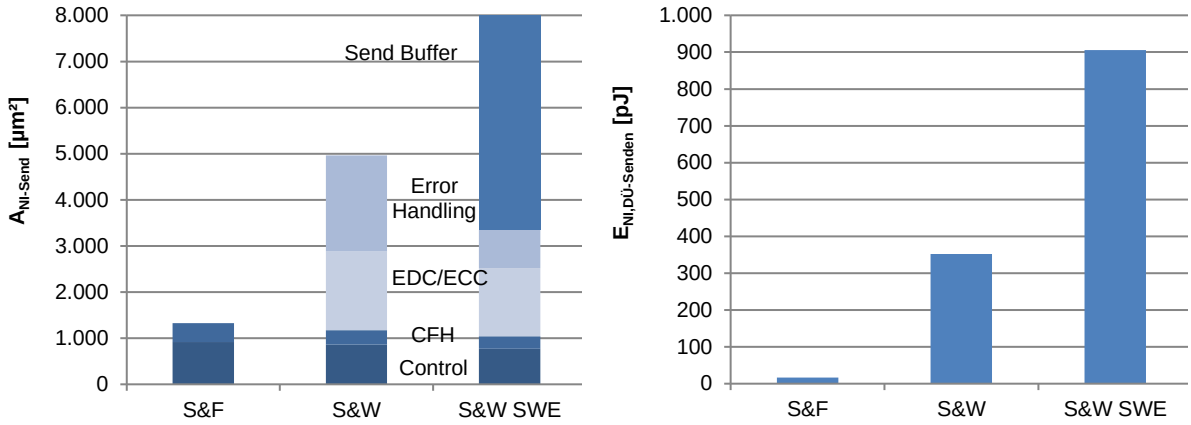


(a) (b)

*Abbildung 4.13: Fläche (a) und Energie (b) für eine Datenübertragung des Blocks ECC/EDC des **empfangenden** Teils des Network-Interfaces für verschiedene Wortbreiten und Fehlerschutzcodierungen*

Bei dem empfangenden Network-Interface zeigt die Verlustleistung und Fläche des Blocks *ECC/EDC* ebenfalls eine lineare Abhängigkeit von der Wortbreite für die verschiedenen Algorithmen. Auch hier verursacht der SED-Code die geringsten Kosten, gefolgt von dem

TED- und DED-Code. Die größten Kosten werden von den SEC- und SEC/DED-Codes verursacht. Dies ist durch die, verglichen mit dem DED- bzw. TED-Codes, hinzugekommene Fehlerkorrektur zu erklären. Wie bei dem Block *ECC/EDC* im sendenden Network-Interface ist auch hier der Mehraufwand des Enhanced Hamming Codes gegenüber dem einfachen Hamming Code gering.



(a) (b)

Abbildung 4.14: Flächenverteilung (a) und Verlustleistung (b) des sendenden Teils des Network-Interfaces für verschiedene Fehlerbehandlungstechniken, Größe des Sliding Windows ist 32 Datenworte

Die Aufteilung der Fläche des Network-Interface für verschiedene Fehlerbehandlungsmechanismen auf die Blöcke des Network-Interfaces ist in Abbildung 4.14 für den sendenden Teil und in Abbildung 4.15 für den empfangenden Teil dargestellt. Dabei wurde ein SED-Code zur Fehlerdetektion verwendet und die Größe des Sliding Windows bei S&W SWE auf 32 festgelegt. Die Fläche des Network-Interfaces wird durch Datenfehlerschutz-Mechanismen deutlich erhöht. Allein die Fehlererkennung (*ECC/EDC*) steigert im sendenden Teil die Fläche um mehr als 180 % gegenüber einer Implementierung ohne Datenfehlerschutz (nur die Blöcke *Control* und *CFH*). Durch das S&W-Protokoll wird neben den Blöcken *Control* und *CFH* auch noch Fläche für die Blöcke *Error Handling* benötigt. Die Fläche dieser zusätzlichen Blöcke hat ebenfalls eine lineare Abhängigkeit von der Wortbreite, die hier jedoch nicht dargestellt ist (vgl. Kapitel 4.4.1). Die Fläche für den Block *Error Handling* ist für S&W SWE geringer als beim einfachen S&W, da hier kein Datenwort zwischengespeichert werden muss. Allerdings wird jetzt ein zusätzlicher Speicher benötigt, der den Flächenbedarf und auch die Verlustleistung dominiert. Die Größe des Sliding Windows bestimmt die Anzahl der Datenworte, die gespeichert werden müssen. Die Fläche, die für den Speicher benötigt wird, ist direkt von der Wortbreite und der Anzahl der zu speichernden Datenworte abhängig. Zudem kann die Fläche und insbesondere die

Verlustleitung für den *Send-Buffer* durch Verwendung spezieller Registerzellen deutlich reduziert werden (vgl. Kapitel 4.4.4).

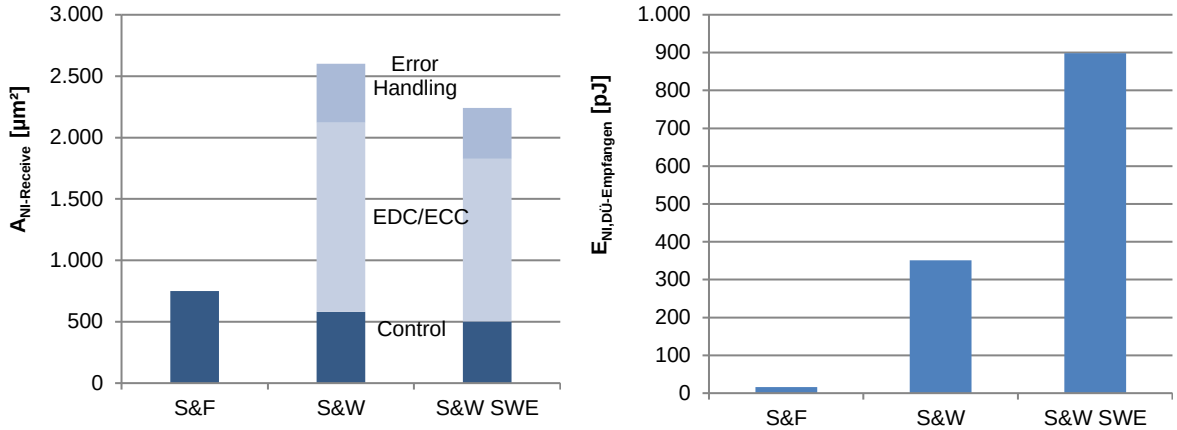


Abbildung 4.15: Flächenverteilung (a) und Verlustleistung (b) des empfangenden Teils des Network-Interfaces für verschiedene Fehlerbehandlungstechniken.

Der große Anstieg der für eine Datenübertragung benötigten Energie, sowohl beim Lesen als auch beim Schreiben von Daten (Abbildung 4.14 und Abbildung 4.15), liegt beim S&W Protokoll an der längeren Übertragungsdauer (siehe auch Abbildung 4.16). Der Speicher, der für das S&W SWE Protokoll im sendenden Network-Interface benötigt wird, dominiert die Verlustleistung bei einer Datenübertragung, die diesen Fehlerbehandlungsmechanismus nutzt.

Der Aufwand im empfangenden Network-Interface ist sowohl relativ als auch absolut gering. Den größten Anteil nimmt der *ECC/EDC* Block ein. Für die Techniken S&W und S&W SWE wird im Empfänger die gleiche Funktionalität benötigt. Daher ist hier kein Unterschied bei Fläche und Verlustleistung. Alle hier verwendeten Blöcke des empfangenden Network-Interfaces habe eine lineare Abhängigkeit von der Wortbreit, was hier ebenfalls nicht abgebildet ist.

Der verwendete Fehlerschutz hat neben der Fläche und Verlustleistung auch einen Einfluss auf die Performance des NoC, bezogen auf Maße wie beispielsweise Durchsatzrate oder Latenz. Im Folgenden wird exemplarisch die durchschnittliche Latenz Δt_{avg} als Maß für die Performance verwendet. Es muss zwischen der Performance einer fehlerfreien Übertragung und der Performance einer fehlerbehafteten Übertragung unterschieden werden. Um den Einfluss auf die Performance eines NoCs zu untersuchen, wurde in einem Experiment eine Datenübertragung zwischen zwei Network-Interfaces über zwei Routing-Switches, die mit einem Link verbunden waren, durchgeführt. Bei dem Link wurde eine Funktionalität implementiert, die es erlaubt, einen zufällig verteilter Ein-Bit-Fehler für eine definierte Fehlerrate zu erzeugen.

Gegenüber einer Datenübertragung über ein NoC ohne Datenfehlerschutz steigt bei einem NoC mit Datenfehlerschutz die Latenz für fehlerbehaftete und fehlerfreie Datenübertragungen um einen festen Wert für die Fehlerschutzcodierung. Bei den hier verwendeten Implementierungen des fehlererkennenden und –korrigierenden Codes setzt sich dieser Wert aus einem Takt für die Codierung im sendenden und einen weiteren Takt für die Dekodierung im empfangenden Network-Interface zusammen.

Der zusätzliche Aufwand für die Fehlerkorrektur ist vom verwendeten Mechanismus abhängig und variiert stark. Durch eine Fehlerkorrektur, je nach Fehlerschutzcodierung ist keine Fehlerkorrektur oder die Korrektur von Ein- oder Zwei-Bit Fehlern möglich, wird keine weitere Performance-Beeinträchtigung verursacht. Dies ist unabhängig davon, ob die Datenübertragung fehlerfrei war oder ob korrigierbare Fehler aufgetreten sind.

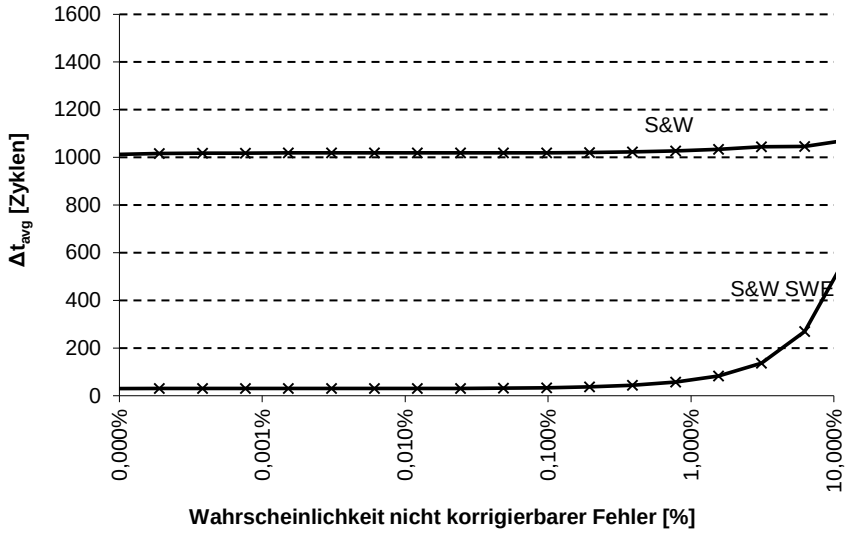


Abbildung 4.16: Einfluss von verschiedenen Fehlerbehandlungstechniken auf die Performance von NoC

Bei der Verwendung des S&W Protokolls wird durch das Warten auf die Bestätigung des korrekten Empfangs des gesendeten Datenwortes die maximal erreichbare Datenrate im fehlerfreien Fall stark reduziert. Die erreichbare Datenrate hängt von der durch das NoC verursachten Latenz für ein Datenwort ab. Diese wird umso größer, je weiter das sendende und empfangende Network-Interface voneinander entfernt sind. Die Latenz eines kompletten Datenpaketes bei fehlerfreier Datenübertragung wird neben der Latenz, die für den Verbindungsaufbau benötigt wird, auch durch die Differenz zwischen angeforderter Datenrate und erreichbarer Datenrate, multipliziert mit der Datenpaket-Größe, bestimmt. Die Performance-Einbußen, die durch das wiederholte Übertragen von Daten entstehen, sind in Abbildung 4.16 abgebildet und haben, selbst für große, und unter realen Bedingungen nicht auftretende Fehlerwahrscheinlichkeiten von 10 % nur einen geringen Einfluss.

Durch die Anwendung des S&W SWE Protokolls tritt, sofern das Sliding Windows groß genug gewählt wurde, keine Beeinträchtigung der Performance auf. Auch im Fall einer fehlerbehafteten Datenübertragung treten erst ab einer Fehlerwahrscheinlichkeit von ca. 1 % merkbare Performance-Einbußen auf (vgl. Abbildung 4.16).

Die beiden Protokolle S&W sowie S&W SWE können mit den verschiedenen fehlererkennenden und -korrigierenden Codes kombiniert werden. Die durchschnittliche Latenz, die in Abbildung 4.16 abgebildet ist, bezieht sich immer auf die Wahrscheinlichkeit eines nicht korrigierbaren Fehlers.

Je nach Anwendung können unterschiedliche Anforderungen an den Datenfehlerschutz gestellt werden. Abhängig von weiteren Parametern wie beispielsweise der verwendeten Technologie oder der Dimensionierung der Verbindungsleitungen können insbesondere bei zukünftigen Technologien Maßnahmen zum Datenfehlerschutz notwendig sein. Wenn eine Datenfehlerbehandlung in einem NoC notwendig sein sollte, stellen die fehlerkorrigierenden Codes, sowohl aus Performance- wie auch aus Kosten-Aspekten eine interessante Lösung des Problems dar. Allerdings können mit den gezeigten Verfahren lediglich Ein- bzw. Zwei-Bit-Fehler korrigiert werden. Treten mehr Fehler in einem Datenwort auf, muss eine Neuübertragung initiiert werden. Das S&W Protokoll hat einen geringen zusätzliche Flächenverbrauch, reduziert jedoch die maximal erreichbare Datenrate erheblich. Das S&W SWE Protokoll verursacht nur geringe Performance-Einbußen. Allerdings ist die zusätzlich benötigte Fläche, die hauptsächlich durch den Speicher benötigt wird, sehr groß. Dieser Speicher kann jedoch auch doppelt genutzt werden, beispielsweise für die Vermittlungstechnik (Wormhole-Routing, Kapitel 4.4.1). Gegebenenfalls kann auf den Speicher auch komplett verzichtet werden, wenn die fehlerhaft übertragenen Daten von der an das sendende Network-Interface angeschlossenen funktionalen Einheit erneut gesendet werden können.

4.4.3 Verbindungsfehlerbehandlung

Bei dem hier verwendeten Loss-Modell wird der Verbindungsaufbau bzw. die Datenübertragung abgebrochen, wenn ein Fehler beim Verbindungsaufbau bzw. bei der Datenübertragung auftritt, beispielsweise bei einem bereits belegten Link. Es wurden verschiedene Möglichkeiten untersucht, um auf dieses Problem zu reagieren: der Verbindungsaufbau / die Datenübertragung wird zu einem späteren Zweitpunkt erneut gestartet. Dabei kann dieser erneute Versuch nach einer festen Anzahl von Taktzyklen erfolgen (FIXED). Daneben kann die Zahl der Wartezyklen zufällig gewählt werden (RANDOM). Auch hier kann die maximale Anzahl der Wartezyklen vorgegeben werden.

Neben der Anzahl der Wartezyklen und der Art, wie dieser Wartezyklen bestimmt werden, kann auch die Priorität der Datenübertragung angepasst werden. Es wurde im Rahmen dieser

Arbeit ein Mechanismus untersucht, bei dem die Priorität einer Datenübertragung mit jedem fehlgeschlagenen Versuch des Verbindungsaufbaus bzw. der Datenübertragung erhöht wird.

Die für das Modul *Connection Failure Handling* benötigte Fläche ist für verschiedene Varianten über der Anzahl der (maximalen) Wartezyklen in Abbildung 4.17, linke Seite, aufgetragen. Die Variante PRIORITY steht für eine dynamische Erhöhung der Priorität bei fester Anzahl von Wartezyklen.

Der Graph der Fläche über den Wartezyklen hat einen logarithmischen Verlauf für alle Varianten. Die geringste Fläche wird für die Variante FIXED benötigt. Bei der Variante PRIORITY erhöht sich die Fläche um einen konstanten Anteil. Diese zusätzliche Fläche wird für die Implementierung der Prioritätsverwaltung benötigt.

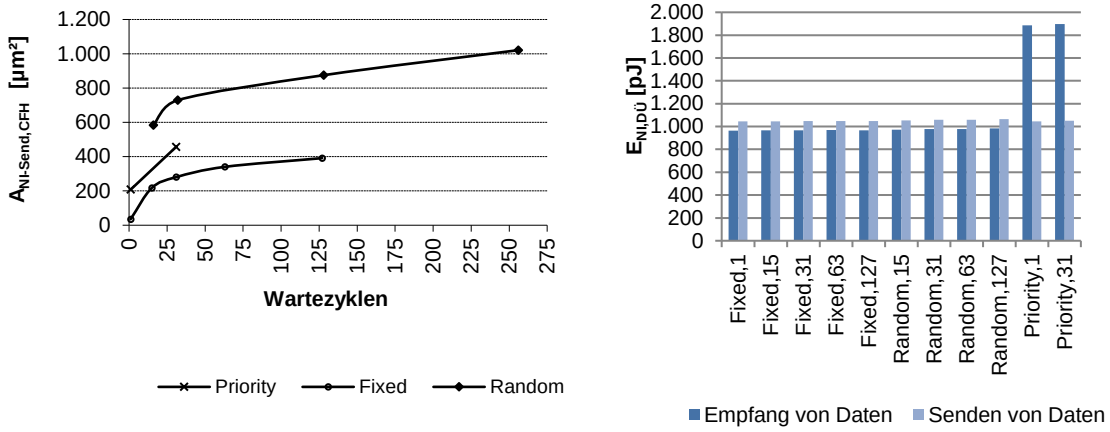
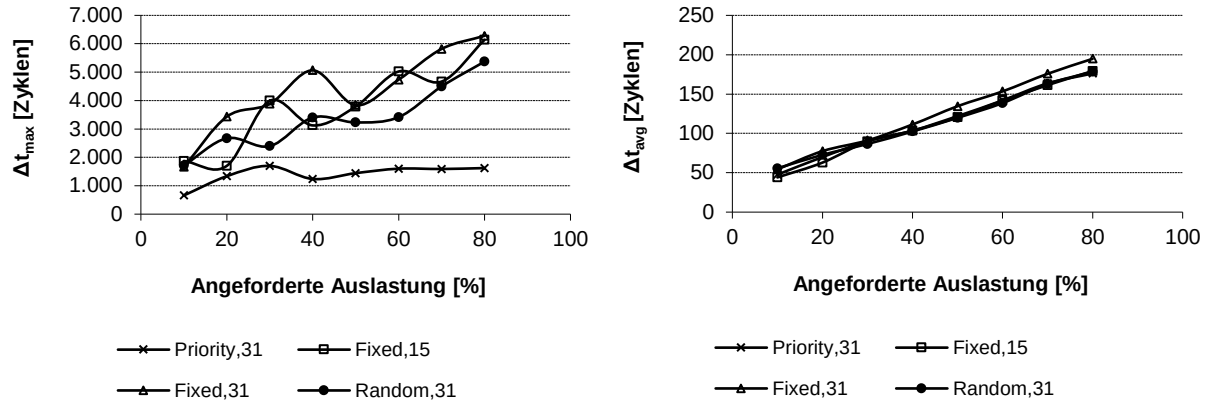


Abbildung 4.17: Fläche des Blocks NI-Send,CFH für verschiedenen Verbindungsfehlerbehandlungen (links) sowie die Energie einer Datenübertragung eines Network-Interfaces mit verschiedenen Verbindungsfehlerbehandlungen beim Senden und Empfangen von Daten (rechts).

Die Verbindungsfehlerbehandlung hat keinen weiteren Einfluss auf die übrigen Module des Network-Interfaces.

Die für eine Datenübertragung benötigte Energie, die beim Senden und Empfangen von Daten benötigt wird, ist auf der rechten Seite von Abbildung 4.17 dargestellt. Die Energie für eine Datenübertragung wird von der Wahl der Verbindungsfehlerbehandlung nur geringfügig beeinflusst, da der Anteil des Blocks *Connection Failure Handling* nur einen kleinen Teil an der Gesamtfläche des Network-Interfaces einnimmt (Abbildung 4.7).



(a) (b)

Abbildung 4.18: Maximal (Δt_{max}) und durchschnittlich (Δt_{avg}) erreichte Latenz für ein NoC mit verschiedenen Verbindungsfehlerbehandlungen

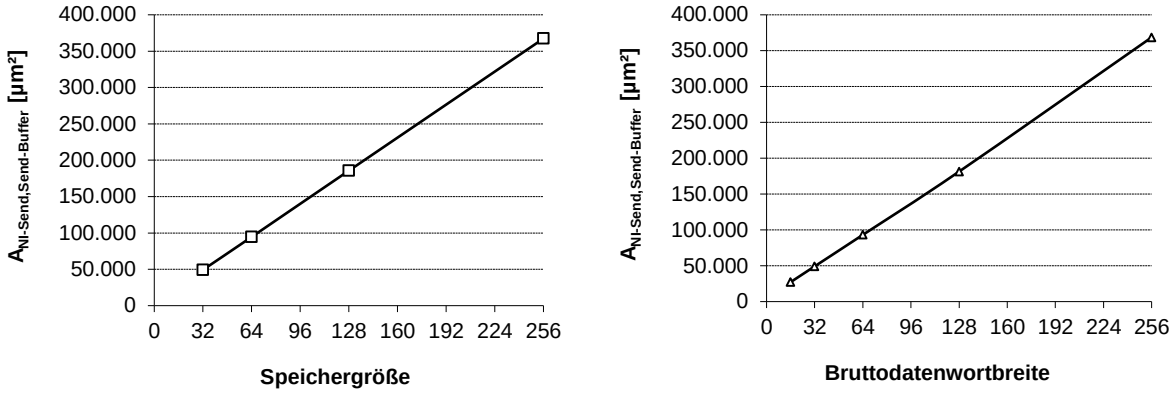
Die Performance von NoC mit unterschiedlichen Verbindungsfehlerbehandlungs-Strategien ist in Abbildung 4.18 dargestellt. Neben der durchschnittlichen Latenz auf der rechten Seite der Abbildung ist auch die maximal während eines Experimentes aufgetretene Latenz für unterschiedliche Auslastungen des NoC aufgetragen. Die durchschnittliche Latenz wird von der Wahl der Verbindungsfehlerbehandlung kaum beeinflusst. Die maximale Latenz ist dagegen, insbesondere für stark ausgelastete NoC, bei der Verwendung der Variante PRIORITY deutlich geringer. Die relativ starken Schwankungen der Messergebnisse, mit Ausnahme von PRIORITY, erklären sich durch die Art des Experimentes. Da hier die maximale Latenz aufgetragen ist, steigt die Wahrscheinlichkeit, dass Verbindungen sich gegenseitig blockieren, an. Bei der durchschnittlichen Latenz mittelt sich dieser Effekt, einzelner stark verzögerter Datenübertragungen, wieder heraus.

Zusammenfassend ist der Einfluss der Verbindungsfehlerbehandlung auf die Fläche des Network-Interfaces vernachlässigbar. Durch die Verwendung der dynamischen Anpassung der Priorität von Verbindungen lässt sich die maximale Latenz senken. Auf die durchschnittliche Latenz hat die dynamische Anpassung der Priorität einen vernachlässigbaren Einfluss. Die Verwendung priorisierter Verbindungen erfordert angepasste Routing-Switches (vgl. Abschnitt 4.5.3) und kann zudem nicht bei der Vermittlungstechnik Wormhole-Routing verwendet werden.

4.4.4 Send-Buffer

Der Block Send-Buffer im sendenden Teil des Network-Interfaces wird zum Speichern von Daten bei der Vermittlungstechnik Wormhole-Routing und bei der Datenfehlerbehandlung Send and Wait mit Sliding Window Enhancement benötigt. Es werden dort Daten gespeichert, die von dem Network-Interface an den angeschlossenen Routing-Switch gesendet wurden,

und bei denen der korrekte Empfang jedoch noch nicht bestätigt wurde. Für den Fall, dass diese Bestätigung ausbleibt, können die Daten so erneut übertragen werden.



(a) (b)

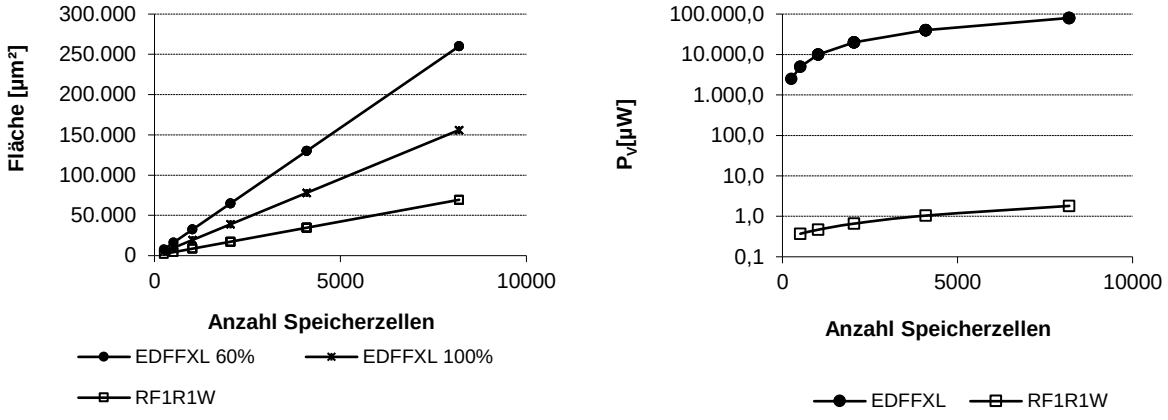
Abbildung 4.19:Fläche des Send-Buffers in Abhängigkeit von der Speichergröße (Anzahl der Datenworte) bei einer Wortbreite von 32 Bit (a) und der Wortbreite (b) bei einer Speichergröße von 32 Datenworten.

Die Fläche des Send-Buffers wird durch die Größe des Speichers (Anzahl der Datenworte) sowie die Wortbreite der zu speichernden Daten bestimmt. Die Kontrolllogik zum Adressieren der einzelnen Speicherzellen ist lediglich von der Größe des Speichers abhängig. In Abbildung 4.19 sind diese Abhängigkeiten für eine reine standardzellenbasierte Implementierung aufgetragen.

Die Fläche des Speichers dominiert die Fläche des Network-Interfaces. Daher wurde in dieser Arbeit untersucht, wie die Fläche und auch die Verlustleistung, weiter optimiert werden kann. Die initiale Implementierung basiert dabei auf der Synthese der funktionalen Beschreibung des Speichers, der mit D-Flipflops synthetisiert wurde. Zunächst wurde versucht, durch Optimierung der Synthese-Parameter eine Kostenreduktion zu erzielen. In einem zweiten Schritt wurde der Speicher nicht mehr automatisch synthetisiert. Der Speicher wurde stattdessen mit Hilfe von zustandsgesteuerten Speicherzellen in VHDL beschrieben und synthetisiert.

Der Flächenbedarf dieser Implementierung kann durch die Verwendung spezieller asynchroner Speicherzellen, die Teil von Standardzellen-Bibliotheken sind, anstelle von D-Flip-Flops sowie durch die Einschränkung der Funktionalität (z.B. Verzicht auf Reset-Funktion) deutlich gesenkt werden. Durch den regulären Aufbau des Speicherfeldes, wird kein zusätzlicher Platz für das Routing benötigt. Eine Steigerung der Flächenauslastung im Rahmen des Place and Route Schrittes auf 100 % bewirkt eine weitere Reduktion der Fläche des Speichers. Dies ist aufgrund des sehr regulären Aufbaus des Speicherfeldes möglich, da hier keine zusätzliche Fläche für das Routing- vorgesehen werden muss.

In Abbildung 4.20 ist der Flächenbedarf des Speichers in Abhängigkeit der Anzahl der benötigten Speicherzellen (Wortbreite multipliziert mit der Speichergröße) dargestellt. Die Steuerlogik ist in dieser Darstellung vernachlässigt worden. Um die Flächenwerte zu berechnen, wurde die Grundfläche der Zelle aus der Standardzellen-Bibliothek mit der Anzahl der benötigten Speicherzellen multipliziert. Anschließend wurde dieser Wert durch die angegebene Flächenauslastung dividiert. Die Grundflächen der einzelnen Zellen wurden dem Datenblatt der Standardzellen-Bibliothek entnommen.



(a) (b)

Abbildung 4.20: Abschätzung der Fläche (a) und der Verlustleistung während einer Datenübertragung (b) für die Speicherzellen (ohne Kontrolllogik) für verschiedene Speicherzellen und Flächenauslastungen (60 % und 100 %) bei einer Bruttodatenwortbreite von 32 Bit.

Die größte Fläche benötigt eine Implementierung des Speichers mit Zellen, die von der Synthese-Software, auf Basis eine funktionale VHDL-Beschreibung des Speichers, aus ausgewählt wurde. Dabei handelt es sich um D-Flipflops mit dem Name EDFFXL. Es wurde eine Flächenauslastung von 60 % angenommen, wie sie für die Synthese der NoC-Komponenten verwendet wird (EDFFXL 60%). Das Speicherfeld ist sehr regulär aufgebaut. Deswegen wird kein zusätzlicher Platz für das Routing benötigt, so dass bei der zweiten Abschätzung die Flächenauslastung auf 100 % gesetzt wurde (EDFFXL 100%). Bei der dritten Abschätzung wurde die vom Synthese-Tool gewählte Speicherzelle durch eine zustandsgesteuerte Speicherzelle (RF1R1W) ersetzt. Diese Abschätzung zeigt ein Optimierungspotential von ca. 70 % bezüglich der Fläche für das Speicherfeld.

Im Folgenden wird der Einfluss dieser Optimierungen auf die durch das Speicherfeld verursachte Verlustleistung diskutiert: Der Send-Buffer wird lediglich in der Phase III der Datenübertragung (Senden von Daten), wenn Wormhole-Routing oder als Fehlerbehandlung das Send and Wait-Protokoll mit Sliding Window Enhancement verwendet wird, benötigt. Dabei wird pro Takt ein Datenwort gelesen und ein Datenwort geschrieben. Die Anzahl der in

einem Takt aktiven Speicherzellen ist somit gleich der Bruttodatenwortbreite, unabhängig von der Größe des Speicherfeldes.

Die Verlustleistung des Send-Buffers lässt sich in eine Verlustleistung im Ruhezustand und in einen dynamischen Anteil, differenzieren. Der erste Anteil wird hier primär durch die Schaltaktivität des Taktsignals verursacht. Der dynamische Anteil wird durch das Lesen und Schreiben von Daten verursacht.

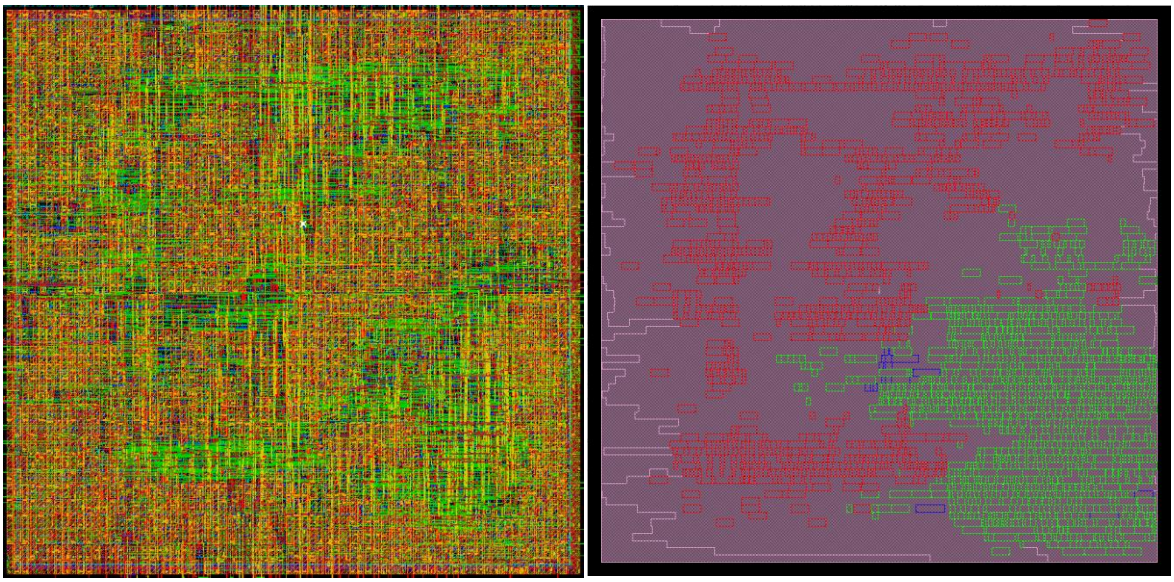
Bei dem, aus einer funktionalen Beschreibung des Speichers synthetisierten, Send-Buffer wurden für die Speicherzellen D-Flipflops verwendet. Es kann gezeigt werden, dass der dynamische Anteil an der Verlustleistung des Speichers unter einem Prozent liegt. Dies wird durch die hohe Verlustleistung im Ruhezustand verursacht. In dem Speicherfeld werden flankengesteuerte Speicherzellen verwendet, die an das globale Taktnetzwerk angeschlossen sind, die eine hohe, von der Anzahl der Speicherelemente abhängige Verlustleistung erzeugen. Die in dem Speicherfeld verursachte Verlustleistung dominiert die im gesamten Network-Interface umgesetzte Verlustleistung.

Bei der im Rahmen der Optimierung verwendeten Speicherzelle handelt es sich um eine zustandsgesteuerte Speicherzelle. Diese ist nicht an das Taktsignal angeschlossen und die Verlustleistung durch Leckströme bei der hier verwendeten Standardzellen-Bibliothek ist zu vernachlässigen. Daher ist die Verlustleistung im Ruhezustand gleich null.

In Abbildung 4.20 wurde die Verlustleistung, die durch das Speicherfeld verursacht wird, aufgetragen. Diese wurde auf Basis der Berechnungsvorschriften aus dem Datenblatt der Standardzellen-Bibliothek abgeschätzt Durch die optimierte Implementierung konnte die durch das Speicherfeld verursachte Verlustleistung um ca. fünf Größenordnungen reduziert werden. Für diese Optimierung muss jedoch die funktionale, technologieunabhängige VHDL-Beschreibung des Blocks *Send-Buffer* durch eine Beschreibung ersetzt werden, die für die Standardzellen-Bibliothek optimiert ist.

4.5 Routing-Switch

Für die in Tabelle 4.1 angegebenen NoC-Parameter (Nettodatenwortbreite: 32 Bit, Vermittlungstechnik: Wormhole-Routing, Routing-Algorithmus: XY-Routing) wurde eine exemplarische Implementierung eines Routing-Switches mit 5 Ein- und Ausgangsports implementiert. Das Ergebnis dieser Implementierung ist im Folgenden dargestellt. In Abbildung 4.21 ist auf der linken Seite das Layout des Routing-Switches dargestellt. Auf der rechten Seite sind die einzelnen Blöcke des Routing-Switches farbig markiert.



(a) (b)

Abbildung 4.21: Layout (a) eines Routing-Switches. Verteilung der Fläche auf den Router (Blau), den Arbiter (Grün), den Switch (Rot) und die Register (restliche Fläche) (b).

Wie im Network-Interface wird auch im Routing-Switch die Fläche durch die Blöcke dominiert, welche die Daten übertragen (Switch und Register). Die Blöcke, die eine steuernde Funktion haben (Arbiter und Router) nehmen dagegen einen kleineren Teil der Fläche ein. Dies ist in Abbildung 4.22 quantitativ dargestellt.

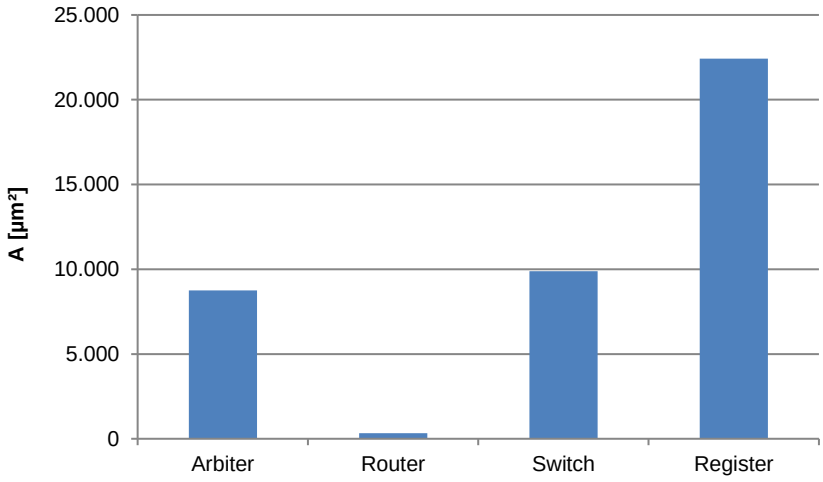


Abbildung 4.22: Fläche der einzelnen Module eines Routing-Switch

Die von einem Routing-Switch verbrauchte Verlustleistung ist nicht konstant, sondern hängt von dem Zustand des Routing-Switches ab. Im Zustand *Idle* werden keine Daten über den Routing-Switch übertragen und es sind keine Verbindungen aktiv oder werden auf- oder

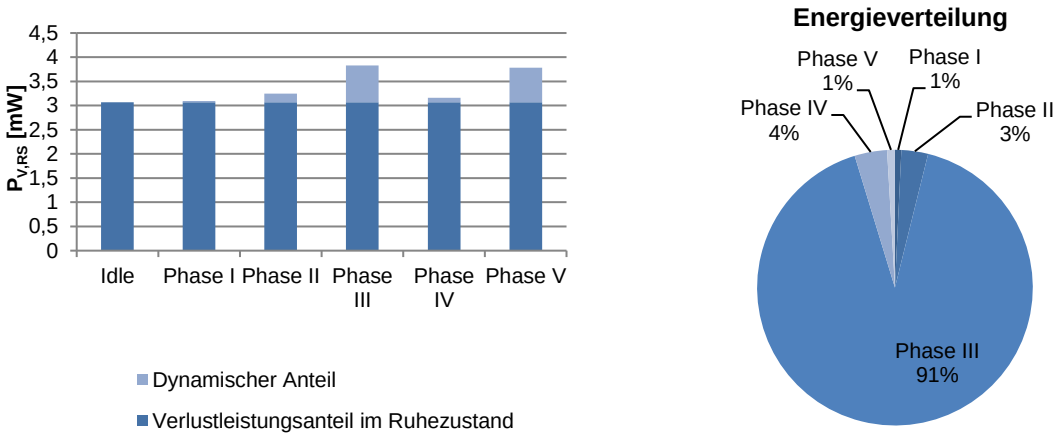
abgebaut. Im Laufe einer Datenübertragung werden die in Tabelle 4.5 aufgeführten Phasen durchlaufen.

Tabelle 4.5: Phasen einer Datenübertragung in einem Routing-Switch

Übertragungs-Phase	**Beschreibung**
Idle	Kein Datenverkehr, lediglich Aktivität auf der Leitung des Taktsignals
I	Routing und Arbitrierung
II	Warten auf den Beginn der Datenübertragung
III	Datenübertragung
IV	Warten auf den Verbindungsabbau bzw. das Ende des Datenpakets
V	Verbindungsabbau

Die Verlustleistung, die während der einzelnen Phasen bei einer aktiven Verbindung verursacht wird, ist in Abbildung 4.23 (a) aufgetragen. Sie ist in die Verlustleistung im Ruhezustand und einen dynamischen Teil aufgegliedert. Die Verlustleistung im Ruhezustand wird durch das Taktnetzwerk sowie Leakage verursacht. Der dynamische Anteil resultiert aus der Schaltaktivität, die für die Übertragung der Daten benötigt wird. Da in einem Routing-Switch mehrere aktive Verbindungen parallel Daten übertragen können, muss bei mehreren aktiven Verbindungen zum Verlustleistungsanteil im Ruhezustand für jede aktive Verbindung der dynamische Anteil addiert werden.

Der Großteil an der Verlustleistung eines Routing-Switches bei einer aktiven Verbindung wird bereits im Ruhezustand verbraucht (ca. 75 %). Bei fünf aktiven Verbindungen, über die Daten übertragen werden, sinkt dieser Anteil auf ca. 40%.



(a) (b)

Abbildung 4.23: Verlustleistung (a) und Energieverteilung (b) einer Datenübertragung eines Routing-Switches für die einzelnen Phasen einer Datenübertragung

Die Verteilung der Energie der zuvor spezifizierten Datenübertragung auf die einzelnen Phasen der Datenübertragung ist in Abbildung 4.23 (b) dargestellt. Auch hier werden über

90 % der Energie während der eigentlichen Datenübertragung umgesetzt. Dieser Anteil ist linear von der Anzahl der zu übertragenden Worte abhängig, und wird durch größere Datenpakete noch gesteigert.

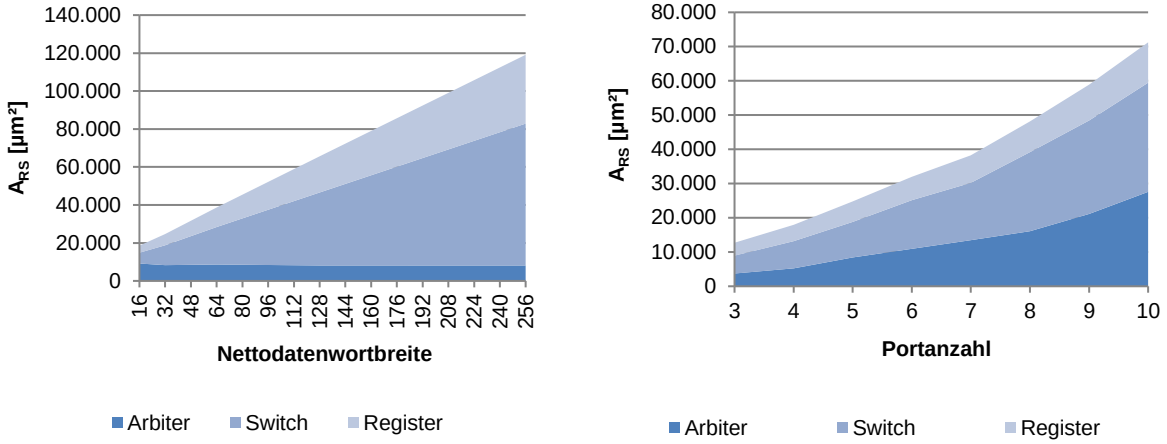
4.5.1 Vermittlungstechnik

Die untersuchten Vermittlungstechniken Wormhole-Routing und Leitungsvermittlung beeinflussen lediglich den Block *Register*. Die übrigen Blöcke sind für beide Vermittlungstechniken nahezu identisch.

Zunächst werden der Einfluss der Port-Anzahl und der Wortbreite auf die Fläche des Routing-Switches und auf die vom Routing-Switch für eine Datenübertragung benötigte Energie untersucht. In Abbildung 4.24 ist die Flächenverteilung der einzelnen Blöcke eines Routing-Switches für verschiedene Wortbreiten (a) und Portanzahlen (b) für Leitungsvermittlung dargestellt. Die Fläche für den Router ist in dem Diagramm nicht darstellbar, da sie im Verhältnis zu den übrigen Modulen sehr klein ist (vgl. Abbildung 4.22). Sowohl die Fläche des Arbiters wie auch des Routers sind unabhängig von der Wortbreite. Die Fläche für den Switch wie auch die Fläche der Register steigt mit wachsender Wortbreite linear an, wobei der Zuwachs bei der Fläche des Switches stärker ist, als bei der Fläche, die von den Registern benötigt wird.

Die Portanzahl hat ebenfalls einen Einfluss auf die Fläche der einzelnen Blöcke: die Fläche der *Register* und des *Switches* steigen linear, die des *Arbiters* exponentiell an. Die Flächenänderung des *Routers* ist vernachlässigbar und ist deshalb in Abbildung 4.24 nicht dargestellt.

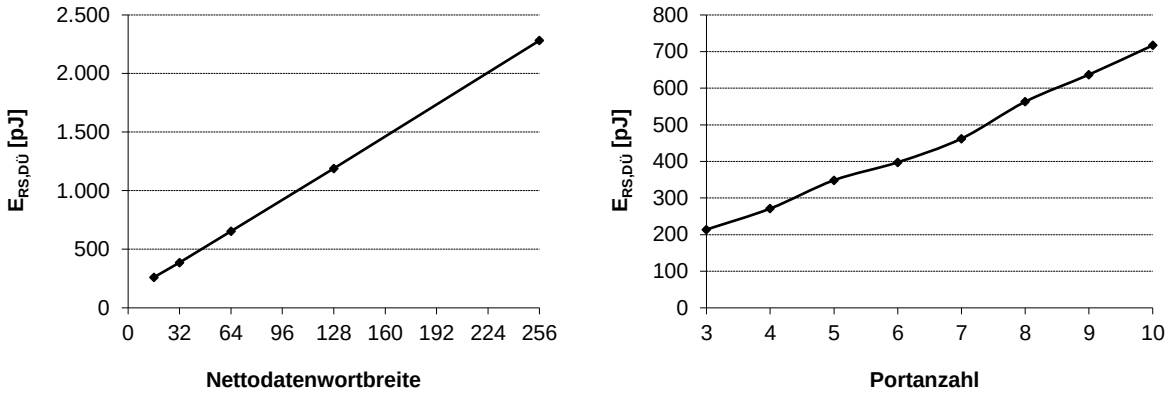
Insbesondere für Wortbreiten größer als 32 Bit wird die Fläche des Routing-Switches durch die Fläche des Switches und der Register dominiert. Je größer die Wortbreite ist, umso größer wird der Anteil der Fläche dieser Komponenten an der Fläche des Routing-Switches. Für steigende Portanzahlen steigt auch die Fläche des Routing-Switches.



(a) (b)

*Abbildung 4.24: Flächenverteilung eines Routing-Switches bei **Leitungsvermittlung** für unterschiedliche Datenwortbreiten (a) und Port-Anzahlen (b)*

Die für eine Datenübertragung benötigte Energie eines Routing-Switches hängt ebenfalls von der Wortbreite und der Portanzahl ab. Auch dieser Zusammenhang ist proportional, wie in Abbildung 4.25 zu sehen ist.



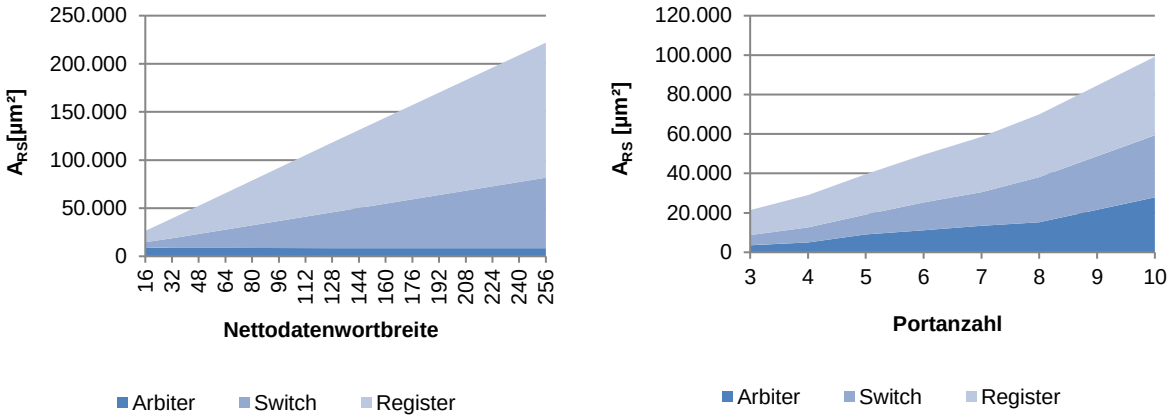
(a) (b)

*Abbildung 4.25: Energie einer Datenübertragung eines Routing-Switches (eine aktive Verbindung) bei **Leitungsvermittlung** für verschiedene Nettodatenwortbreiten (a) und Portanzahlen (b)*

Bei Wormhole-Routing werden nach dem Header direkt die Datenworte über das NoC übertragen (vgl. Abbildung 4.5). Daher müssen in jedem Routing-Switch die Daten in jedem Eingangsport um die Zeitperiode verzögert werde, die für Routing und Arbitrierung benötigt werden. Dies erfolgt mit Hilfe einer Registerkette im Block Register.

Die Flächen der Blöcke Register, Arbiter und Router ändern sich bei der Vermittlungstechnik Wormhole-Routing nicht. Lediglich die Fläche für die Register ist deutlich gestiegen und

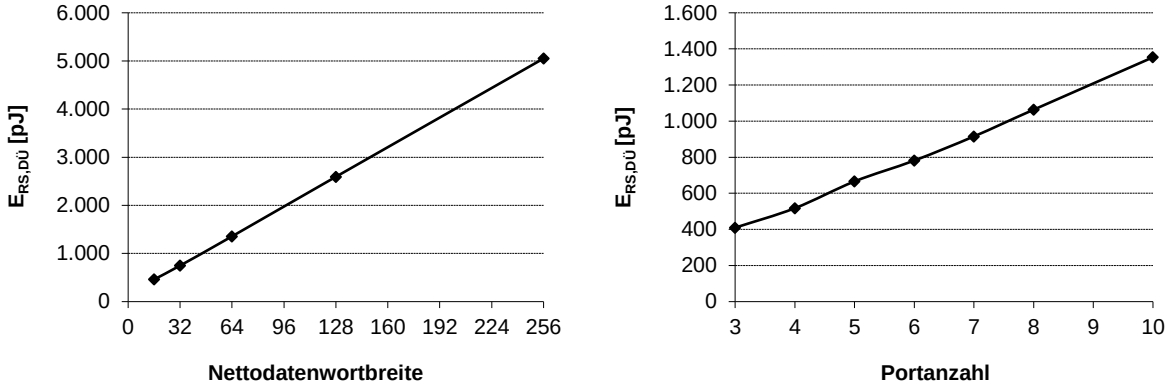
wächst auch mit steigender Wortbreite stärker als bei einem Routing-Switch mit Leitungsvermittlung (Abbildung 4.26). Auch über der Portanzahl wächst die Fläche der Register stärker als bei Leitungsvermittlung.



(a) (b)

*Abbildung 4.26: Flächenverteilung eines Routing-Switches bei **Wormhole-Routing** für unterschiedliche Nettodatenwortbreiten (a) und Portanzahlen (b)*

Durch die größere Fläche und die zusätzlichen Register, die jedes Datenwort passieren muss, steigt die für eine Datenübertragung benötigte Energie gegenüber Leitungsvermittlung stark an. Die Energie besitzt eine proportionale Abhängigkeit sowohl von der Portanzahl als auch von der Wortbreite.



(a) (b)

*Abbildung 4.27: Energie einer Datenübertragung eines Routing-Switches (eine aktive Verbindung) bei **Wormhole-Routing** für verschiedene Nettodatenwortbreiten (a) und Port-Anzahlen (b)*

Die durch den Routing-Switch und die Register verursachten Kosten können durch die Verwendung von physikalisch optimierten Kernkomponenten, insbesondere bei der

Vermittlungstechnik Wormhole-Routing, reduziert werden. Dies ist in Kapitel 4.5.4 ausführlich beschrieben.

4.5.2 Routing-Algorithmus

Im Folgenden sollen einige Routing-Algorithmen für NoC im Detail vorgestellt werden, bei denen nur noch Unicast-Übertragungen betrachtet werden. Eine der Hauptforderungen, die an NoC gestellt werden, ist die Skalierbarkeit. Aus diesem Grund sind zentralisierte Routing-Strategien in NoC wenig verbreitet [97]. Da die Implementierung keinen Einfluss auf den eigentlichen Algorithmus hat, wird sie bei dieser funktionalen Betrachtung der Routing-Algorithmen vernachlässigt. Bei den in Kapitel 2.2.5 vorgestellten Vermittlungstechniken wird eine Strategie der Deadlock-Prävention betrieben. Daher muss bei der Wahl eines geeigneten Routing-Algorithmus keine Rücksicht auf Deadlocks genommen werden, solange immer auf direktem Weg zum Ziel übertragen wird [37]. Die Routing-Algorithmen zur Vermeidung von Deadlocks werden nicht weiter betrachtet. Sie schränken meistens die verfügbaren Ausgangsports eines Routing-Switches ein, wie zum Beispiel beim Turn-Model [98]. Dies führt zu einer schlechteren Verteilung des Datenverkehrs über das Netzwerk und resultiert in einer verminderten Leistung des Netzwerkes. Daraus ergibt sich eine vereinfachte Einteilung von Routing-Algorithmen für NoC wie sie in Abbildung 4.28 dargestellt ist.

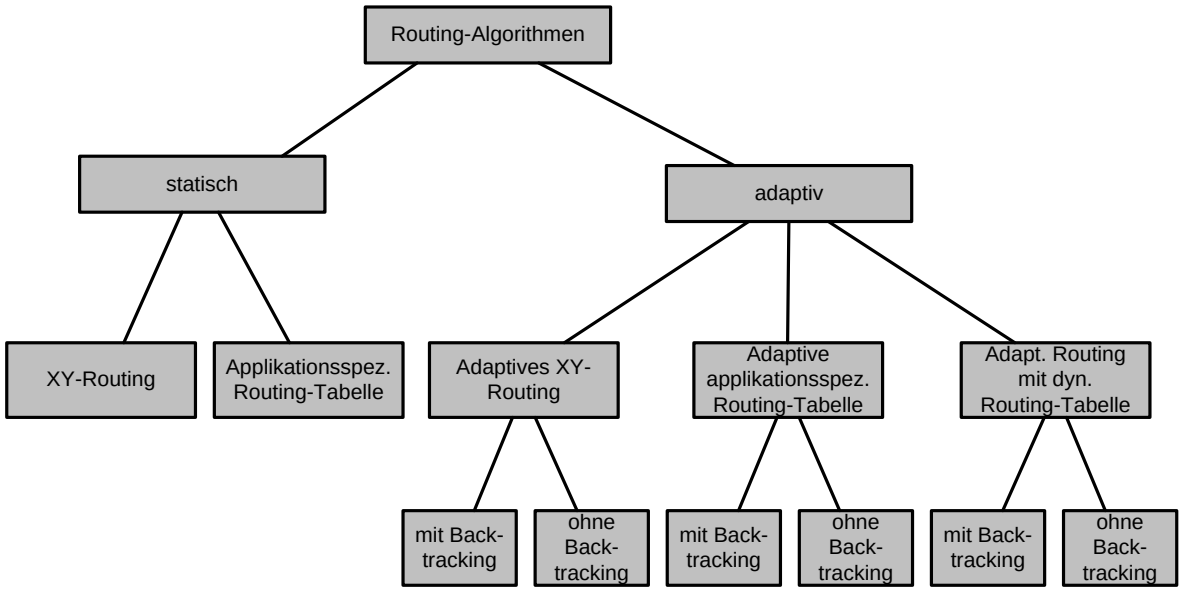


Abbildung 4.28: Übersicht über Routing-Algorithmen in NoC

Bei den präsentierten Algorithmen wird von einer Abbildung der NoC-Komponenten auf ein 2D-Grid ausgegangen. Alle in Kapitel 2.2.8 vorgestellten Topologien lassen sich auf eine solches Grid übertragen. In Abbildung 4.29 ist ein NoC mit Mesh-Topologie (Kapitel 2.2.8) auf ein 2D-Grid abgebildet. Die Adresse der jeweiligen NoC-Komponente ist dabei in der linken oberen Ecke jedes Grid-Elements als X- und Y-Position dargestellt. Grundsätzlich

lassen sich die Algorithmen aber auch auf Darstellungen mit mehr als zwei Dimensionen übertragen.

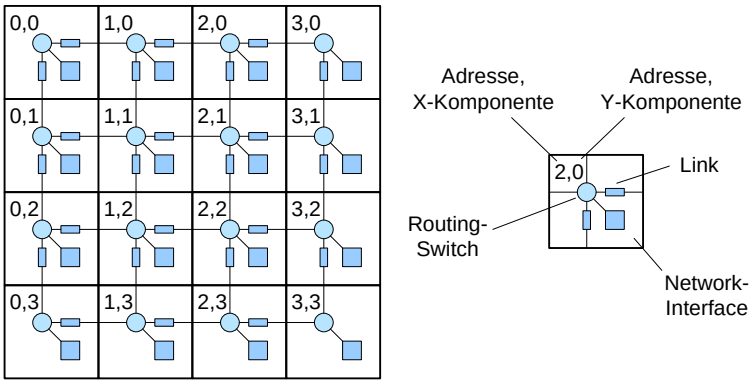


Abbildung 4.29: Abbildung eines NoC mit Mesh-Topologie auf ein 2D-Grid.

XY-Routing (XY): Das XY-Routing ist ein klassischer dimension-order Routing-Algorithmus. Mit Hilfe dieses Algorithmus werden die Daten zunächst entlang der ersten Dimension, hier der X-Dimension, übertragen, bis die X-Adresse des Ziels erreicht wurde. Anschließend werden sie entlang der Y-Dimension bis zur Zieladresse transportiert. Wenn der vom Routing-Algorithmus bestimmte Ausgangsport bereits belegt ist, wird die Datenübertragung abgebrochen. Eine sehr ähnliche Variante ist das YX-Routing, bei dem die Daten zunächst in Y- und anschließend in X-Richtung übertragen werden.

Applikationsspezifische Routing-Tabelle: Bei diesem Routing-Algorithmus wird die Routing-Entscheidung in jedem Routing-Switch individuell so angepasst, dass sie für ein gegebenes SoC und eine gegebene Anwendung optimal ist. Je nach Wahl der Routing-Tabelle lassen sich Netzwerk-Ressourcen, die beispielsweise beim XY-Routing sehr stark genutzt werden, entlasten. Bei der hier gewählten Implementierung wird die Routing-Tabelle während des NoC-Entwurfs festgelegt.

Adaptives XY-Routing (A-XY): Wie beim deterministischen XY-Routing wird auch zunächst in X-Richtung und anschließend in Y-Richtung übertragen. Sollte ein Routing-Ziel in X-Richtung belegt sein, wird überprüft, ob eine Übertragung in Y-Richtung möglich ist, d.h. das alternative Routing-Ziel führt ebenfalls über einen minimalen Weg zum Ziel und ist frei.

Adaptive applikationsspezifische Routing-Tabelle: Auch bei Routing mit applikationsspezifischen Routing-Tabellen kann ein adaptives Routing angewendet werden. Dazu muss lediglich eine zweite Routing-Tabelle mit alternativen Routing-Zielen / Ausgangs-Ports angelegt werden. Diese wird dann bei einer Kollision verwendet.

Adaptives Routing mit dynamischer Routing-Tabelle (A-d-RT): Im Gegensatz zum adaptiven XY-Routing hängt die Wahl des Routing-Ziels innerhalb eines Routing-Switches

auch von dem letzten erfolgreichen Verbindungsaufbau des Routing-Switches ab. Das bedeutet, dass der Routing-Switch lernt, ob es in der Vergangenheit besser war zunächst in X-Richtung (XY-Routing) oder zunächst in Y-Richtung (YX-Routing) eine Verbindung aufzubauen. Dazu wird in einer Tabelle gespeichert, ob zuerst in XY- oder in YX-Routing verwendet werden soll. Die Tabelle besteht aus Einträgen für jede Zieladresse, die einen entsprechenden Zustand speichern. Die Zustandsübergänge dieser Tabelleneinträge sind in Abbildung 4.30 dargestellt. Bei einem erfolgreichen Routing mit XY-Routing (Ereignis *XY-Routing erfolgreich*) wird beispielsweise vom Zustand X1 zu Zustand X2 gewechselt, bei einem erfolgreichen Routing in YX-Richtung (Ereignis *YX-Routing erfolgreich*) zu Zustand Y1. Wenn der Router in den Zuständen X1 und X2 ist, wird zunächst XY- und bei einem Fehler anschließend YX-Routing verwendet.

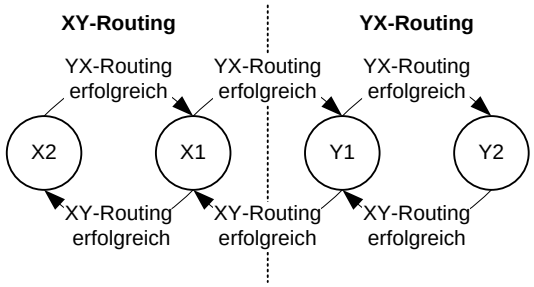


Abbildung 4.30: Zustandsübergänge der dynamischen Routing-Tabelle

Routing mit Backtracking: Die drei vorgestellten adaptiven Routing-Algorithmen lassen sich, wie bisher beschrieben, progressiv realisieren (vgl. Kapitel 2.2.7). Daneben besteht die Möglichkeit, bei der in Kapitel 2.2.5 vorgestellten Leitungsvermittlung eine Implementierung mit Backtracking zu verwenden. Dabei wird bei einer auftretenden Kollision der Verbindungsaufbau nicht abgebrochen, sondern versucht, einen alternativen Weg, ausgehend von dem letzten besuchten Routing-Switch, zu finden. In den folgenden Diagrammen sind Routing-Algorithmen, die Backtracking unterstützen, durch die Erweiterung BT gekennzeichnet (XY-Routing mit Backtracking: XY-BT).

Um bei allen vorgestellten Routing-Algorithmen das Routing-Ziel zu bestimmen, kommen zum einen Zustandsmaschinen in Frage, die das Routing-Ziel berechnen. Eine Alternative ist die Verwendung von Routing-Tabellen. Die Wahl der Implementierungsvariante hat keinen Einfluss auf die Performance, sondern lediglich einen Einfluss auf die durch den Routing-Switch verursachten Kosten. Routing-Algorithmen, die auf einer Routing-Tabelle beruhen, werden durch die Endung RT markiert (XY-Routing mit Routing-Tabelle: XY-RT).

Im Folgenden werden die Routing-Algorithmen XY-Routing (XY, XY-RT) adaptives XY-Routing (A-XY, A-XY-RT), XY-Routing mit Backtracking (XY-BT) sowie adaptives Routing mit dynamischer Routing-Tabelle (A-d-RT) bezüglich der Kosten, ausgedrückt in Fläche und Energie einer Datenübertragung, und Performance untersucht. Routing-

Algorithmen, die auf applikationsspezifischen Routing-Tabellen beruhen, werden hier nicht betrachtet, da bei zufälligem Datenverkehr keine Optimierung der Routing-Tabellen möglich ist. Die Kosten, die durch einen solchen Routing-Algorithmus verursacht werden, entsprechen jedoch denen des XY-Routings bzw. des adaptiven XY-Routings, das mit Hilfe von Routing-Tabellen implementiert wird.

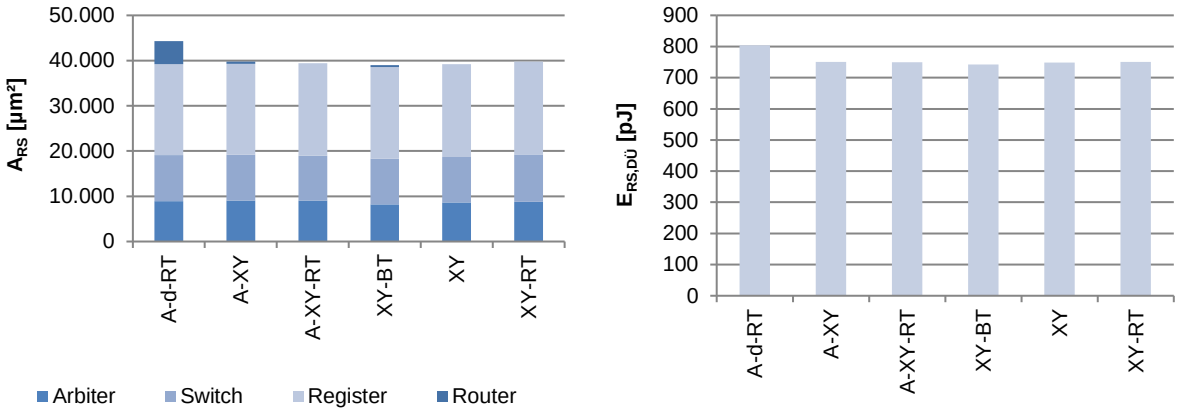


Abbildung 4.31 Flächen des Routers und des Arbiters (links) und Verlustleistung eines Routing-Switches für unterschiedliche Routing-Algorithmen

In Abbildung 4.31 links ist die Verteilung der Fläche in einem Routing-Switch für die beschriebenen Routing-Algorithmen dargestellt. Der Einfluss des Routing-Algorithmus auf die Fläche ist, abgesehen von dem Routing-Algorithmus, der auf einer dynamischen Routing-Tabelle beruht, zu vernachlässigen. Die auftretenden Schwankungen lassen sich durch Zufallsprozesse während der Implementierung im Synthese- und Place and Route-Tool erklären. Lediglich bei der Verwendung einer dynamischen Routing-Tabelle ist eine signifikante Steigerung der Fläche zu erkennen.

Der verwendete Routing-Algorithmus hat lediglich einen geringen Einfluss auf die Energie, die in einem Routing-Switch für eine Datenübertragung umgesetzt wird. Lediglich bei der Verwendung einer dynamischen Routing-Tabelle steigt der Energiebedarf um ca. 6 % gegenüber den restlichen Algorithmen an. Dies liegt an den zusätzlich benötigten Registern, die an das Taktnetzwerk angeschlossen sind und auch während der eigentlichen Datenübertragung Energie verbrauchen.

In Abbildung 4.32 ist die Performance in Form der durchschnittlichen Latenz von NoC mit 25 Teilnehmern und verschiedenen Routing-Algorithmen dargestellt. Das XY-Routing hat die schlechteste Performance, gefolgt von dem adaptiven XY-Routing und dem adaptiven XY-Routing mit dynamischer Routing-Tabelle. Der letztgenannte Algorithmus kann in diesem Test-Szenario nicht besser abschneiden, da bei zufälligem Datenverkehr keine Korrelation unter den Datenübertragungen auftreten, die dieser Algorithmus ausnutzen könnte. Die beste Performance zeigt das adaptive XY-Routing mit Backtracking.

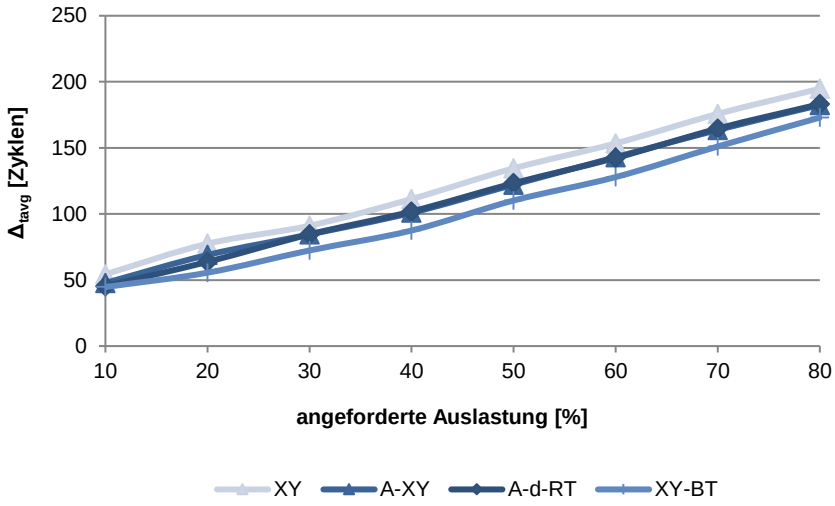


Abbildung 4.32: Performance unterschiedlicher Routing-Algorithmen bei einem NoC mit 25 Teilnehmern, Mesh-Topologie und Leitungsvermittlung

Weitere Untersuchungen, die im Rahmen dieser Arbeit durchgeführt wurden, haben gezeigt, dass die Performance-Unterschiede zwischen den einzelnen Routing-Algorithmen für größere NoC weiter steigen. Auch hat die Art des Datenverkehrs einen Einfluss auf die Performance des NoC. Bei der Übertragung größerer Datenblöcke werden die Performance-Unterschiede zwischen NoC mit unterschiedlichen Routing-Algorithmen ebenfalls größer.

Für ein NoC ist bei zufälligem Datenverkehr nach diesen Ergebnissen die Verwendung von dynamischem XY-Routing mit Backtracking am effizientesten, da hier ein Performance-Gewinn bei gleichen Kosten gegenüber NoC mit anderen Routing-Algorithmen erzielt werden kann. Lediglich bei der Vermittlungstechnik Wormhole-Routing muss ein alternativer Routing-Algorithmus gewählt werden, beispielsweise dynamisches XY-Routing.

Bei Datenverkehr, der nicht zufällig ist, kann die Verwendung von applikationsspezifischen Routing-Tabellen oder adaptiver Routing-Tabellen zu Performance-Steigerungen führen. Diese kann jedoch nicht pauschal beurteilt werden, sondern kann nur experimentell, beispielsweise mit dem FPGA-basierten Emulator, untersucht werden.

4.5.3 Priorisierung von Verbindungen

Die Unterstützung von Verbindungen mit Priorität wird unter anderem für die in Kapitel 4.4.3 vorgestellte Verbindungsfehlerbehandlung oder für die Realisierung von QoS-Diensten benötigt. In der hier vorgestellten Implementierung werden diese Verbindungen allerdings nur bei Leitungsvermittlung unterstützt.

In Abbildung 4.33 ist links die benötigte Fläche für eine Routing-Switch-Implementierung mit und ohne Unterstützung von priorisierten Verbindungen dargestellt. Die Prioritätsunterstützung zeigt sich in einer Vergrößerung der Fläche des Arbiters. Die übrigen

Blöcke bleiben unbeeinflusst. Wie auch in den vorrangegangenen Untersuchungen ist die Fläche des Routers klein, verglichen mit den übrigen Komponenten. Auf der rechten Seite der Abbildung ist der entsprechende Zuwachs bei der für eine Datenübertragung im Routing-Switch umgesetzten Energie aufgetragen.

Der Einfluss auf die Performance für den Einsatz von priorisierten Verbindungen wird in Kapitel 4.4.3 diskutiert.

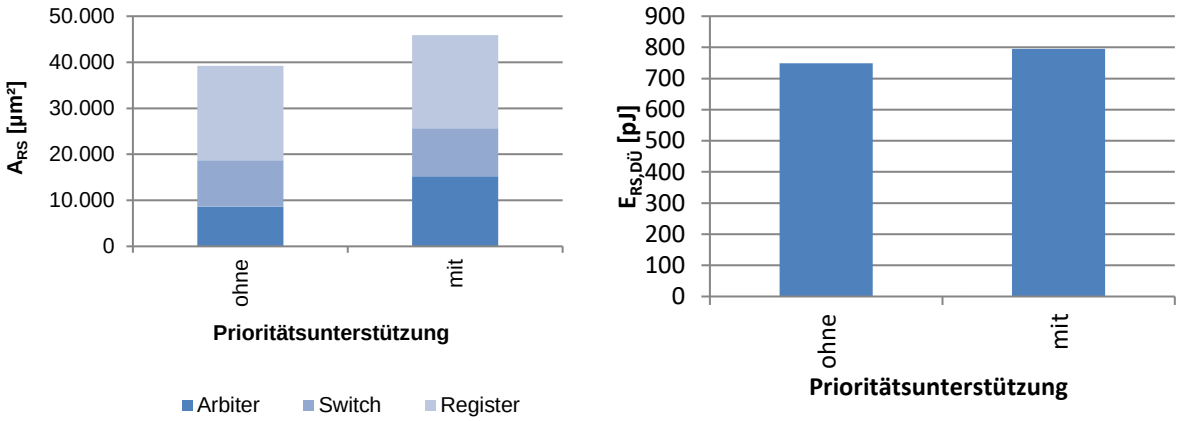


Abbildung 4.33: Vergleich der Fläche und Verlustleistung von Routing-Switches mit und ohne Prioritätsunterstützung

4.5.4 Implementierung von Switch und Registern als physikalisch optimierte Kernkomponenten

Der Routing-Switch lässt sich in zwei Teile gliedern: den Datenpfad, der aus den Registern und dem Switch besteht und den Datenkanal verteilt, sowie der Steuerlogik, die aus Arbiter und Router bestehen. Diese Aufteilung ist in Abbildung 4.34 dargestellt. Der Datenpfad zeichnet sich durch einen sehr regulären Aufbau aus, der sich für eine physikalisch optimierte Implementierung eignet. Für diese physikalisch optimierte Implementierung müssen nur wenige Zellen entworfen werden, aus denen der Switch im Anschluss aufgebaut werden kann.

Im Gegensatz dazu ist die Steuerlogik irregulär aufgebaut. Eine Implementierung mit physikalisch optimierten Komponenten wäre daher aufwändig. Daher wird dieser Teil mit Standardzellen implementiert.

Der Datenpfad besteht aus (optionalen) Eingangsregistern, je nach Vermittlungstechnik weiteren Registern für Wormhole-Routing, dem Switch und Ausgangsregistern mit Ausgangstreibern. Die Register sind für jeden Port identisch aufgebaut.

Der Switch besteht aus einer Kontrolllogik, die aus einem Steuerwort, welches von dem Arbiter erzeugt wird, die Signale zur Ansteuerung der einzelnen Ein-Bit-Switches erzeugt (Steuerleitungen), einer Treiberstufe, in der die Signale auf den Steuerleitungen verstärkt

werden, und *n* Ein-Bit-Switches. Diese verbinden die Eingangsports mit den Ausgangsports. Der Aufbau des Switches ist in Abbildung 4.35 abgebildet. Die Kontrolllogik besteht aus NAND-Gattern mit teilweise invertierten Eingängen.

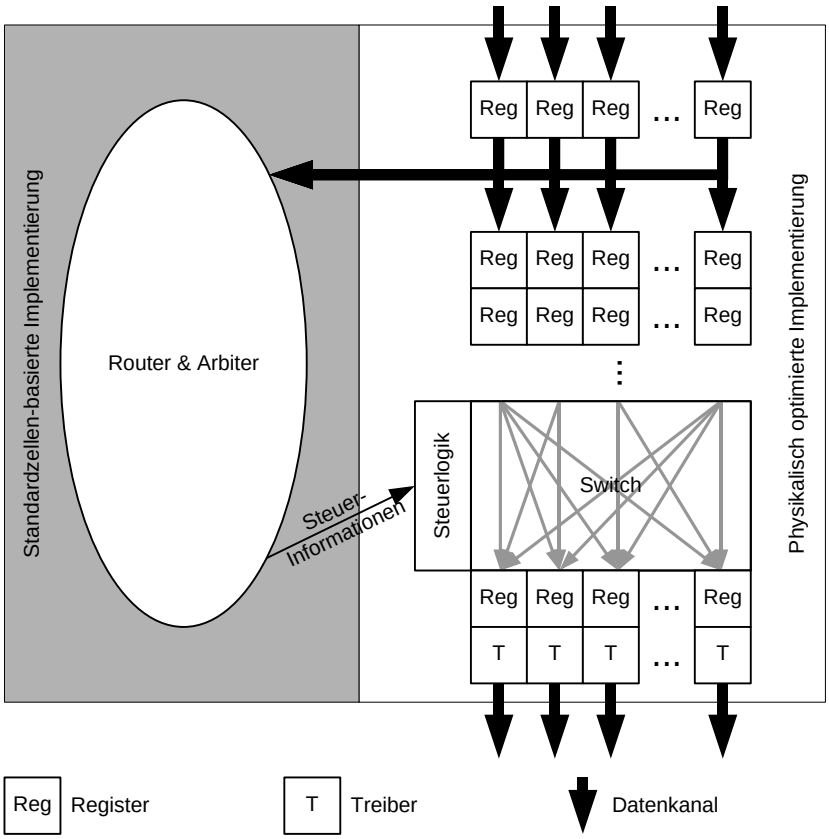


Abbildung 4.34: Übersicht optimierte Implementierung des Routing-Switch

Der Aufbau der Ein-Bit-Switches ist in Abbildung 4.36 gezeigt. Diese Ein-Bit-Switches bestehen aus einem 4 zu 1-Multiplexer für jeden Ausgangsport. Die implementierten Routing-Algorithmen unterstützen kein Rückwärts-Routing, d.h. dass Daten von einem Eingangsport 0 nicht an den gleichen Ausgangsport 0 zurück übertragen werden können. Dies wird durch die Minimal-Path-Bedingung der verwendeten Routing-Algorithmen sichergestellt (vgl. Kapitel 4.5.2 und 2.2.7). Deshalb ist für die Implementierung eines Routers mit 5-Ports ein 4 zu 1-Multiplexer pro Port ausreichend.

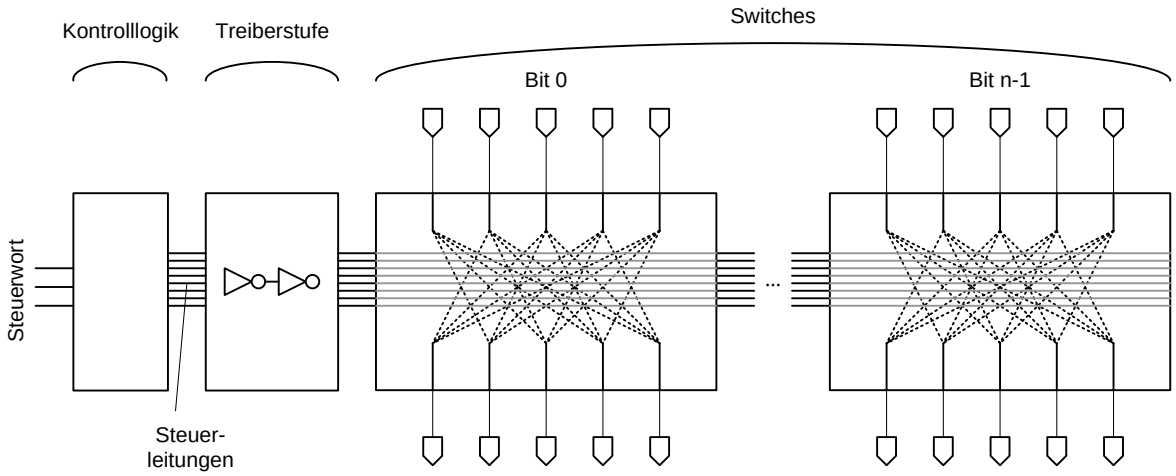


Abbildung 4.35: Aufbau des physikalisch optimierten Switches (ein Port)

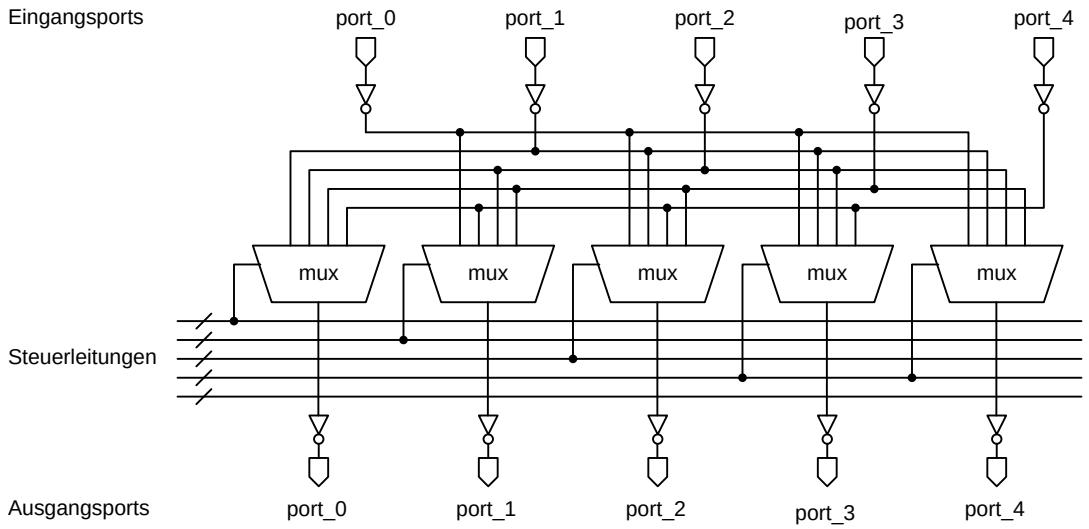


Abbildung 4.36: Aufbau der 1-Bit-Switches

Die Multiplexer (*mux*) sind als One Hot-Implementierung mit Transmission-Gates ausgeführt (vgl. Abbildung 4.37). Wenn keine Verbindung von einem Eingang zu einem Ausgang hergestellt werden soll, wird der Ausgang mit dem Null-Potenzial verbunden.

Mit Hilfe des in Kapitel 3.7.2 vorgestellten Datenpfad-Generators können Switches mit beliebiger Wortbreite und einer frei wählbaren Anzahl an Registern am Eingang erzeugt werden. Die dazu benötigten DPG-Beschreibungen existieren für drei, vier und fünf Ports.

Um diesen Datenpfad aufbauen zu können, werden insgesamt fünf Leaf-Zellen benötigt. Das Layout und die Schematics der Leaf-Zellen sowie weitere Informationen zur Implementierung sind in [99] gegeben. Gegebenenfalls ist eine weitere Optimierung dieser Leaf-Zellen für die zu erreichende Taktfrequenz und die Datenwortbreite notwendig. Insbesondere sind das die Treiber für die Steuerleitungen (Abbildung 4.35) und die Treiber an den Ausgangsports.

Diese müssen an die Last, die durch einen angeschlossenen Link oder ein angeschlossenes Network-Interface verursacht wird, angepasst werden.

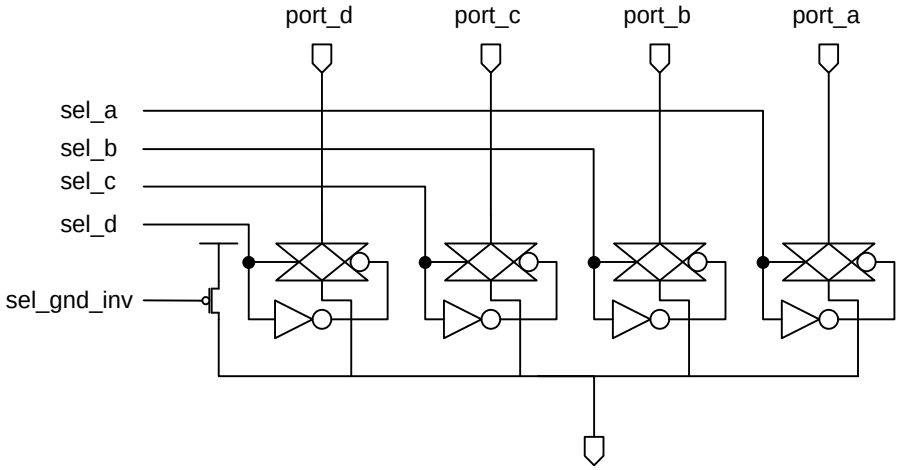


Abbildung 4.37: Schematic des mux für vier Eingänge und einen Ausgang

In Abbildung 4.38 ist das Layout einer Switch-Implementierung, wie er für die Vermittlungstechnik Wormhole-Routing benötigt wird, dargestellt. Er ist für fünf Ports und eine Wortbreite von 32 Bit aufgebaut worden. An jedem Eingang sind für jedes Bit jeweils vier Register vorgesehen.

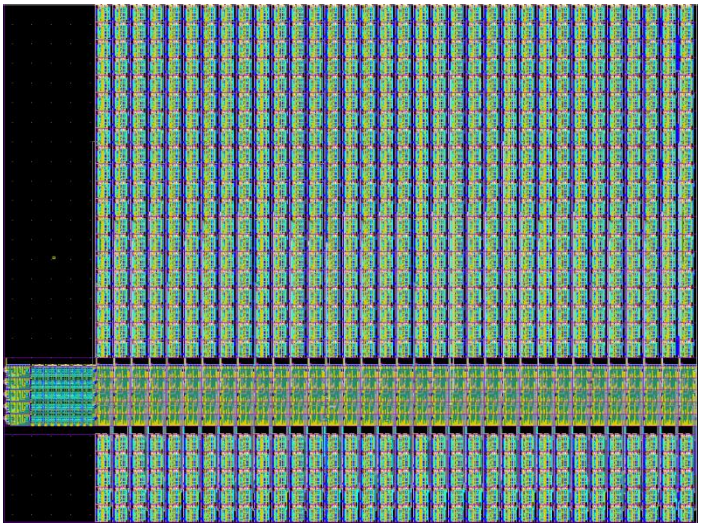


Abbildung 4.38: Layout und Verteilung der Blöcke im Layout von Registern und Switch für Wormhole-Routing mit 32 Bit Wortbreite, 5 Ports

Ein Vergleich der Fläche, die für den Switch in einer standardzellenbasierten Implementierung und mit einer Implementierung mit physikalisch optimierten Kernkomponenten benötigt wird, zeigt, dass die Fläche um ca. 50 % reduziert werden kann. Die Fläche, die von den Eingangsregistern benötigt wurde, konnte ebenfalls um ca. 50 % verringert werden (vgl. Abbildung 4.39 und Abbildung 4.26).

Die Abhängigkeit der Fläche eines Routing-Switches mit 5 Ports von der Wortbreite ist in Abbildung 4.39 gezeigt. Es ist eine lineare Abhängigkeit der Fläche von der Wortbreite zu erkennen. Da mit zunehmender Wortbreite die Last auf den Steuerleitungen immer weiter ansteigt, muss die Treiberstärke der dazugehörigen Treiber gesteigert werden, damit die maximal erreichbare Taktfrequenz weiterhin erreicht werden kann. Dieser zusätzliche Aufwand zur Steigerung der Treiberfähigkeit ist hier vernachlässigbar.

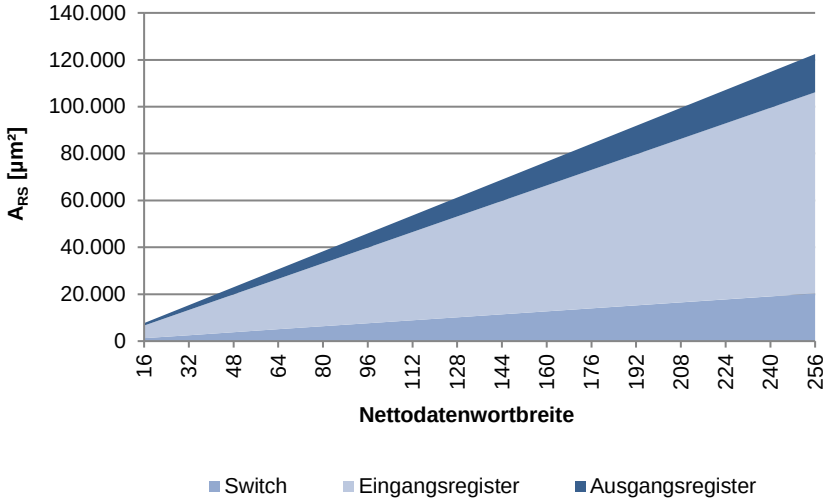


Abbildung 4.39: Fläche der einzelnen Komponenten eines physikalisch optimierten Switches mit 5 Ports und dazugehörigen Registern.

Durch die Verwendung von physikalisch optimierten Kernkomponenten für die Implementierung des Switches und der Register kann sowohl die Fläche eines Routing-Switch als auch die durch einen Routing-Switch verursachte Verlustleistung minimiert werden. Der Aufwand für eine solche Implementierung ist relativ gering. Es müssen für jede Technologie lediglich fünf Leaf-Zellen entworfen werden.

4.6 Einfluss der Topologie

Die Topologie eines NoC beschreibt die Anordnung der einzelnen NoC-Komponenten untereinander. Diese Anordnung hat einen Einfluss auf die Performance eines NoC und die durch das NoC verursachten Kosten. In diesem Abschnitt wird der Einfluss auf die Performance untersucht. Die Betrachtung der Kosten wird in Kapitel 5.3 im Rahmen einer Untersuchung des Entwurfsraumes durchgeführt.

In Abbildung 4.40 ist die Performance, dargestellt hinsichtlich der durchschnittlichen Latenz, für die Topologien Lineares Array, Mesh, Mesh mit externen NI, Ring, Star und Torus für unterschiedliche Auslastungen des NoC und zufälligen Datenverkehr aufgezeigt. Diese Untersuchungen wurden für die NoC mit 4, 9, 16 und 25 Teilnehmern auf einem FPGA-basierten Emulator durchgeführt.

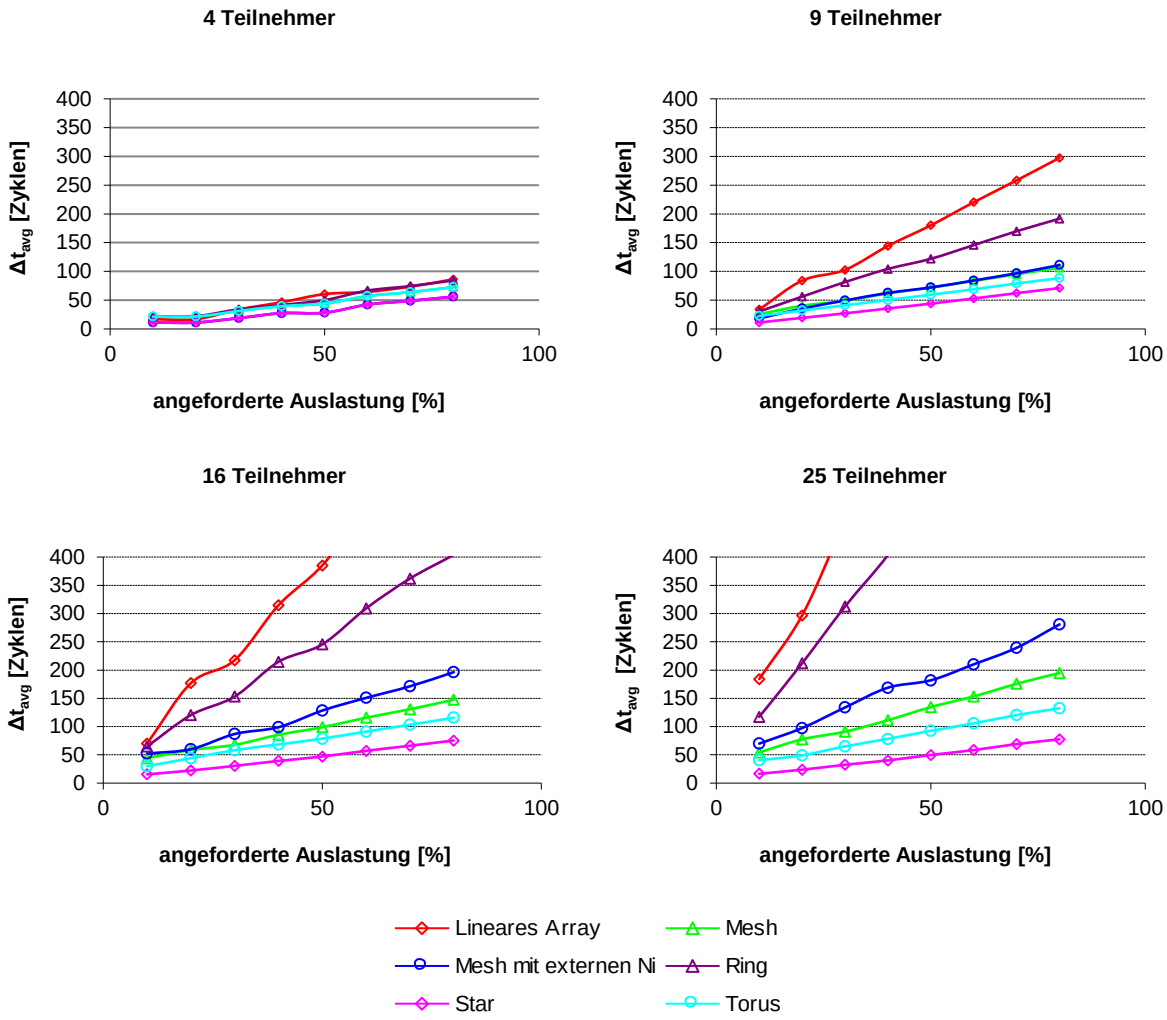


Abbildung 4.40: Einfluss der Topologie auf die Performance von NoC mit XY-Routing, Leitungsvermittlung für verschiedene Teilnehmeranzahlen

Für eine geringe Teilnehmerzahl von 4 Teilnehmern sind die Unterschiede in der Performance der NoCs mit den verschiedenen Topologien gering. Die Unterschiede zwischen den einzelnen Topologien sind bei dieser Größe gering. Die Stern-Topologie ist sogar identisch mit der Topologie Mesh mit externen NI. Bei größeren NoC entstehen große Performance-Unterschiede für die einzelnen Topologien. Je größer die angeforderte Auslastung ist, desto größer werden auch die Performance-Unterschiede. Die NoC mit Stern-Topologie besitzen die beste Performance, gefolgt von der Torus und Mesh Topologie. Die NoC mit der Topologie Mesh mit externen NI haben, außer bei kleinen NoC, eine schlechtere Performance als NoC, die eine Mesh-Topologie unterstützen. Die schlechteste Performance bieten die Topologien lineares Array und Ring. Dabei hat ein NoC mit Ring-Topologie eine bessere Performance als ein NoC, das als lineares Array angeordnet ist.

Bei der Auswahl einer geeigneten Topologie muss neben der Performance und den Kosten auch die physikalische Realisierbarkeit und das Layout des SoC berücksichtigt werden. Eine

Stern-Topologie ist beispielsweise nicht beliebig skalierbar. Bei einer zu großen Anzahl an Teilnehmern kann mit der in dieser Arbeit vorgeschlagenen Implementierung der Routing-Switches die vorgegebene Taktfrequenz von 500 MHz nur für NoC mit 10 Teilnehmern realisiert werden.

4.7 Einfluss der Vermittlungstechnik

Die untersuchten Varianten der Vermittlungstechnik Leitungsvermittlung und Wormhole-Routing haben einen erheblichen Einfluss auf die von den Network-Interfaces und Routing-Switches verursachten Kosten. Die Fläche ist für einen Routing-Switch bei Wormhole-Routing um ca. 80 % höher als bei Leitungsvermittlung (vgl. Kapitel 4.5.1), ein entsprechendes Network-Interface ist sogar um ca. 100 % größer (vgl. Kapitel 4.4.1). Durch die Verwendung optimierter Switches und Speicher kann dieser Mehraufwand jedoch reduziert werden.

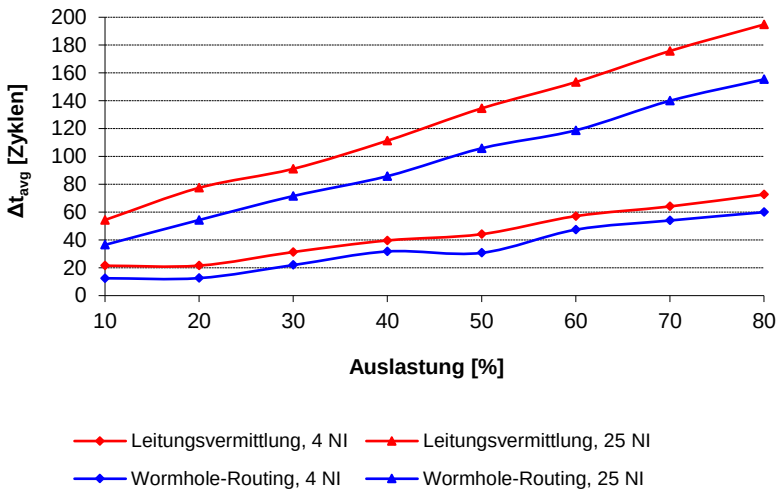


Abbildung 4.41: Einfluss der Vermittlungstechnik auf die Performance von NoC für verschiedene Teilnehmeranzahlen.

Die Auswirkungen, die die Vermittlungstechnik auf die Performance von NoC hat, sind in Abbildung 4.41 gezeigt. NoC, die Wormhole-Routing unterstützen, haben eine bessere Performance als ein vergleichbares NoC, das Leitungsvermittlung unterstützt. Der Performance-Unterschied ist umso größer, je höher die Auslastung des NoC ist. Je größer das NoC ist, umso größer ist der Performance-Gewinn.

Der Performance-Gewinn von Wormhole-Routing gegenüber Leitungsvermittlung, insbesondere bei großen NoC, lässt sich wie folgt erklären: bei der Leitungsvermittlung muss zunächst eine Verbindung zwischen zwei Network-Interfaces aufgebaut werden und dieser Aufbau dem sendenden Network-Interface bestätigt werden. Erst dann kann die eigentliche Datenübertragung erfolgen. Beim Wormhole-Routing wird dagegen die Datenübertragung mit

dem Verbindungsaufbau begonnen. Somit entfällt hier zweimal die Laufzeit über das NoC (Bestätigung des Verbindungsaufbaus und Datenübertragung). Diese Laufzeit ist eine Funktion der während der Datenübertragung benutzten Routing-Switches. Da bei großen NoC und zufälligem Datenverkehr die im Mittel benutzte Anzahl von Routing-Switches pro Datenübertragung größer ist als bei kleinen NoC, ist die Performance von Leitungsvermittlung überproportional schlechter.

Bei realen Anwendungen ist jedoch ein gleichmäßig verteilter Datenverkehr unwahrscheinlich, da funktionale Einheiten, die miteinander kommunizieren, beim Mapping möglichst nahe bei einander angeordnet werden. Daher wird, auch für große NoC die mittlere Anzahl besuchter Routing-Switches pro Datenübertragung nicht im gleichen Maße ansteigen, wie das NoC wächst.

4.8 Vergleich mit Referenzimplementierungen

Nachdem in den vorhergehenden Abschnitten der Entwurfsraum von NoC-Komponenten aufgespannt wurde, sollen nun exemplarisch die Implementierung des Routing-Switches mit Implementierungen aus der Literatur verglichen werden. Dort ist eine Vielzahl von NoC-Implementierungen vorgestellt worden. Diese unterscheiden sich in ihrem Funktionsumfang (z.B. Vermittlungstechnik, Topologie etc.), der verwendeten Technologie und den Kosten. Als Referenz wurden zwei Routing-Switch Implementierungen gewählt, die auf den Referenzparametern, die in Tabelle 4.1 gegeben sind, basieren. Die Implementierung mit dem Namen EECS-LV verwendet Leitungsvermittlung, die Implementierung EECS-WH Wormhole-Routing. In Tabelle 4.6 ist eine Übersicht von Routing-Switch-Implementierungen gegeben.

Damit die Fläche, Verlustleistung und die maximale Taktfrequenz der einzelnen Routing-Switches fair miteinander verglichen werden können, wurde eine Technologieskalierung auf eine Referenztechnologie, hier eine 90 nm Technologie, durchgeführt. Dazu wurden die üblichen Funktionen zur Deep-Submicron-Skalierung, wie sie beispielsweise in [18] vorgestellt wurden, verwendet. Der zur Skalierung benötigte Skalierungsfaktor *s* berechnet sich aus der Technologiefeinheit der zu skalierenden Komponente λ und der Technologiefeinheit der Referenztechnologie λ_{ref} zu

$$s = \frac{\lambda}{\lambda_{ref}}. \tag{4.1}$$

Mit Hilfe dieses Skalierungsfaktors berechnet sich die normierte Fläche mit

$$A_{norm} = \frac{A}{s^2} \tag{4.2}$$

und die normierte maximale Taktfrequenz mit

$$f_{max,norm} = f_{max} \cdot s \tag{4.3}$$

bestimmen. Mit Hilfe dieser Normierungsformeln wurden die in Tabelle 4.6 aufgeführten Implementierungen in das Diagramm in Abbildung 4.42 eingetragen, mit der (normierten) maximalen Taktfrequenz auf der Ordinate und der (normierten) Fläche pro Bit auf der Abszisse. Die Vierecke repräsentieren Routing-Switches, die Leitungsvermittlung unterstützen, die Rauten stehen für Routing-Switches, die Paketvermittlung nutzen.

Tabelle 4.6: Vergleich von Routing-Switch-Implementierungen

Name	**Technologie**	**Ports**	**Wortbreite**	**A [mm²]**	**f_{max} [GHz]**	**Referenz**
XPipes	130 nm	5	32	0,188	0,80	[100]
Aetheral	130 nm	5	32	0,188	0,50	[46]
Intel	65 nm	5	36	0,047	4,00	[101]
STM[1]	130 nm	5	32	0,625	0,16	[102]
4S (Circuit)	130 nm	5	16	0,050	1,01	[103]
4S (Packet)	130 nm	5	16	0,086	0,51	[103]
KAIST	180 µm	5	8	---	1,60	[42]
Proteo[2]	180 µm	3	8	0,036	0,2	[104]
SoCBus	180 µm	5	16	0,050	1,2	[51]
EECS-LV	90 nm	5	32	0,013	1,00	Diese Arbeit
EECS-WH	90 nm	5	32	0,041	0,50	Diese Arbeit

Durch die Verwendung eines sehr schlanken Protokolls, in dem auf Speicher in den Routing-Switches verzichtet werden kann, sind die im Rahmen dieser Arbeit erstellten Routing-Switch-Implementierungen kleiner als vergleichbare Implementierungen. Lediglich die Implementierung SoCBus liegt bei der Fläche in einer ähnlichen Größenordnung ist jedoch schneller. Die Routing-Switches, wurden für eine maximale Taktfrequenz von 1 GHz bzw. 500 MHz optimiert. Eine Optimierung für höhere Taktfrequenzen ist möglich. Allerdings hat dies eine gesteigerte Fläche zur Folge. Dieser Anstieg der Fläche mit der Frequenz wird zunächst nur moderat, ab einer bestimmten Frequenz jedoch exponentiell sein [100]. Um die Taktfrequenz weiter zu steigern, sind dann strukturelle Änderungen an den NoC-Komponenten notwendig.

[1] Implementierung von Routing-Switch und Network-Interface

[2] Der Router besitzt einen Eingang, einen Ausgang und einen Ein- und Ausgang

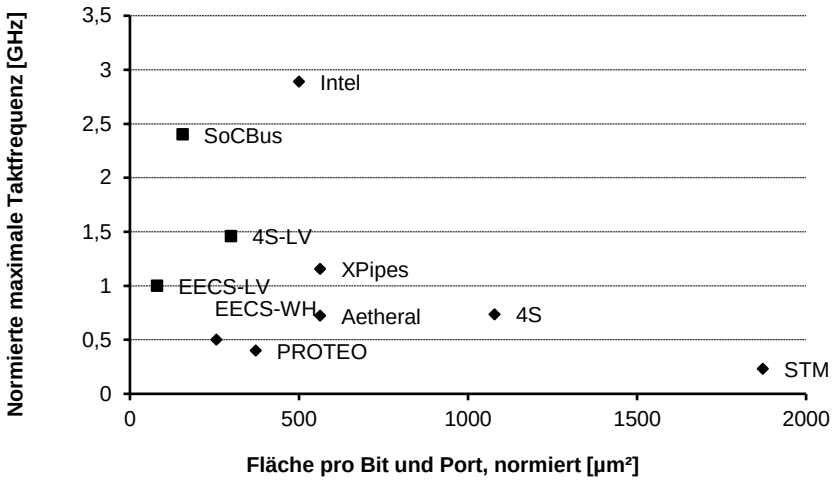


Abbildung 4.42: Vergleich von Routing-Switches mit Referenzimplementierungen bezügliche der Fläche/Bit und der maximalen Taktfrequenz. Werte sind auf eine 90 nm Technologie normiert.

4.9 Zusammenfassende Bewertung der NoC-Parameter

Nachdem in den vorangegangenen Abschnitten dieses Kapitels eine quantitative Analyse des Einflusses von NoC-Parametern auf die durch ein NoC verursachten Kosten sowie die Performance eines NoC durchgeführt wurde, ist in Tabelle 4.7 zusammenfassend eine qualitative Bewertung dieser Einflüsse vorgenommen worden. Diese Tabelle ist die Grundlage für die Reihenfolge, in der die Parameter bei dem Spiral-Modell des NoC-Entwurfs, das in Kapitel 3.8 vorgestellt wurde, variiert werden.

Tabelle 4.7: Qualitative Bewertung von NoC-Parametern auf Kosten und Performance (++ := großer Einfluss; + := geringer Einfluss; o := kein Einfluss)

NoC-Parameter	Fläche	Verlustleistung	Performance
Topologie	++	++	++
Wortbreite	++	++	++
Taktfrequenz	++	++	++
Vermittlungstechnik	++	o - ++[1]	+
Routing-Algorithmus	o	o	+
Verbindungsfehlerbehandlung	o	o	o
Datenfehlerbehandlung	++	++	++
Priorisierung von Verbindungen	+	o	+
Implementierungsvarianten	++	++	o

[1] Abhängig von der Implementierung des Speichers im Network-Interface

5 Kostenmodellierung von NoC

Die Schätzung der Kosten für eine NoC-Implementierung eines Systems zu einem frühen Zeitpunkt im Entwurfsprozess ist ein elementarer Bestandteil der in Kapitel 3 vorgestellten Entwurfsmethodik. Ziel dieser Kostenschätzung ist die Identifikation von NoC, die geringen Kosten verursachen, um diese weiter zu untersuchen. Auf die zeitintensive Implementierung von NoC mit zu hohen Kosten und das Verwerfen dieser Implementierungen in einer späten Entwurfsphase kann so verzichtet werden. Die Kostenschätzung darf deswegen nur mit geringem Aufwand verbunden sein. Da auf Basis dieser Kostenschätzung lediglich eine Vorauswahl von interessanten NoC getroffen wird, die anschließend näher untersucht werden, kann die Genauigkeit der Kostenschätzung zugunsten der Rechenzeit abgetauscht werden (vgl. Kapitel 3.3).

Um diesen Anforderungen gerecht zu werden, wurden mathematische Kostenfunktionen für die einzelnen NoC-Komponenten aufgestellt. Die Kosten, die modelliert wurden, sind die benötigte Fläche und die Verlustleistung. Zusätzlich wurde das zeitliche Verhalten der NoC-Komponenten modelliert. Mit Hilfe dieser Funktionen kann beispielsweise die Latenz, die bei einer Datenübertragung über ein NoC entsteht, bestimmt werden. Diese Funktionen werden bei der, im Rahmen dieser Arbeit entwickelten, statischen Performance-Analyse oder bei der SystemC- und CPN-basierten Simulation verwendet.

Bei der Modellierung wird der modulare und generische Aufbau des NoCs genutzt. So kann das komplexe Problem der Kostenschätzung eines NoC mit seiner Vielzahl an Parametern in viele kleine und leicht zu überschaubare Kostenschätzungen einzelner NoC-Komponenten aufgeteilt werden. So wird das NoC in die einzelnen Komponenten Routing-Switch, Network-Interface und Link aufgeteilt. Jede dieser Komponenten kann isoliert modelliert werden. Diese können gegebenenfalls weiter unterteilt werden, bis die zu modellierende Einheit nur noch von wenigen Parametern abhängt. Die Modellfunktionen basieren auf den in Kapitel 4 vorgestellten Ergebnissen.

5.1 Modellierungsmethodik

5.1.1 Zeitliches Verhalten

Das zeitliche Verhalten eines NoC kann zum einen für eine einzelne Datenübertragung zwischen zwei funktionalen Einheiten beschrieben werden, oder für Datenverkehr, der aus mehreren Datenübertragungen zwischen verschiedenen funktionalen Einheiten besteht.

Eine einzelne Datenübertragung wird primär durch die Übertragungsdauer t_{real} charakterisiert. Sie beinhaltet die komplette Zeit, die benötigt wird, um eine Datenmenge von einer funktionalen Einheit A zu einer funktionalen Einheit B zu übertragen. Sie beginnt, sobald das

erste Datum beim sendenden Network-Interface zum Übertragen bereit ist und endet, wenn das letzte Datum von der empfangenden funktionalen Einheit übernommen wurde.

Da diese Zeit von der Datenmenge abhängt, wird stattdessen häufig die Latenz Δt der Datenübertragung bestimmt. Diese ist unabhängig von der übertragenen Datenmenge. Dabei wird die Zeit von der Übertragungsdauer t_{real} abgezogen, welche die Daten für eine Übertragung zwischen zwei funktionalen Einheiten ohne eine durch ein NoC verursachte Verzögerung benötigt hätten. Diese Zeit t_{opt} lässt sich durch die Multiplikation der Datenrate r mit der Datenmenge D bestimmen (vgl. Formel (2.3)), wenn, wie hier angenommen, die empfangende funktionale Einheit die Daten direkt aufnehmen kann.

$$\Delta t = t_{real} - t_{opt} \tag{5.1}$$

Eine dritte Kenngröße, die das zeitliche Verhalten beschreibt, ist die Verbindungsdauer t_C. Sie enthält neben der Übertragungsdauer auch die Zeit, die NoC-Komponenten nach einer Datenübertragung noch belegt sind (z.B. Verbindungsabbau bei Leitungsvermittlung). Für die Modellierung dieser Zeiten wird angenommen, dass die Daten über ein unbelastetes NoC übertragen werden. Eine detaillierte Beschreibung der Modellierung des zeitlichen Verhaltens, inklusive entsprechender Modellfunktionen, ist in [105] gegeben.

5.1.2 Fläche

Für eine VLSI-Implementierung mit Hilfe von Standardzellen kann die Fläche durch die Anzahl der äquivalenten Gatter beschrieben werden. Diese Zahl kann durch die Synthese des VHDL-Codes bestimmt werden, ist jedoch wenig aussagekräftig. Exakter ist die Angabe der benötigten Fläche. Dies kann nach der Synthese und nach dem Place and Route-Schritt bestimmt werden. Nach der Synthese sind die Flächenwerte ungenauer, da hier die für die Verbindung der Standardzellen benötigte Fläche, sowie die daraus entstehenden parasitären Kapazitäten, lediglich abgeschätzt werden können. Diese Kapazitäten haben einen Einfluss auf die erreichbare Taktfrequenz und somit auf die Größe der benötigten Treiber. Im Place and Route-Schritt werden die Standardzellen platziert und untereinander verbunden. Somit kann die Fläche für die Verbindungen zwischen den Zellen bestimmt und berücksichtigt werden. Nach diesem Schritt wird das Design nicht mehr verändert. Daher entspricht die Fläche, die eine NoC-Komponente nach dem Place and Route-Schritt benötigt auch derjenigen Fläche, die auf dem späteren NoC benötigt wird.

Dieser, in Kapitel 3.7 vorgestellte, Design-Flow besitzt eine Vielzahl von Parametern, mit deren Hilfe das Ergebnis von Synthese und Place and Route beeinflusst wird. Eine gezielte Optimierung kann jedoch nicht vollständig automatisch erfolgen. Um ein optimales Ergebnis zu erhalten, insbesondere bei hohen Anforderungen bezügliche Fläche, Timing und Verlustleistung, müssen individuelle Optimierungen vorgenommen werden. Zur Bestimmung

der Flächen, die die Grundlage für die Modellierung bilden, wurde hier ein automatisiertes Verfahren genutzt, das bereits in Kapitel 4.3 beschrieben wurde. Daher sind die Ergebnisse und die daraus abgeleiteten Modellfunktionen tendenziell zu pessimistisch, da eine individuelle Optimierung der Synthese-Parameter nicht vorgenommen wurde. Für eine Implementierung mit Hilfe von physikalisch optimierten Zellen kann die Fläche dieser physikalisch optimierten Zellen genau auf Basis der DPG-Beschreibung (vgl. Kapitel 3.7.2) und der implementierten Leaf-Zellen errechnet werden.

Die Summe der so bestimmten Flächen der einzelnen Routing-Switches und der einzelnen Network-Interfaces bestimmt so die Fläche eines NoCs mit n_{RS} Routing-Switches, n_{NI} Network-Interfaces und n_L Links:

$$A_{NoC} = \sum_{i=1}^{n_{RS}} A_{RS,i} + \sum_{i=1}^{n_{NI}} A_{NI,i} + \sum_{i=1}^{n_L} A_{L,i} \; . \tag{5.2}$$

Die Fläche der Links berücksichtigt hier lediglich die Siliziumfläche, die von ggf. benutzten Registern oder Buffern auf der Verbindungsleitung verwendet wird. Die Fläche für die eigentlichen Verbindungsleitungen wird in der Fläche A'_{NoC} zusammengefasst und separat abgeschätzt.

$$A'_{NoC} = \sum_{i=1}^{n_L} A'_{L,i} \tag{5.3}$$

Die zur Implementierung eines NoC benötigte Fläche kann bei einer FPGA-Implementierung, wie sie beispielsweise für den FPGA-basierten Emulator verwendet wird, in benötigten Logikelementen ausgedrückt werden. Dazu muss die Synthese und der Place and Route-Schritt mit einem entsprechenden Entwurfswerkzeug für FPGAs durchgeführt werden. Im Rahmen dieser Arbeit wurde die Software Quartus II von Altera verwendet [79]. Die Gleichung (5.2) gilt entsprechend, wobei die Flächen A_x. durch die Anzahl der benötigten Logikelemente LE_x ersetzt wird.

Die Details der Flächenmodellierung für eine VLSI-Implementierung mit Standardzellen und mit physikalisch optimierten Zellen, wie auch für eine FPGA-Implementierung sind in [105] angegeben.

5.1.3 Energie und Verlustleistung

Die Energie, die benötigt wird, um die Daten die für ein gegebenes System zu übertragen, hängt neben den Anforderungen des Systems, von der Implementierung des NoC sowie von der Zuordnung der funktionalen Einheiten zu den Network-Interfaces eines NoC ab (Mapping). Um diese Energie zu bestimmen, kann zunächst der gesamte Datenverkehr in einzelne Datenübertragungen aufgeteilt werden. Für diese Datenübertragungen setzt sich die

Energie aus der Energie, die im sendenden und empfangenden Network-Interface, sowie in den zur Übertragung benötigten Links und Routing-Switches umgesetzt wird, zusammen.

$$E_C = E_{NI-Send} + \sum_{i=1}^{L}\left(E_{LINK,i}\right) + \sum_{i=1}^{H}\left(E_{RS,i}\right) + E_{NI-Receive} \tag{5.4}$$

Die Summe der Energien der einzelnen Datenübertragungen, sowie die Energie, die von den NoC-Komponenten im Ruhezustand umgesetzt wird, ergibt die Gesamtenergie, die von einem System zur Datenübertragung über ein NoC benötigt wird. Die Energie, die von den an das NoC angeschlossenen funktionalen Einheiten benötigt wird, wird hier nicht betrachtet.

Grundsätzlich lässt sich die von einem System auf einem NoC umgesetzten Energie in einen statischen Anteil aufteilen, der umgesetzt wird, wenn alle NoC-Komponenten währen der gesamten Ausführungszeit des Systems im Ruhezustand sind. Dazu lässt sich ein dynamischer Anteil addieren, der vom Datenverkehr abhängt

$$E_{NoC} = E_{NoC,stat} + E_{NoC,dyn}\,. \tag{5.5}$$

Der statische Anteil ergibt sich durch die Aufsummierung der individuellen Verlustleistung der einzelnen NoC-Komponenten und der Multiplikation dieser Verlustleistung des NoC im Ruhezustand mit der zur Ausführung des Systems benötigten Zeit

$$E_{NoC,stat}(T) = \left(\sum_{n=1}^{n_{NI}} P_{NI,stat,n} + \sum_{r=1}^{n_{RS}} P_{RS,stat,r} + \sum_{l=1}^{n_L} P_{L,stat,l}\right)\cdot T\,. \tag{5.6}$$

Für gleichmäßig aufgebaute NoC kann der dynamische Anteil der Verlustleistung durch Formel 5.7 abgeschätzt werden. Dabei kann die Energie, die für fehlgeschlagene Datenübertragungen, sowie die für Kontroll-Overhead benötigte Energie, durch eine entsprechend verlängerte Ausführungszeit *T* des Systems berücksichtigt werden. Die Herleitung dieser Formel wird in [105] dargestellt.

$$E_{NoC,dyn}(n_{NI}, T, l, H_{avg}, L_{avg}) = n_{NI}\cdot T\cdot l\cdot\begin{pmatrix}\Delta P_{NI-Send} + \\ H_{avg}\cdot\Delta P_{RS} + \\ L_{avg}\cdot\Delta P_{L} + \\ \Delta P_{NI\_Receive}\end{pmatrix} \tag{5.7}$$

Der dynamische Anteil der Energie hängt somit von der dynamischen Verlustleistung der NoC-Komponenten (ΔP_x), der Anzahl der Network-Interfaces (n_{NI}), der durchschnittlich besucht Links (L_{avg}) und Routing-Switches (H_{avg}), der Ausführungszeit des Systems auf dem NoC (T) sowie erreichte Auslastung des NoCs (l) ab.

Für die statische und dynamische Verlustleistung der NoC-Komponenten wurden auf Basis der in Kapitel 4 vorgestellten Implementierungen mathematische Modellfunktionen bestimmt.

Da in den implementierten NoC grundsätzlich Minimal Path Routing verwendet wird, lässt sich die Anzahl der durchschnittlich besuchten Routing-Switches und Links im Rahmen einer statischen Performance-Analyse bestimmen. Die Ausführungszeit des Systems auf dem NoC, sowie die erreichte Auslastung des NoCs, kann durch eine dynamische Performance-Analyse, beispielsweise mit Hilfe des FPGA-basierten Emulators, ermittelt werden.

Die durchschnittliche Verlustleistung eines NoC kann analog berechnet werden:

$$P_{NoC} = P_{NoC,stat} + P_{NoC,dyn} \tag{5.8}$$

$$P_{NoC,stat} = \left(\sum_{n=1}^{n_{NI}} P_{NI,stat,n} + \sum_{r=1}^{n_{RS}} P_{RS,stat,r} + \sum_{l=1}^{n_L} P_{L,stat.l} \right) \tag{5.9}$$

$$P_{NoC,dyn}(n_{NI}, l, H_{avg}, L_{avg}) = n_{NI} \cdot l \cdot \begin{pmatrix} \Delta P_{NI-Send} \\ +H_{avg} \cdot \Delta P_{RS} \\ +L_{avg} \cdot \Delta P_{L} \\ \Delta P_{NI\_Receive,D} \end{pmatrix} \tag{5.10}$$

5.2 Bewertung der Modellierungsqualität

Anhand zahlreicher Implementierungen wurde die Genauigkeit der Modellfunktionen überprüft. Dazu wurden die Ergebnisse der Modellfunktionen mit allen Ergebnissen der Implementierungen verglichen, die für die Modellbildung verwendet wurden. Für die Fläche ergibt sich ein maximaler relativer Fehler von 5 % und für die Verlustleistung bzw. Energie ein maximaler relativer Fehler von 20 %. Dieser Fehler wird durch die Modellierungsmethodik verursacht. Zum Beispiel wird bei einer linearen Abhängigkeit eine gerade durch die Messwerte gelegt, die jedoch von einzelnen Messwerten eine geringe Abweichung aufweisen kann.

Durch eine Optimierung der Parameter, welche die Synthese bzw. das Place and Route beeinflussen, können die Ergebnisse der Kostenschätzung weiter verbessert werden. Bei der Modellierung der Verlustleistung wurde die Datenabhängigkeit der Verlustleistung nicht berücksichtigt. Durch diese Berücksichtigung kann die Genauigkeit, sofern Informationen über die zu übertragenden Daten, wie zum Beispiel der Schaltaktivität, vorliegen, weiter gesteigert werden.

5.3 Untersuchung des Entwurfsraumes für zufälligen Datenverkehr

Im Folgenden wird exemplarisch der Entwurfsraum von NoC mit 16 Teilnehmern für zufälligen Datenverkehr mit einer angeforderten Gesamt-Datenrate von ca. 95 GBit/s untersucht. Im Rahmen dieser Untersuchung wurden die Datenwortbreite, die Topologie, der

Routing-Algorithmus und die Vermittlungstechnik variiert. Die Fläche wurde anhand der in diesem Kapitel vorgestellten Modellfunktionen bestimmt.

Die Performance wurde mit Hilfe des FPGA-basierten Emulators für die jeweilige NoC-Implementierung und einer übertragenen Datenmenge von 2,3 GBit bestimmt. Durch die Ergebnisse der Performance-Analyse konnten unter Verwendung der Modellfunktionen für die Verlustleistung die mittlere Verlustleistung und die für die Datenübertragung benötigte Energie bestimmt werden.

Die Ergebnisse dieser Untersuchungen wurden in Abbildung 5.1 für die Performance, ausgedrückt durch die mittlere Latenz, über verschiedenen Kostenmaßen aufgetragen. Als Kostenmaße wurde hier exemplarisch die durchschnittliche Verlustleistung, die für die Datenübertragung benötigte Energie, die Fläche sowie das Fläche-Energie-Produkt (AE) verwendet. Jeder der in den Diagrammen eingetragenen Punkte repräsentiert eine NoC-Implementierung.

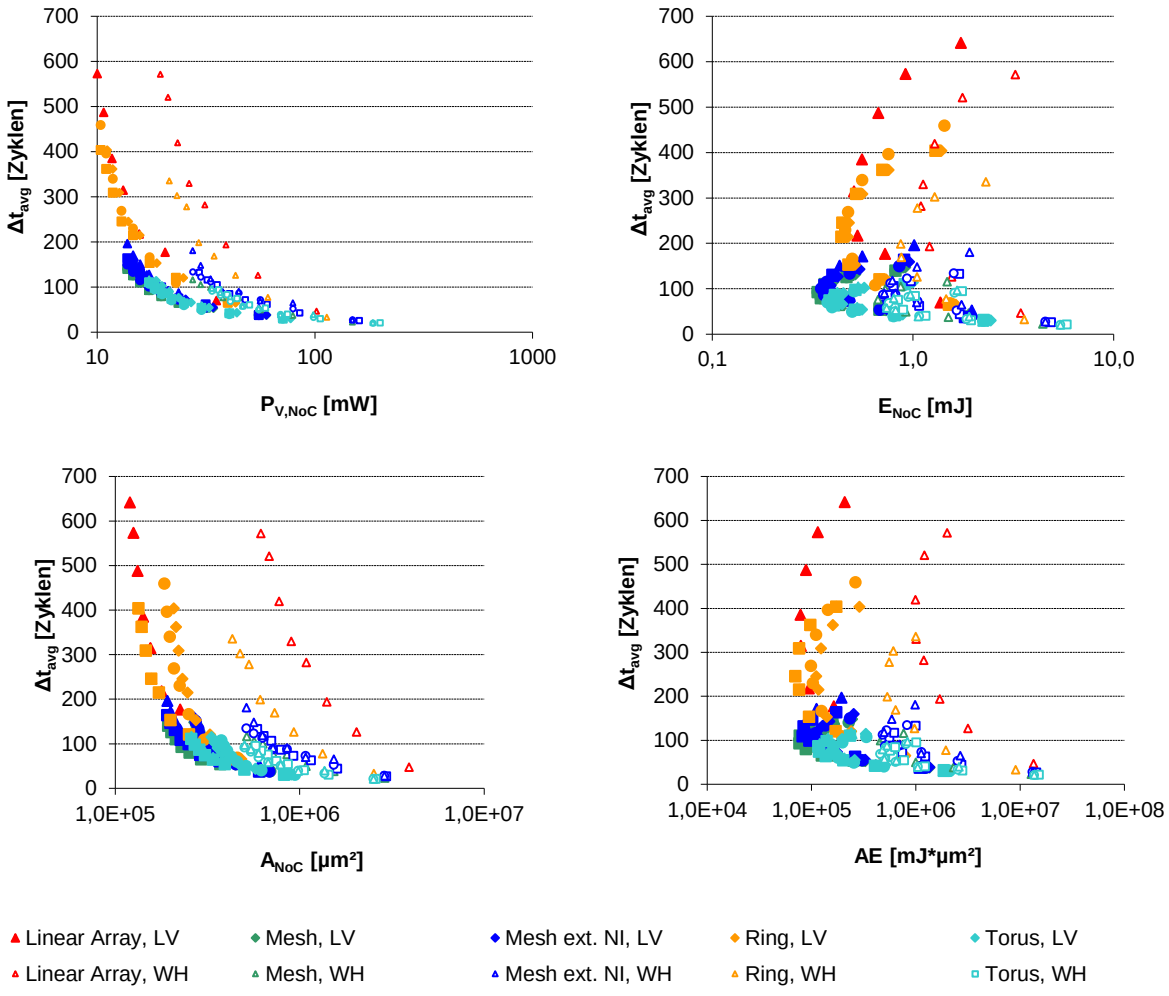


Abbildung 5.1: Pareto-Plot von NoC-Implementierungen für zufälligen Datenverkehr mit einer durchschnittlichen angeforderten Datenrate von 95 GBit/s.

Je nach Kostenfunktion ergeben sich in den Pareto-Plots unterschiedliche Pareto-Fronten. Für eine gegebene Kostenfunktion und eine mindestens erreichbare Performance (hier mittlere Latenz) kann so die passende NoC-Implementierung ausgewählt werden.

Pareto-optimalen NoC-Implementierungen für alle Kostenmaße für eine maximal zulässige mittlere Latenz von 100 Taktzyklen werden durch die Mesh-Topologie mit Leitungsvermittlung und einer Datenwortbreite von 64 Bit beschrieben. Für die Kostenmaße Verlustleistung und Energie ist adaptives XY-Routing mit Backtracking optimal, für die Kostenmaße Fläche sowie das Fläche-Energie-Produkt verursacht adaptives XY-Routing die geringsten Kosten.

6 Anwendungsbeispiel Entwicklungsmethode

In Kapitel 3.8 wurde das Spiralmodell für die Entwicklung von NoC vorgestellt. In Kapitel 6.1 wird dieses Modell auf drei reale Anwendungen aus dem Bereich der Videosignalverarbeitung angewendet. Die so ermittelten NoC werden anschließend mit NoC-Implementierungen für dieses System verglichen, die in [106] und [107] präsentiert wurden.

Der Datenverkehr, der durch die untersuchten Systeme der Videosignalverarbeitung verursacht wird, stellt nur geringe Kommunikationsanforderungen an das NoC. Aus diesem Grund werden in Kapitel 6.2 synthetische Task-Graphen vorgestellt, die höhere Kommunikationsanforderungen an das NoC stellen. Für diese Task-Graphen wurden ebenfalls Pareto-optimale NoC entwickelt.

6.1 Anwendung der Entwurfsmethodik für Systeme aus dem Bereich der Video-Signal-Verarbeitung

Bei den Systemen aus dem Bereich der Videosignalverarbeitung handelt es sich um ein System für ein Multi-Window-Display (MWD), ein System eines MPEG4-Decoders sowie ein System eines Video Object Plane Decoders. Diese Systeme, sowie der durch diese Systeme verursachte Datenverkehr, wurden in [108] vorgestellt.

Die Task-Graphen, die diesen Datenverkehr beschreiben, sind in Abbildung 6.1, Abbildung 6.2 und Abbildung 6.3 dargestellt. Die Werte an den Kanten in den Abbildungen stellen die angeforderte Datenrate in MBit/s dar. Die Periodendauer des Graphen wird für die folgenden Untersuchungen zu 0,02 s angenommen, was einer Bildfrequenz von 50 Hz entspricht. Da es sich bei den hier vorgestellten Systemen um Echtzeitsysteme handelt, bei denen das Periodenende eine Deadline darstellt, müssen zum Ende einer Periode alle Datenübertragungen abgeschlossen sein. Eine weitere Eigenschaft des Datenverkehrs ist die Granularität. Diese beschreibt, ob die Daten als ein monolithischer Block oder auf mehrere kleinere Datenblöcke verteilt übertragen werden. In [108] werden keine Angaben bezüglich der Granularität gemacht. Der Einfluss der Datenblockgröße auf die Performance des NoCs wird in Kapitel 6.1.6 separat untersucht.

Die Technologie, die verwendet werden soll, ist eine 90 nm Technologie. Die Implementierung soll mit Standardzellen erfolgen, die in [109] beschrieben sind. Es wurde weiterhin angenommen, dass die Länge der Verbindungsleitungen zwischen den NoC-Komponenten 1 mm entspricht.

Die optimierten NoC werden in dieser Untersuchung in vier Iterationen mit dem Spiralmodell erstellt. Dieses sieht in diesem Fall vier Iterationen vor, in denen die Topologie, die

Taktfrequenz und die Datenwortbreite, die Vermittlungstechnik sowie abschließend der Routing-Algorithmus ausgewählt wird.

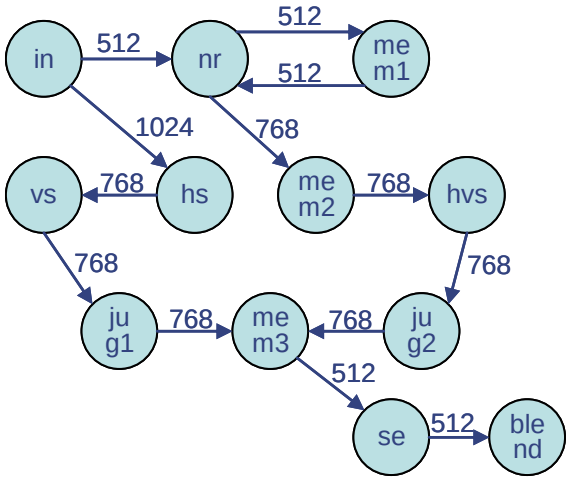


Abbildung 6.1: Task-Graph Multi-Window-Display (MWD)

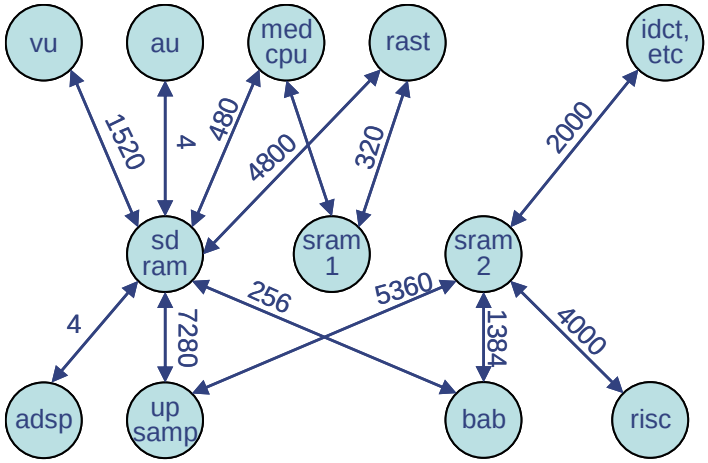


Abbildung 6.2: Task-Graph MPEG4-decoder (MPEG4)

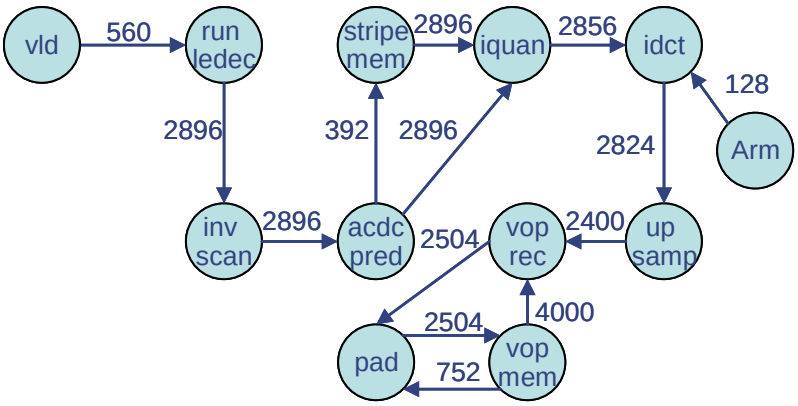


Abbildung 6.3: Task-Graph Video Object Plane Decoder (VOPD)

6.1.1 Iteration 1 - -Topologie

Während der ersten Iteration werden die Topologie und die Dimensionen des NoCs gewählt, sowie ein passendes Mapping des Task-Graphen auf das NoC bestimmt. Als Bewertungsmaße für diese Iteration dienen die von dem NoC benötigte Fläche als Kostenmaß sowie die mit der jeweils übertragenen Datenmenge gewichtete Anzahl der Hops als Performance-Maß. Dieses Performance-Maß bietet sich an, da die gewichtete Anzahl der Hops proportional zum dynamischen Anteil der Verlustleistung ist, die somit auch berücksichtigt wird. Die Fläche wird mit Hilfe der in Kapitel 5 vorgestellten Kostenfunktionen berechnet, die gewichtete Anzahl der benötigten Hops wird im Rahmen des Mappings (vgl. Kapitel 3.4) des Task-Graphen auf das NoC bestimmt.

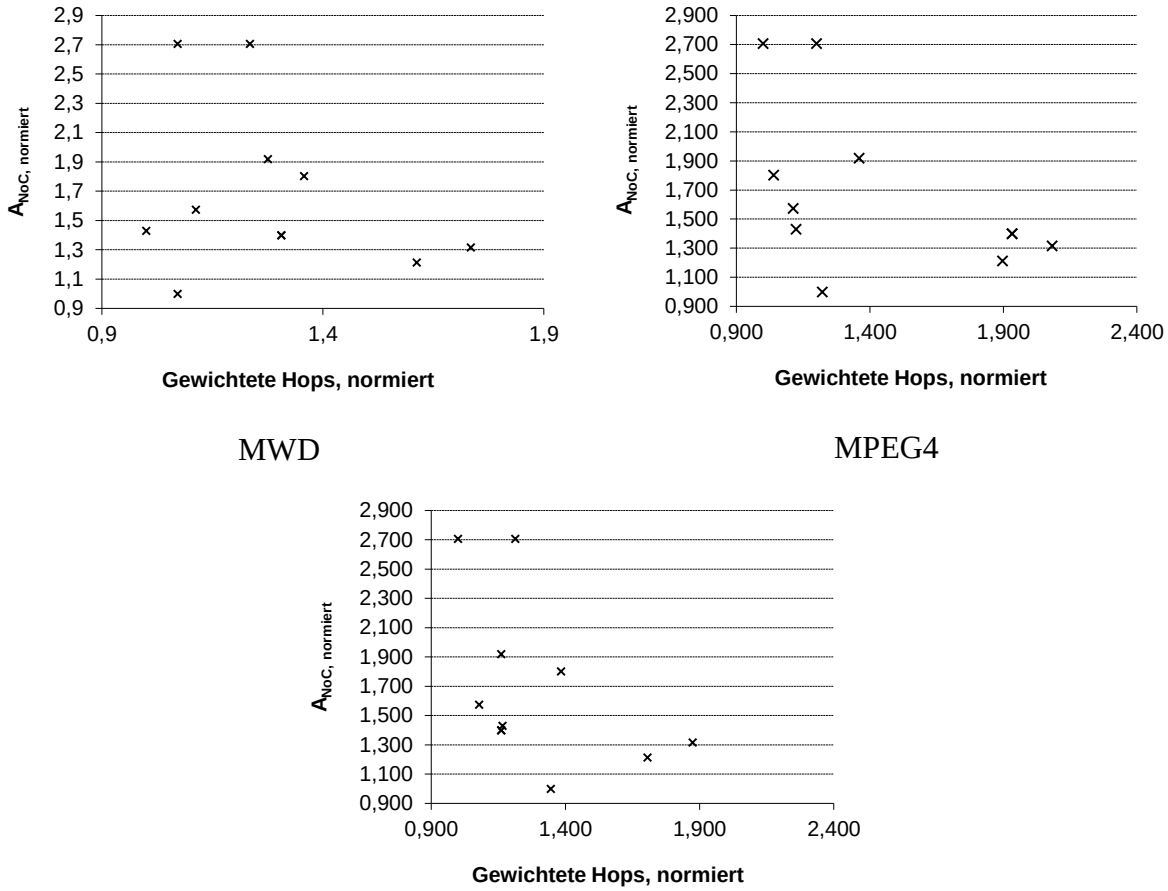


Abbildung 6.4: Ergebnisse der ersten Iteration des Design-Flows für variierte Topologien und Dimensionen.

Der Parameter Datenwortbreite hat einen großen Einfluss auf die Fläche des NoC. Da dieser Parameter im gegenwärtigen Entwurfsschritt noch nicht feststeht, wird im Folgenden die

Fläche des NoC relativ zur Fläche des kleinsten NoC dieser Iteration angegeben. Um die Daten übersichtlicher darstellen zu können wird dasselbe Verfahren bei dem zweiten Bewertungsmaß abgewendet: die gewichtete Anzahl der Hops wird ebenfalls auf den kleinsten Wert dieser Untersuchung bezogen.

Die Ergebnisse dieser ersten Iteration (unterschiedliche Topologien) sind in den Diagrammen in Abbildung 6.4 beziehungsweise in Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3 dargestellt. Die bezüglich der Fläche und gewichteten Hops Pareto-optimalen NoC-Topologien sind in den Tabellen grau markiert. Diese bilden die Basis für die Parametervariation der nächsten Iteration.

Tabelle 6.1: Für MWD untersuchte NoC-Topologien mit normierter Fläche und normierten Mapping-Kosten

Topologie	**X-Dim**	**Y-Dim**	**Gewichtete Hops, normiert**	**$A_{NoC,normiert}$**
Ring	12	1	1,399	1,399
Lineare Anordnung	12	1	1,213	1,213
Mesh	12	1	1,316	1,316
Torus	12	1	1,399	1,399
Mesh	2	6	1,803	1,803
Torus	2	6	2,707	2,707
Mesh	3	4	1,920	1,920
Torus	3	4	2,707	2,707
Mesh mit ext. NI	3	7	1,000	1,000
Mesh mit ext. NI	4	6	1,430	1,430
Mesh mit ext. NI	5	5	1,574	1,574

Tabelle 6.2: Für MPEG4 untersuchte NoC-Topologien mit normierter Fläche und normierten Mapping-Kosten

Topologie	**X-Dim**	**Y-Dim**	**Gewichtete Hops, normiert**	**$A_{NoC,normiert}$**
Ring	12	1	1,932	1,399
Lineare Anordnung	12	1	1,896	1,213
Mesh	12	1	2,083	1,316
Torus	12	1	1,932	1,399
Mesh	2	6	1,039	1,803
Torus	2	6	1,000	2,707
Mesh	3	4	1,359	1,920
Torus	3	4	1,200	2,707
Mesh mit ext. NI	3	7	1,221	1,000
Mesh mit ext. NI	4	6	1,124	1,430
Mesh mit ext. NI	5	5	1,113	1,574

Tabelle 6.3: Für VOPD untersuchte NoC-Topologien mit normierter Fläche und normierten Mapping-Kosten

Topologie	X-Dim	Y-Dim	Gewichtete Hops, normiert	$A_{NoC,normiert}$
Ring	12	1	1,160	1,399
Lineare Anordnung	12	1	1,705	1,213
Mesh	12	1	1,874	1,316
Torus	12	1	1,160	1,399
Mesh	2	6	1,384	1,803
Torus	2	6	1,213	2,707
Mesh	3	4	1,161	1,920
Torus	3	4	1,000	2,707
Mesh mit ext. NI	3	7	1,345	1,000
Mesh mit ext. NI	4	6	1,165	1,430
Mesh mit ext. NI	5	5	1,078	1,574

6.1.2 Iteration 2 – Vermittlungstechnik

Bei der zweiten Iteration des Design-Flows wird die Vermittlungstechnik (VT) untersucht. Dazu wird für jedes Pareto-optimale NoC der ersten Iteration eine NoC-Variante mit den verschiedenen Vermittlungstechniken Leitungsvermittlung (LV) und Wormhole-Routing (WH), erzeugt. Für diese wird eine statische Timing-Analyse durchgeführt, um die maximal auftretende Latenz (ohne dynamische Effekte) zu bestimmen. In diesem Fall sind die Bewertungsparameter die normierte Fläche und die maximal auftretende Latenz.

Tabelle 6.4:Iteration 2, Vermittlungstechnik MPEG4 normierter Fläche und maximale Latenz

Topologie	Dimension	VT	$A_{NoC,normiert}$	Maximale Latenz
Mesh mit ext. NI	5x5	LV	1,574	22
Mesh mit ext. NI	5x5	WH	3,136	12
Mesh mit ext. NI	4x6	LV	1,430	27
Mesh mit ext. NI	4x6	WH	2,899	15
Mesh mit ext. NI	3x7	LV	1,000	27
Mesh mit ext. NI	3x7	WH	2,142	15
Torus	2x6	LV	2,707	27
Torus	2x6	WH	4,857	15
Mesh	2x6	LV	1,803	27
Mesh	2x6	WH	3,740	15

Tabelle 6.5:Iteration 2, Vermittlungstechnik MWD normierter Fläche und maximale Latenz

Topologie	Dimension	VT	$A_{NoC,normiert}$	Maximale Latenz
Mesh mit ext. NI	3x7	WH	2,142	15
Mesh mit ext. NI	3x7	LV	1,000	27
Mesh mit ext. NI	4x6	WH	2,899	12
Mesh mit ext. NI	4x6	LV	1,430	22

Tabelle 6.6:Iteration 2, Vermittlungstechnik VOPD normierter Fläche und maximale Latenz

Topologie	Dimension	VT	$A_{NoC,normiert}$	Maximale Latenz
Mesh mit ext. NI	5x5	LV	1,574	22
Mesh mit ext. NI	5x5	WH	3,136	12
Mesh mit ext. NI	4x6	LV	1,430	27
Mesh mit ext. NI	4x6	WH	2,899	15
Mesh mit ext. NI	3x7	LV	1,000	27
Mesh mit ext. NI	3x7	WH	2,142	15
Torus	2x6	LV	2,707	27
Torus	2x6	WH	4,857	15
Mesh	2x6	LV	1,803	27
Mesh	2x6	WH	3,740	15

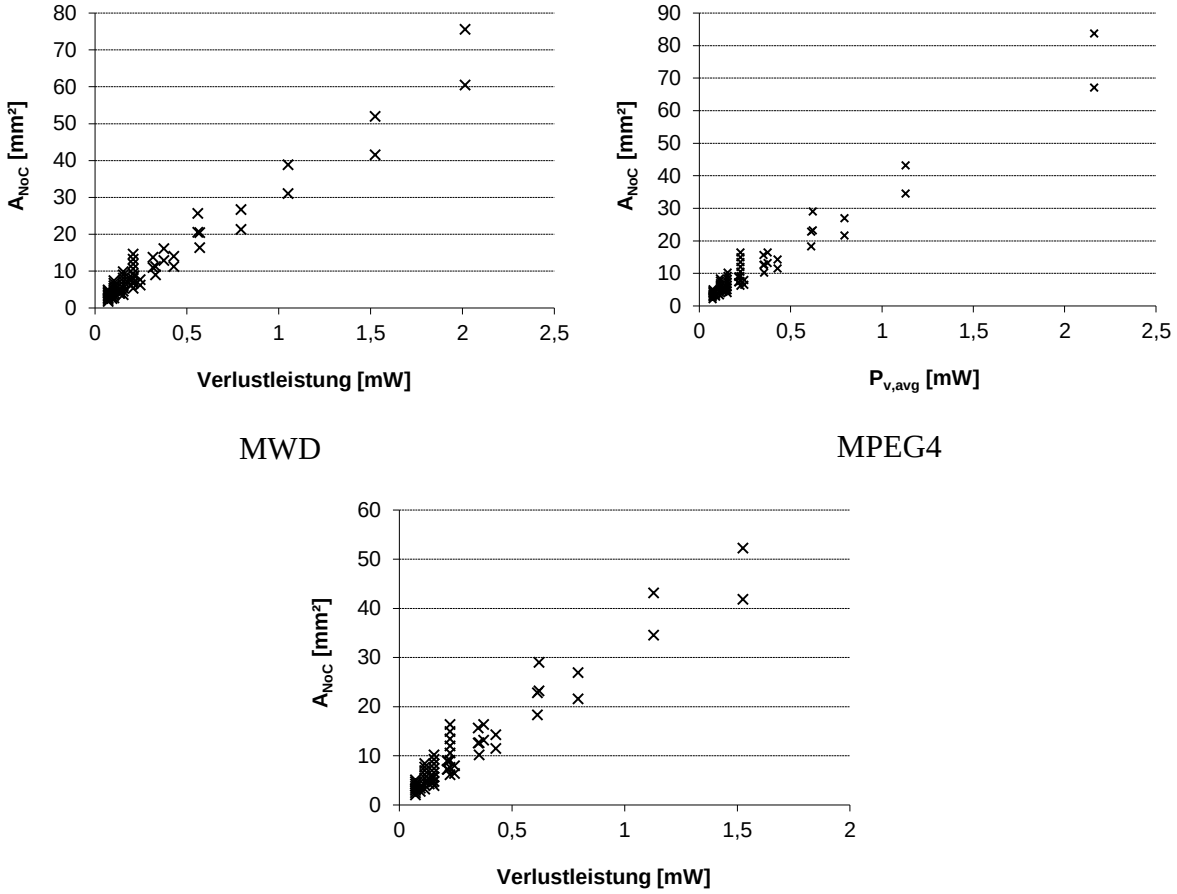
Die durch das Wormhole-Routing gegenüber der Leitungsvermittlung erzielte höhere Performance, die sich durch eine geringere Latenz ausdrückt, wird durch eine entsprechende größere Fläche des NoC erkauft.

6.1.3 Iteration 3 - Taktfrequenz und Wortbreite

Für die dritte Iteration, in der sowohl die Datenwortbreite als auch die Taktfrequenz des NoC festgelegt werden sollen, werden für die Pareto-optimalen NoC-Varianten VHDL-Beschreibungen der NoC erzeugt. Diese sind die Grundlage der FPGA-basierten NoC-Emulatoren. Mit diesen Emulatoren wurden verschiedene Experimente durchgeführt, um die Performance des NoCs für den jeweiligen Datenverkehr bestimmen zu können. Im Rahmen dieser Experimente wird der durch die Task-Graphen beschriebene Datenverkehr über das NoC übertragen und so die resultierende Performance des NoC bestimmt. Mit Hilfe der Modellfunktionen aus Kapitel 5 werden die Fläche und die von den Kommunikationsanforderungen abhängende Verlustleistung bestimmt.

Bei der Pareto-Analyse werden alle NoC berücksichtigt, bei denen die in den Task-Graphen gegebenen Deadlines, hier das Ende einer Periode, eingehalten werden. Die NoC werden bezüglich der Fläche (A_{NoC}) und durchschnittlicher Verlustleistung ($P_{V,NoC,Avg}$) untersucht.

Bei den Experimenten auf dem Emulator werden die jeweils zu übertragenden Datenmengen in Datenworte und die Zeit in Taktzyklen umgerechnet. Durch eine entsprechende Skalierung können so Experimente für verschiedene Datenwortbreiten und Taktfrequenzen auf einem Emulator durchgeführt werden. Die Taktfrequenz wurde hier zwischen 250 MHz und 2 GHz, die Datenwortbreite zwischen 16 und 256 Bit variiert. Es wurden dazu für die Wortbreiten 16, 32, 64, 128 und 256 Bit jeweils die Taktfrequenz ermittelt, bei der die Verlustleistung minimal ist und die Deadlines des Task-Graphen eingehalten werden. Diese Ergebnisse sind in Abbildung 6.5 dargestellt.



MWD MPEG4

VOPD

Abbildung 6.5: Ergebnisse der dritten Iteration für NoC, welche die Timing-Vorgaben erfüllen. Es wurde die Datenwortbreite und die Taktfrequenz variiert.

Die Performance-Evaluierung ergab, dass für alle drei Systeme ein NoC mit minimaler Wortbreite (16 Bit) und minimaler Taktfrequenz (200 MHz) eine ausreichende Performance bieten und dabei die geringsten Kosten verursacht.

6.1.4 Iteration 4 – Routing-Algorithmus

Der Routing-Algorithmus, der optimal zu den gegebenen Systemen passt, soll in dieser Iteration evaluiert werden. Die Iterationen eins bis drei haben ergeben, dass für alle drei Systeme ein NoC mit der in Abbildung 6.6 dargestellten Topologie (Mesh mit externen NI, Dimension 3x7) die geringsten Kosten bei ausreichender Performance verursacht. Bei dieser Topologie handelt es sich um eine eindimensionale Topologie. Daher gibt es für jede Datenübertragung nur einen möglichen Weg durch das NoC. Die für diese Topologie sinnvollen Routing-Algorithmen reduzieren sich deshalb auf das statische XY-Routing. Adaptive Routing-Algorithmen können hier nicht verwendet werden, da keine alternative Verbindung zwischen Quelle und Ziel der Datenübertragung existiert.

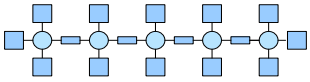


Abbildung 6.6: Topologie Mesh mit externen NI mit den Dimensionen 3x7

6.1.5 Pareto-Optimale NoC

In den folgenden Tabellen sind die Parameter der NoC aufgeführt, die ein NoC beschreiben, das für das jeweilige System Pareto-optimal ist. Die Kommunikationsanforderungen der Systeme an ein NoC sind in diesem Fall relativ gering. Für alle drei Systeme wurde hier ein NoC gefunden das die kleinste zulässige Datenwortbreite und Taktfrequenz besitzt. Die gewählte Topologie ist ein Mesh mit externen NI, da für diese Teilnehmerzahl die Anzahl der Routing-Switches pro Network-Interface minimal ist und die geringsten Kosten verursacht. Die Vermittlungstechnik ist Leitungsvermittlung. Für die drei hier betrachteten Systeme ist die Fläche des NoC identisch. Die Verlustleistung, die vom Datenverkehr abhängig ist, variiert zwischen 1,8 mW und 2,1 mW.

Tabelle 6.7: Pareto-optimale NoC für MPEG4 nach der 4. Iteration des Design-Flows

Topologie	**Dimension**	**VT**	**RA**	**Datenwort-breite**	**Takt-frequenz**	**A_{NoC} [mm²]**	**$P_{v,avg}$ [mW]**
Mesh mit externen NI	3x7	LV	XY	16	200 MHz	0,071	2,1

Tabelle 6.8: Pareto-optimale NoC für MWD nach der 4. Iteration des Design-Flows

Topologie	**Dimension**	**VT**	**RA**	**Datenwort-breite**	**Takt-frequenz**	**A_{NoC} [mm²]**	**$P_{v,avg}$ [mW]**
Mesh mit externen NI	3x7	LV	XY	16	200 MHz	0,071	1,8

Tabelle 6.9: Pareto-optimale NoC für VOPD nach der 4. Iteration des Design-Flows

Topologie	**Dimension**	**VT**	**RA**	**Datenwort-breite**	**Takt-frequenz**	**A_{NoC} [mm²]**	**$P_{v,avg}$ [mW]**
Mesh mit externen NI	3x7	LV	XY	16	200 MHz	0,071	2,08

6.1.6 Einfluss der Datenblockgröße auf die Performance von NoC

Ein wichtiger Parameter einer Datenübertragung ist neben der zu übertragenden Datenmenge die Granularität der zu übertragenden Daten. Diese beschreibt, für den Fall, dass die Daten nicht als ein monolithischer Block, sondern aufgeteilt auf mehrere Datenpakete übertragen werden, die Größe dieser Datenpakete. Die Größe der Datenpakete hängt von der Anwendung und von systemspezifischen Parametern wie zum Beispiel der Größe von lokalen Speichern einzelner Prozessoren ab. Der Einfluss der Granularität einer Datenübertragung auf die Performance eines NoCs wird im Folgenden untersucht.

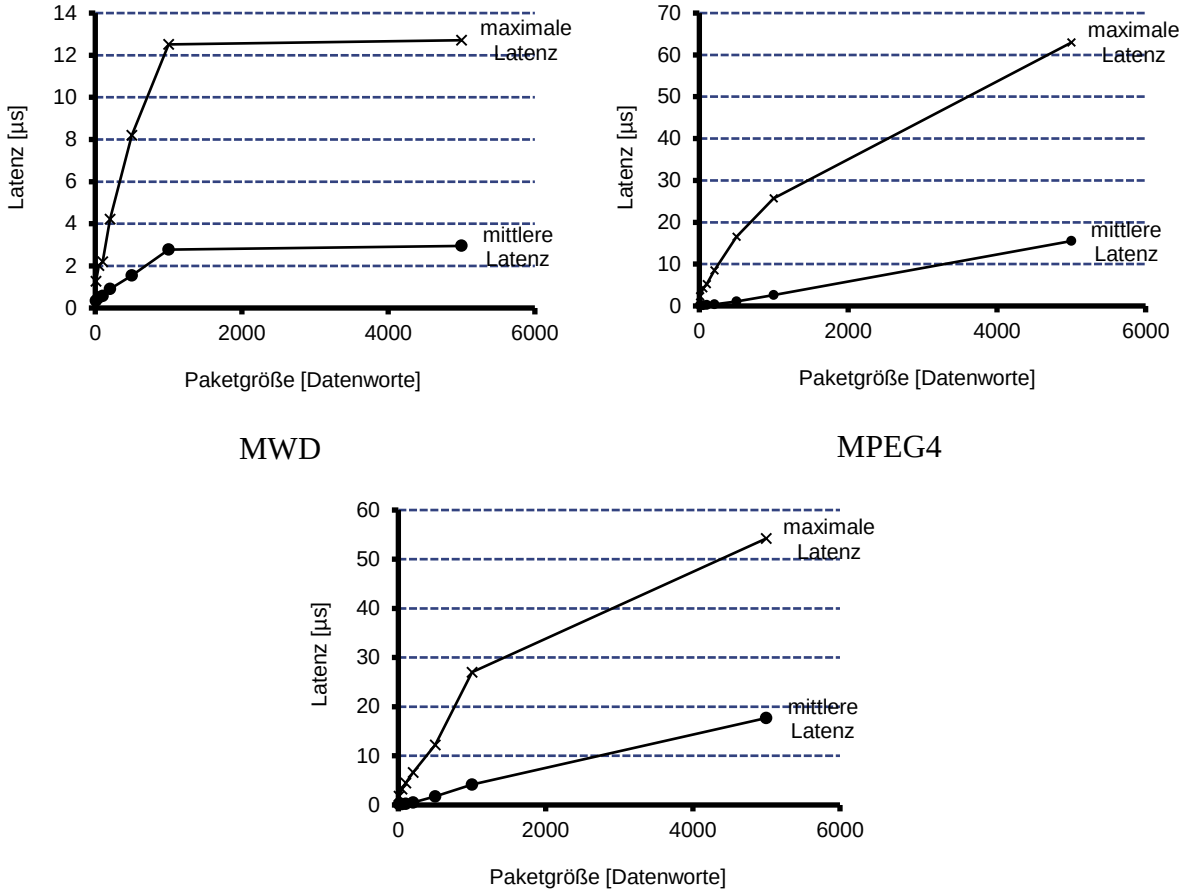


Abbildung 6.7: Einfluss der Datenpaketgröße auf die Performance, dargestellt durch die mittlere und maximale Latenz.

Um diesen Einfluss für das im Kapitel 6.1 ermittelten NoC für die Systeme zu ermitteln, wurden mehrere Experimente auf dem NoC durchgeführt. Es wurden die zu sendenden Daten in Datenpakete aufgeteilt, die über die Perioden verteilt gesendet wurden. Die Größe der Datenpakete wurde zwischen 20 Byte (10 Datenworte á 16 Bit) und 10.000 Byte (5000 Datenworte á 16 Bit) variiert. Die Performance wurde anhand der mittleren sowie der maximal auftretenden Latenz für das Pareto-optimale NoC für die drei betrachteten Systeme evaluiert. Die Ergebnisse dieser Untersuchungen sind in Abbildung 6.7 dargestellt. In allen Untersuchungen wurden die geforderten Deadlines eingehalten.

In diesen Beispielen ist die Performance für Datenverkehr, bei dem die Daten auf kleine Datenpakete aufgeteilt sind, höher als für Datenverkehr, der aus großen Datenpaketen besteht. Dies liegt daran, dass ein großes Datenpaket die für die Datenübertragung benötigten Netzwerk-Ressourcen länger belegt als ein kleines Datenpaket. Daher müssen andere Datenpakete, die ebenfalls eine der belegten Netzwerk-Ressourcen benötigen, länger warten, bis diese wieder zur Verfügung stehen.

Die hier untersuchten NoC verwenden Leitungsvermittlung als Vermittlungstechnik. Daher ist der Aufwand für den Verbindungsauf- und -abbau bei vielen, kleinen Datenpakten größer, als wenn lediglich ein großes Datenpaket übertragen wird. Bei NoC, die Paketvermittlungstechniken nutzen, ist dieser Aufwand geringer. Somit werden diese eine bessere Performance bei kleiner Datenblockgröße haben. Diese bessere Performance wird jedoch durch den höheren Aufwand erkauft, der z.B. durch benötigte Speicher verursacht wird (vgl. Kapitel 4.5.1).

6.1.7 Benchmark der Ergebnisse

In diesem Abschnitt wird ein Vergleich der erzielten Ergebnisse mit in bereits veröffentlichten Ergebnissen vorgenommen. Für die Systeme MWD, MPEG4 und VOPD wurden applikationsspezifische NoC in [106] vorgestellt. Dazu wurde die vielzitierte xPipes Entwurfsmethodik für NoC verwendet. Ein besonderes Augenmerk wurde bei dieser Untersuchung auf die Topologie des NoCs sowie das Potential von applikationsspezifischen Optimierungen der Topologie gelegt. Für das System VOPD wurde in [107] eine weitere NoC-Implementierung präsentiert. Hier werden sogenannte rekonfigurierbare Topologien (ReNoC) verwendet, um die Implementierungskosten zu minimieren.

Tabelle 6.10: Parameter der zu Vergleichenden NoC-Implementierungen

	Diese Arbeit	**xPipes** [106]	**ReNoC** [107]
Technologie	90 nm TSMC-Standardzellen	100 nm	90 nm
Taktfrequenz	200 MHz	500 MHz [111]	100 MHz
Wortbreite	16 Bit	32 Bit (1 Flit)	32 Bit
Topologie	hier: Mesh mit peripheren Network-Interfaces	Mesh und applikationsspezifische Topologien	Mesh und applikationsspezifische Topologien
Vermittlungstechnik	Leitungsvermittlung	Wormhole-Routing	Wormhole-Routing
Routing-Switch	4-Ports Routing im Routing-Switch	3-8 Ports, abhängig von Topologie Quellen-Routing Buffergröße: 16 - 22 Filts optional: CRC-Fehlerschutz	3-5 Ports 2 Virtual Channels Quellen-Routing Clock-Gating
Link	Länge: 1 mm	Länge abhängig vom Floorplan, hier ~1 mm	Länge: 1 mm

Der Vergleich erfolgt bezüglich der Implementierungskosten, die hier durch die für das NoC benötigte Fläche sowie die benötigte Verlustleistung ausgedrückt werden. Die Performance des NoC wird nur insoweit berücksichtigt, dass alle betrachteten NoC die Kommunikationsanforderungen erfüllen.

Die Bestimmung der Fläche und der Verlustleistung erfolgt bei allen Implementierungen mit Hilfe von Modellfunktionen für die Komponenten Routing-Switch und Network-Interface. In dieser Arbeit wurden die Modellfunktionen für einen 90 nm Technologie bestimmt. Dazu wurde mit Hilfe von Standardzellen das Layout der NoC-Komponenten erstellt und anschließend simuliert. Bei xPipes wurden die Modellfunktionen für Fläche aus einer Synthese mit Standardzellen für eine 100 nm Technologie bestimmt. Die Verlustleistung wurde mit Hilfe von ORION [110] modelliert. Bei der dritten NoC-Implementierung (ReNoC) wurden die Flächen- und Verlustleistungsmodellfunktionen aus der Synthese mit Standardzellen abgeleitet. Dazu wurde eine 90 nm Technologie verwendet.

Die Performance der NoC wurde in dieser Arbeit mit Hilfe von umfangreichen Experimenten auf einem FPGA-basierten Emulator verifiziert. Bei [106] wurde eine zyklengenaue Simulation eines SystemC-Modells durchgeführt. In [107] wurde lediglich ein Vergleich der angeforderten und der zur Verfügung stehenden Bandbreite durchgeführt.

In Abbildung 6.8 sind die Kosten der einzelnen NoC-Implementierungen für die drei Systeme aufgetragen. Die Rauten, die mit ‚EECS' bezeichnet sind, stehen für die Ergebnisse dieser Arbeit, die Vierecke mit der Bezeichnung ‚xPipes' repräsentieren die Ergebnisse aus [106] und die Vierecke mit der Beschriftung ‚ReNoC' stehen für die Ergebnisse aus [107]. Um einen fairen Vergleich der NoC-Implementierungen vornehmen zu können, wurde die Verlustleistung und die Fläche auf eine 90 nm Technologie skaliert. Dazu wurden die die Skalierungsregeln aus [18] angewendet. Die Parameter der NoC-Implementierungen bzw. der NoC-Komponenten ist in Tabelle 6.10 angegeben.

Für das System MWD wurden in [106] zwei Topologien untersucht: eine Mesh-Topologie (‚xPipes, Mesh') sowie eine applikationsspezifisch optimierte Topologie (‚xPipes, Custom'). Bei dieser applikationsspezifischen Optimierung wurden einzelne NoC-Komponenten, die für diese Anwendung nicht benötigt wurden, aus dem NoC eliminiert, wodurch sich die Implementierungskosten reduzieren. Allerdings können nicht mehr alle Teilnehmer des NoC untereinander (direkt) miteinander kommunizieren. Somit wird durch diese Optimierung die Flexibilität, andere Anwendungen auf diesem SoC zu realisieren, eingeschränkt. Die Fläche der NoC-Implementierung dieser Arbeit entspricht mit 0,07 mm² lediglich 7 % der Fläche der Mesh-Implementierung aus [106] (0,98 mm²) und 88 % der Fläche der unflexiblen applikationsspezifisch optimierten Topologie (0,08mm²). Die Verlustleitung beträgt mit 1,8 mW ca. 8 % der Mesh-Implementierung (21,0 mW) und 28 % der Implementierung der optimierten Topologie (6,3 mW).

Für das System MPEG4 ergeben sich qualitativ ähnliche Ergebnisse. Die Fläche (Verlustleistung) der Implementierung dieser Arbeit entspricht 7 % (2%) der Fläche einer Implementierung mit Mesh Topologie (‚xPipes, Mesh') und 10 % (2 %) bzw. 12 % (3 %)

einer Implementierung mit applikationsspezifischer Optimierung (‚xPipes, Custom1' und ‚xPipes, Custom2').

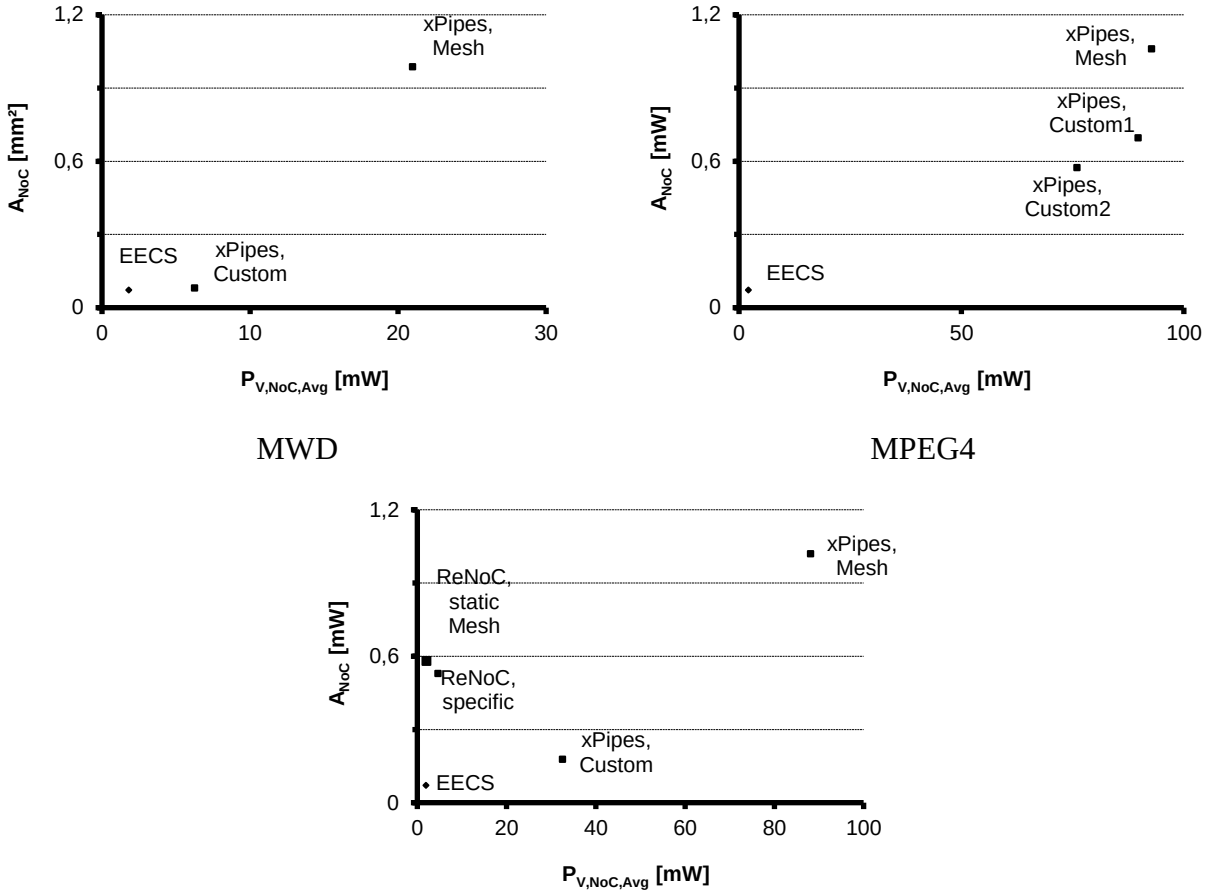


Abbildung 6.8: Vergleich der Implementierungskosten (skaliert auf 90 nm Technologie) verschiedener NoC für Anwendungen MWD, MPEG4 und VOPD

Der Vergleich des letzten Systems VOPD beruht ebenfalls auf einem Vergleich zwischen den Implementierungen aus dieser Arbeit und den Implementierungen aus [106]. Auch hier ist die Fläche und die Verlustleistung der Implementierung aus dieser Arbeit nur ein Bruchteil der Fläche und Verlustleistung der Implementierung ‚xPipes, Mesh' (7 % und 2 %). Die applikationsspezifisch optimierte Variante hat eine deutlich geringere Fläche und Verlustleistung gegenüber der Implementierung mit Mesh Topologie, ist verglichen mit der Implementierung dieser Arbeit jedoch immer noch mehr als doppelt so groß und benötigt knapp die 17-fache Verlustleistung. Die NoC-Implementierungen aus [107] haben eine ähnliche Verlustleistung wie die Implementierung dieser Arbeit haben dabei jedoch eine deutlich größere Fläche.

Die geringere Fläche der Implementierung dieser Arbeit mit den Vergleichs-Implementierungen lässt sich durch mehrere Faktoren erklären: bei der Implementierung

dieser Arbeit wird Leitungsvermittlung anstelle von Paket-Vermittlung verwendet. Dadurch kann auf Speicher in den Routing-Switches bzw. den Network-Interfaces verzichtet werden, die einen Großteil der Fläche eines NoC ausmachen können. Darüber hinaus wurde das verwendete Übertragungs-Protokoll für geringen Hardware-Aufwand optimiert. Verglichen mit den Implementierungen aus [106] ist ein weiterer Aspekt, der eine geringere Fläche verursacht die Synthese auf Basis einer VHDL-Beschreibung wie sie in dieser Arbeit verwendet wurde, gegenüber einer Synthese auf Basis einer abstrakteren SystemC-Beschreibung. Die trotz relativ großer Fläche geringe Verlustleistung der Implementierungen aus [107] wird durch die Verwendung von Clock-Gating erreicht. Im Ganzen bestätigen die Ergebnisse dieses Kapitels die Ergebnisse des Vergleichs einzelner Routing-Switches, der bereits in Kapitel 4.7 vorgestellt wurde.

6.2 Anwendung der Entwurfsmethodik für synthetische Kommunikations-anforderungen

Die im vorherigen Abschnitt untersuchten realen Systeme stellten lediglich geringe Kommunikationsanforderungen an das NoC. Aus diesem Grund werden in diesem Abschnitt synthetische Kommunikationsanforderungen mit mehr Teilnehmern und höherem Datendurchsatz betrachtet.

Die Task-Graphen, welche die Kommunikationsanforderungen von Systemen beschreiben, lassen sich grob in zwei Klassen unterteilen. In der einen Klasse gibt es eine oder mehrere zentrale funktionale Einheiten, die mit vielen anderen funktionalen Einheiten kommunizieren. Ein Beispiel für eine solche funktionale Einheit wären gemeinsame, zentrale Speicher. Bei diesen Task-Graphen ist der die Performance limitierende Faktor nicht das NoC, sondern die häufig belegten Network-Interfaces der zentralen funktionalen Einheiten. Die zweite Klasse von Task-Graphen ist dadurch gekennzeichnet, dass die einzelnen funktionalen Einheiten nur mit jeweils wenigen funktionalen Einheiten direkt kommunizieren. Die Ausführungszeiten der einzelnen Tasks auf den funktionalen Einheiten werden in diesem Beispiel vernachlässigt, da sie keinen direkten Einfluss auf die Kommunikationsanforderungen haben.

Im Folgenden wird jeweils ein exemplarischer Task-Graph der jeweiligen Klasse mit Hilfe der in [62] vorgestellten Software erzeugt. Dabei werden für einstellbare Rahmenbedingungen zufällige Task-Graphen gebildet. Der Task-Graph TG_1 steht hier für die Klasse der Task-Graphen, die funktionale Einheiten mit vielen ein- und ausgehenden Kanten besitzen, TG_2 gehört zur zweiten Klasse von Task-Graphen. Der erste Task-Graph TG_1 besteht aus 16 Tasks, der zweite Graph TG_2 aus 15. Die beiden Graphen sind in Abbildung 6.2 abgebildet. Um diese Graphen zu erzeugen, wurde die Anzahl der maximal in einen Knoten einlaufenden Kanten bzw. aus einem Knoten auslaufenden Kanten auf jeweils 10 bei TG_1 und 4 bei TG_2 begrenzt.

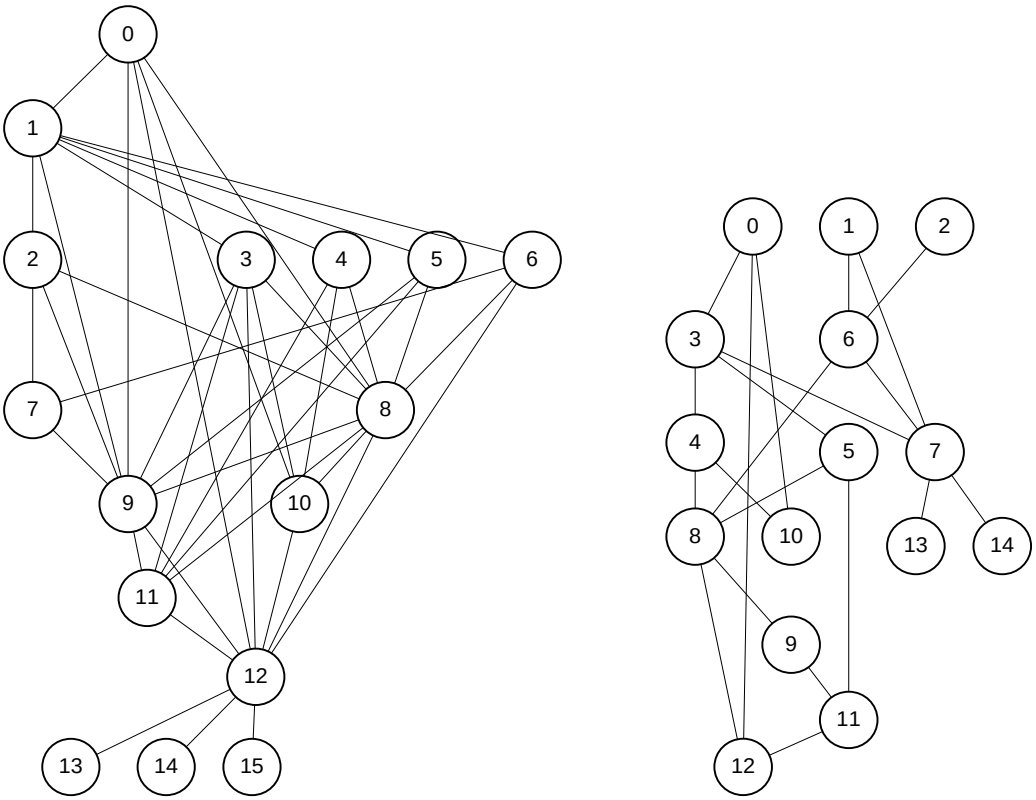


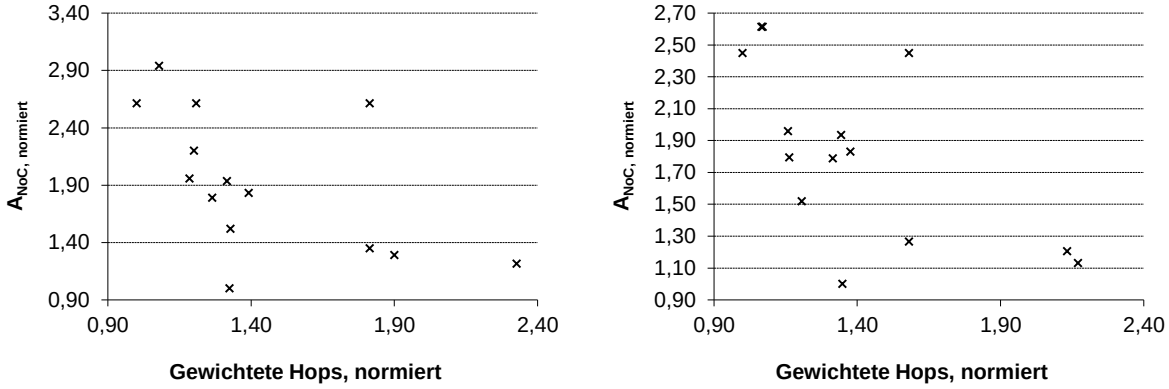
Abbildung 6.9: Task-Graphen TG_1 (links) und TG_2 (rechts)

Bei beiden Graphen handelt es sich um nicht zyklische Graphen, die nach einer jeweils gegebenen Periodendauer neu gestartet werden. Die Gewichte der Kanten repräsentieren hier die Datenmenge, die pro Periode übertragen werden soll. Diese sind im Anhang C in den Tabelle C.5 und Tabelle C.6 angegeben. Die Periodendauer entspricht in diesen Beispielen gleichfalls auch der Deadline, die den jeweiligen End-Knoten der Graphen zugeordnet sind.

6.2.1 Iteration 1 – Topologie

Auch für diese synthetischen Kommunikationsanforderungen werden in der ersten Iteration die Topologie und die Dimensionen des NoC bestimmt. Die Bewertungsmaße sind wie im vorhergehenden Beispiel die Fläche des NoCs als Kostenmaß sowie die mit der jeweils übertragenen Datenmenge gewichtete Anzahl der Hops als Performance-Maß (vgl. Kapitel 6.1.1).

Neben der gewichteten Anzahl der Hops als Bewertungsmaß können auch weitere Maße, wie beispielsweise eine möglichst geringe maximale Auslastung einzelner NoC-Komponenten verwendet werden. Ein solches Bewertungsmaß würde die Verlustleistung nicht berücksichtigen, allerdings würde durch eine geringere Anzahl von potenziellen Kollisionen eine höhere Performance erreicht werden.



(a) (b)

Abbildung 6.10: Ergebnisse der ersten Iteration des Design-Flows für TG_1 (a) und TG_2 (b) für unterschiedliche Topologien und Dimensionen

Für eine bessere und übersichtlichere Darstellung der Daten wird die von den NoC benötigte Fläche auf die Fläche des kleinsten NoCs normiert. Die Kosten, die durch die Abbildung der jeweiligen Task-Graphen auf das NoC entstehen, sind jeweils auf die jeweils geringsten entstehenden Kosten normiert. Die Ergebnisse dieser ersten Iteration sind in den Diagrammen in Abbildung 6.10 dargestellt.

Die bezüglich der Fläche und Mapping-Kosten Pareto-optimalen NoC-Topologien sind in Tabelle 6.1 und Tabelle 6.2 angegeben. Diese bilden die Basis für die Parametervariation der nächsten Iteration.

Tabelle 6.11: Für die Task-Graphen TG_1 untersuchte NoC-Topologien mit normierter Fläche und normierten Mapping-Kosten

Topologie	**Dimension**	**$A_{NoC,normiert}$**	**Gewichtete Hops, normiert**
Mesh mit externen NI	9x3	1,00	1,32
Mesh	8x2	1,79	1,26
Mesh	4x4	1,96	1,18
Torus	4x4	2,61	1,00

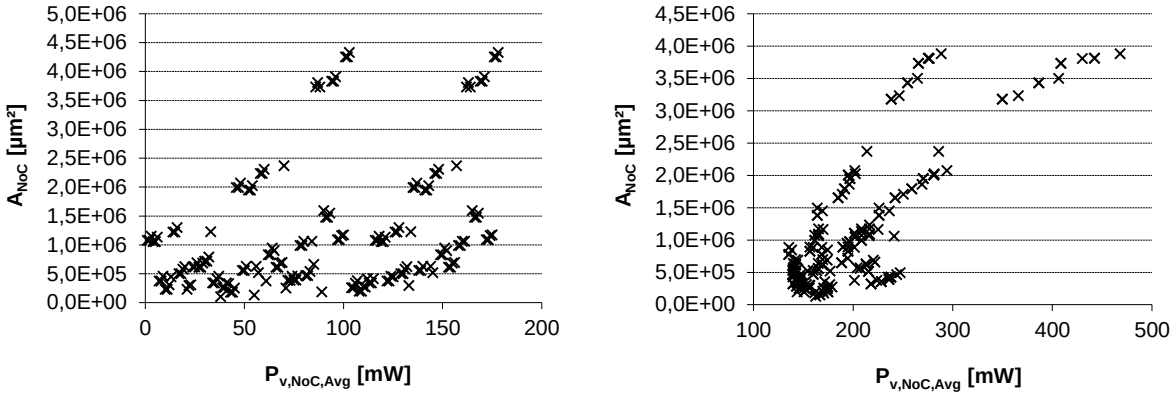
Tabelle 6.12: Für die Task-Graphen TG_2 untersuchte NoC-Topologien mit normierter Fläche und normierten Mapping-Kosten

Topologie	**Dimension**	**$A_{NoC,normiert}$**	**Gewichtete Hops, normiert**
Mesh	5x3	1,80	1,16
Mesh	4x4	1,96	1,16
Torus	5x3	2,45	1,00
Mesh mit externen NI	9x3	1,00	1,35
Mesh mit externen NI	4x8	1,52	1,21

6.2.2 Iteration 2 – Vermittlungstechnik, Routing-Algorithmus, Taktfrequenz und Wortbreite

Bei der zweiten Iteration des Design-Flows werden zunächst die Vermittlungstechnik (VT) und der verwendete Routing-Algorithmus (RA) variiert. Für die neu gewählten Parameter-Kombinationen werden entsprechende VHDL-Beschreibungen der NoC erzeugt, mit deren Hilfe FPGA-basierte NoC-Emulatoren synthetisiert werden. Mit diesen Emulatoren wurden verschiedene Experimente durchgeführt, um die Performance des NoCs für den jeweiligen Task-Graphen bestimmen zu können. Im Rahmen dieser Experimente wird der durch die Task-Graphen beschriebene Datenverkehr über das NoC übertragen und so die resultierende Performance des NoC bestimmt.

Bei den Experimenten auf den Emulatoren werden die jeweils zu übertragenden Datenmengen in Datenworte und die Zeit in Taktzyklen umgerechnet. Durch eine entsprechende Skalierung können so Experimente für verschiedene Datenwortbreiten und Taktfrequenzen durchgeführt werden. In diesem Fall wurde die Taktfrequenz zwischen 250 MHz und 2 GHz und die Datenwortbreite zwischen 16 und 256 Bit variiert. Mit Hilfe der Modellfunktionen werden die Fläche und die Task-Graph-spezifische Verlustleistung bestimmt.



(a) (b)

Abbildung 6.11: Ergebnisse der zweiten Iteration des Design-Flows für TG_1 (a) und TG_2 (b) für NoC, welche die Timing-Vorgaben erfüllen. Es wurde die Vermittlungstechnik sowie der Routing-Algorithmus variiert.

Bei der anschließenden Pareto-Analyse werden alle NoC berücksichtigt, bei denen die in den Task-Graphen gegebenen Deadlines eingehalten wurden. Sie wurden bezüglich der Fläche (A_{NoC}) und durchschnittlicher Verlustleistung des NoC ($P_{V,NoC,Avg}$) untersucht. In den konkreten Fällen wurden für den Task-Graphen TG_1 23 und für den Task-Graphen TG_2 30 Emulatoren synthetisiert, auf denen insgesamt 254 bzw. 474 Versuche durchgeführt wurden.

Die resultierenden, Pareto-optimalen NoC sind in Tabelle 6.13 für den Task-Graphen TG_1 und in Tabelle 6.14 für TG_2 aufgeführt. Als Implementierungsvariante bei den Modellen wurde für den gegebenenfalls benötigten Speicher die in Kapitel 4.4.4 vorgestellte Implementierung mit speziellen Standardzellen und für die Implementierung der Routing-Switches die Variante mit physikalisch optimierten Kernkomponenten gewählt (vgl. Kapitel 4.5.4). Die Verlustleistung wurde mit Hilfe der statischen Performance-Analyse, sowie den Modellfunktionen bestimmt. Die Fläche wurde ebenfalls mit Hilfe der Modellfunktionen bestimmt.

Tabelle 6.13: Pareto-optimale NoC für TG_1 nach der 2. Iteration des Design-Flows

Topologie	**Dimension**	**VT**	**RA**	**Datenwort-breite**	**Takt-frequenz**	**A_{NoC} [µm²]**	**$P_{v,avg}$ [mW]**
Mesh mit externen NI	3x9	LV	XY	16	1250 MHz	98943	15
Mesh mit externen NI	3x9	LV	XY	32	750 MHz	126914	13
Mesh mit externen NI	3x9	LV	XY	128	250 MHz	294740	12

Für den TG_1 hat sich die Topologie Mesh mit externen NI als optimal erwiesen. Da auf dem NoC eine relativ geringe Auslastung herrscht, ist diese Topologie, bei der das Verhältnis zwischen Routing-Switches bzw. Links und Network-Interfaces maximal ist, flächen-minimal bei ausreichender Performance. Die lineare Struktur dieser Topologie bewirkt, dass nur eine einzige Verbindung zwischen zwei Network-Interfaces aufgebaut werden kann. Deshalb ist hier lediglich das einfache XY-Routing sinnvoll anzuwenden.

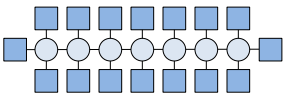


Abbildung 6.12: (Pareto-)Optimale Topologie (Mesh mit externen NI, 3x9) eines NoC für Task-Graphen TG_1

Die Ergebnisse in Tabelle 6.13 zeigen, dass sich die Fläche gegen die durchschnittlich benötigte Verlustleistung abtauschen lässt, indem Wortbreite und Taktfrequenz variiert werden. Bei einer geringen Wortbreite ist die Fläche relativ klein. Jedoch muss, damit die Timing-Vorgaben erfüllt werden, die Taktfrequenz gesteigert werden. Dies resultiert in einer höheren Verlustleistung. Jedoch ist in diesem Beispiel der Einfluss auf die Verlustleistung relativ gering, da durch die geringe Auslastung des NoCs die Verlustleistung von der Verlustleistung im Ruhezustand dominiert wird.

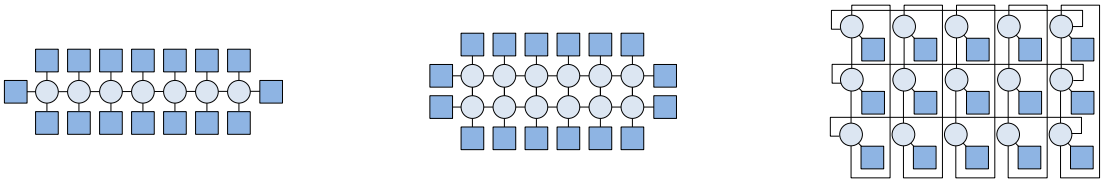
Bei Taktfrequenzen, die größer als 500 MHz sind, muss überprüft werden, inwieweit diese Timing-Vorgaben erreicht werden können und ob gegebenenfalls Treiber aufgeweitet und weitere Pipeline-Register eingefügt werden müssen. Beide Maßnahmen führen zu einer

Vergrößerung der benötigten Fläche wie auch der durchschnittlichen Verlustleistung. Durch zusätzliche Register kann sich außerdem die Performance des NoC verschlechtern. Dennoch kann diese Abschätzung aufgrund der Modellfunktionen Aufschluss darüber liefern, ob eine evtl. notwendige Optimierung für hohe Taktfrequenzen sinnvoll sein kann.

Tabelle 6.14: Pareto-optimale NoC für TG_2 nach der 2. Iteration des Design-Flows

Topologie	**Dimension**	**VT**	**RA**	**Daten-wortbreite**	**Takt-frequenz**	**A_{NoC} [µm²]**	**$P_{v,avg}$ [mW]**
Mesh mit externen NI	3x9	LV	XY	32	1500 MHz	126914	163
Mesh mit externen NI	3x9	LV	XY	64	1000 MHz	182856	162
Mesh mit externen NI	4x8	LV	A-XY	32	750 MHz	195439	144
Mesh mit externen NI	4x8	LV	XY	32	750 MHz	195439	144
Mesh mit externen NI	4x8	LV	XY	64	500 MHz	285634	143
Mesh mit externen NI	4x8	LV	A-XY	64	500 MHz	285634	143
Torus	3x5	LV	XY	32	750 MHz	318350	140
Torus	3x5	LV	A-XY	32	750 MHz	318350	140
Mesh mit externen NI	4x8	LV	XY	128	250 MHz	466025	139
Mesh mit externen NI	4x8	LV	A-XY	128	250 MHz	466025	139
Torus	3x5	LV	XY	128	250 MHz	773927	135
Torus	3x5	LV	A-XY	128	250 MHz	773927	135

Die für diese Task-Graphen optimale Vermittlungstechnik ist Leitungsvermittlung. Der Performance-Gewinn für den gegebenen Task-Graphen bei Wormhole-Routing ist nicht groß genug, um die zusätzlichen durch diese Vermittlungstechnik entstehenden Kosten zu kompensieren.



Mesh mit externen NI 3x9 *Mesh mit externen NI 3x9* *Torus 3x5*

Abbildung 6.13: Pareto-optimale Topologien eines NoC für Task-Graphen TG_2

Durch die höhere Auslastung des NoC sind für den TG_2 auch weitere Topologien interessant, obwohl sie eine größere Fläche beanspruchen (vgl. Tabelle 6.14). Neben der Mesh-Topologie mit externen Network-Interfaces mit einer oder zwei Reihen von Routing-Switches ist auch die Torus Topologie Pareto-optimal.

Wie zuvor diskutiert, ist auch bei diesem Task-Graphen die Leitungsvermittlung effizienter als das Wormhole-Routing. Ebenfalls lässt sich die Fläche gegen Verlustleistung abtauschen. Auch hier ist der Einfluss auf die Fläche (zwischen 126 und 773 µm²) größer als auf die Verlustleistung (zwischen 135 und 163 mW). Da für diesen Task-Graphen eine größere Datenmenge pro Periode übertragen werden muss, ist die Verlustleistung auch deutlich größer als bei TG_1.

Da bei den hier verwendeten Topologien potentiell mehrere minimale Wege zwischen zwei Network-Interfaces existieren, wurden auch adaptive Routing-Algorithmen untersucht. Der Mehraufwand für komplexere Routing-Algorithmen erzielt jedoch nur eine geringe Performance-Verbesserung, so dass hier die einfachen Algorithmen, welche die geringsten Kosten verursachen, verwendet werden. Der Unterschied bei den Implementierungskosten zwischen dem XY-Routing und dem adaptiven XY-Routing ist so gering, dass er von den Modellfunktionen für die verwendete Technologie nicht erfasst wird. Daher sind in diesem Beispiel sowohl NoC mit einfachem XY-Routing wie auch mit adaptiven XY-Routing Pareto-optimal. Die leichte Performance-Verbesserung, die durch das adaptive Routing erzielt werden kann, wurde bei dieser Pareto-Untersuchung nicht berücksichtigt.

6.3 Bewertung

Die Ergebnisse haben gezeigt, dass je nach Anwendung (zufälliger Datenverkehr und applikationsspezifischer Datenverkehr, der durch Task-Graphen repräsentiert wird) und Optimierungsziel (Fläche, Verlustleistung oder eine Kombination aus beiden Zielen) unterschiedliche NoC diese Ziele optimal erfüllen. Eine systematische Untersuchung des Entwurfsraums für unterschiedliche Anwendungen bzw. Anwendungsklassen ist aufgrund der Unterschiede bezogen auf die Implementierungskosten wie auch auf die erzielbare Performance unerlässlich. Für eine solche Untersuchung ist ein automatisierter Design-Flow notwendig, der mit möglichst geringem zeitlichem Aufwand eine entsprechende Parameterwahl ermöglicht und das NoC implementiert, oder zumindest die Grundlagen für eine optimiert Implementierung schafft. Mit dem erarbeiteten und implementierten Design-Flow ist eine solche systematische Erforschung des Entwurfsraumes sowie eine automatisierte Implementierung des Pareto-optimalen NoCs möglich.

7 Anwendungsbeispiel HOG Algorithmus auf dem Xeon Phi Multicore Prozessor

Die Praktikabilität der in den vorherigen Kapiteln vorgestellten Methode wird in diesem Kapitel anhand eines realen Beispiels verifiziert: Ein Algorithmus wird auf einem Multicore Prozessor (Device Under Test, DUT) implementiert, ausgeführt und vermessen. Die Kommunikationsarchitektur des Multicore-Prozessors ist bereits ein NoC. Im Rahmen von Benchmarks wird die Leistungsfähigkeit dieses NoC untersucht und der Datenverkehr sowie das NoC modelliert. Mit diesen Beschreibungen wird das NoC auf einem FPGA emuliert und die Ergebnisse der Emulation mit den Ergebnissen der Benchmarks auf dem DUT verglichen. Somit lassen sich die Modelle des NoC und des Datenverkehrs auf dem FPGA-basierten Emulator verifizieren. In einem zweiten Schritt wird untersucht, ob alternative NoC-Mehrwerte generieren können.

Das Vorgehen ist in Abbildung 7.1 dargestellt. Dazu wird ein Algorithmus auf einem existierenden Multi-Core Prozessor (Device Under Test) ausgeführt. Durch Benchmarks wird das Task-Mapping ermittelt und die Task-Graphen erstellt, die den Datenverkehr beschreiben. Zusätzlich wird über Performance Counter die Leistungsfähigkeit der Kommunikationsarchitektur auf dem Device Under Test ermittelt.

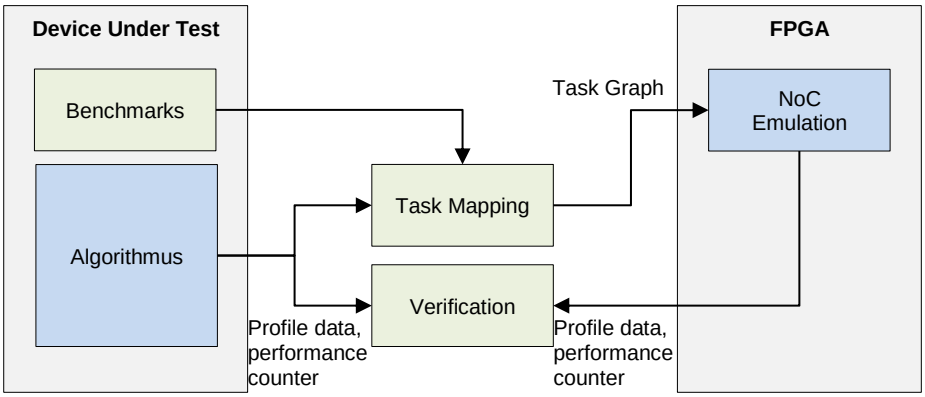


Abbildung 7.1: Vorgehen zur Verifikation der NoC-Simulationsergebnisse

Im nächsten Schritt wird die Kommunikations-Architektur des Device Under Test modelliert und auf dem FPGA nachgebildet. In Experimenten werden die Task-Graphen auf dem FPGA ausgeführt und die Ergebnisse der Simulation können in dem Schritt Verification mit den realen Performance Messwerten verglichen werden. Abschließend werden die Vorteile alternativer Kommunikations-Architekturen und NoC für dieses Anwendungsbeispiel untersucht und werden aufgezeigt.

Für Fahrerassistenz-Systeme werden immer komplexere Algorithmen entwickelt, die entsprechend hohe Anforderungen an die ausführende Hardware haben, sowohl hinsichtlich

der Rechenleistung wie auch des Arbeitsspeichers. Häufig lassen sich diese Algorithmen theoretisch gut parallelisieren, um die Ausführung in Echtzeit zu realisieren. Many-Core Architekturen sind somit eine gute Basis, diese Algorithmen effizient auszuführen. Aufgrund der hohen Kommunikationsanforderungen zwischen den (parallelen) Tasks ergeben sich entsprechend hohe Anforderungen an die Kommunikationsarchitektur.

In diesem Kapitel wird eine parallele Implementierung des Histogram of Oriented Gradients Algorithmus (HOG) für die kamerabasierte Erkennung von Fußgängern [112] auf dem Intel Xeon Phi Prozessor (Knights Corner) untersucht. In [113] wurde aufgezeigt, dass sich bei einer parallelen Implementierung des HOG Algorithmus auf dem Xeon Phi Performance Einbußen gegenüber der theoretisch erreichbaren Performance ergeben. Ein Grund für diese Einbußen können durch die Kommunikation zwischen den Prozessorkernen erklärt werden. Daher wurde die Datenkommunikation genauer untersucht.

Die Abbildung des Datenverkehrs in Task-Graphen wurde dann genutzt, um auf dem in Kapitel 3.6.1.2 vorgestellten NoC-Emulator den Ring-Bus des Prozessors nachzubilden und den Datenverkehr zu simulieren. Anhand des Vergleichs der Simulationsergebnisse mit der realen Ausführung auf dem Prozessor konnte die Genauigkeit der Analyse für NoC gezeigt werden. Anschließend wurde das Optimierungspotenzial durch alternative NoC für den HOG-Algorithmus untersucht. Dieser Ergebnisse wurden bereits in [114] veröffentlicht.

7.1 Beschreibung HOG-Algorithmus

Um aus Kamerabildern von Fahrassistenz Systemen in Autos Fußgänger erkennen zu können, werden häufige Machine Learning Algorithmen, wie Support-Vector-Machines (SVM) [115] oder neuronale Netze [116], verwendet. Diese benötigen Merkmale, die aus den Kamerabildern extrahiert werden.

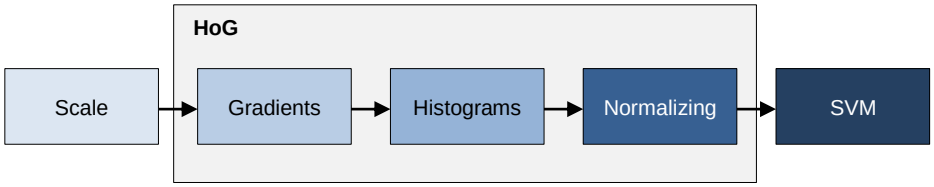


Abbildung 7.2:Pipeline des HOG-Algorithmus

Dazu wird in diesem Beispiel der HOG-Algorithmus verwendet, wie er in [112] vorgestellt wurde. Abbildung 7.2 zeigt die einzelnen Verarbeitungsschritte: Zunächst wird das Bild skaliert (Scale). Dann wird ein sogenanntes Detection Window (DW) über das skalierte Bild geschoben. Für jede Position des DW wird ein Vektor mit Merkmalen (Feature Descriptor) berechnet, der dann von der SVM einer Klasse zugeordnet wird.

Innerhalb des DW wird zunächst für jedes Pixel der Gradient (Gradients) berechnet. Die Pixel werden in Zellen zu 8x8 Pixel zusammengefasst. Für jede der Zellen wird ein Zell-Histogramm (Histograms) gebildet. Diese Zellen werden wiederum zu Blöcken von 2x2 Zellen zusammengefasst, welche dann abschließend zu einem Vektor der Länge 1 normalisiert werden.

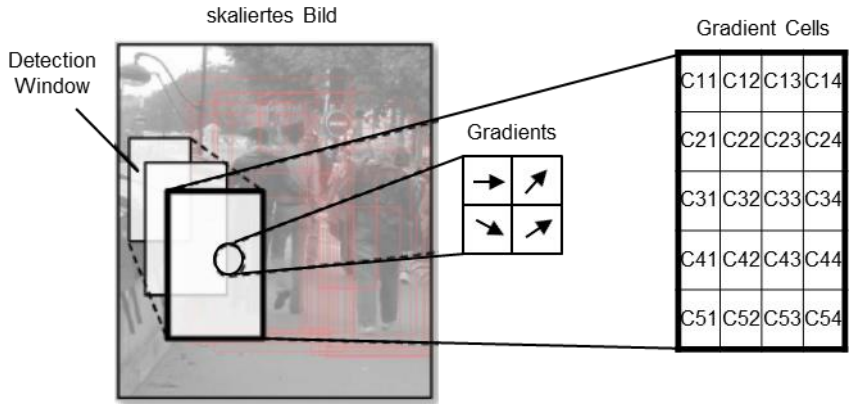


Abbildung 7.3:Extraktion von Merkmalen mit dem HOG-Algorithmus

Im Weiteren werden die rechenintensiven Verarbeitungsschritte im grauen gestrichelten Kasten HOG (Gradient, Histograms und Normalizing) in Abbildung 7.2 betrachtet.

Da Fußgänger in unterschiedlichen Größen in den Kamerabildern erscheinen können, wird das Bild in vier unterschiedlichen Skalierungen analysiert. Dabei wird mit dem DW jeweils das komplette Bild abgetastet. Durch die Skalierung des Bildes anstelle des DW verändert sich die Länge des Merkmal Vektors nicht, so dass immer die gleiche SVM genutzt werden kann.

7.1.1 Parallele Implementierung

Die hier verwendete Implementierung des HOG Algorithmus bietet zwei Möglichkeiten der Parallelisierung: zum einen kann die komplette Pipeline, wie sie in Abbildung 7.4 gezeigt wird, für die einzelnen Skalierungsstufen Parallel ausgeführt werden. Hier spricht man von einer Functional Decomposition. Bei der zweiten Möglichkeit der Parallelisierung werden die einzelnen Stufen des Algorithmus weiter in einzelnen Sub-Tasks aufgeteilt, die dann parallel ausgeführt werden. Dies wird als Data Decomposition bezeichnet. In diese Arbeit wurden beide Ansätze kombiniert.

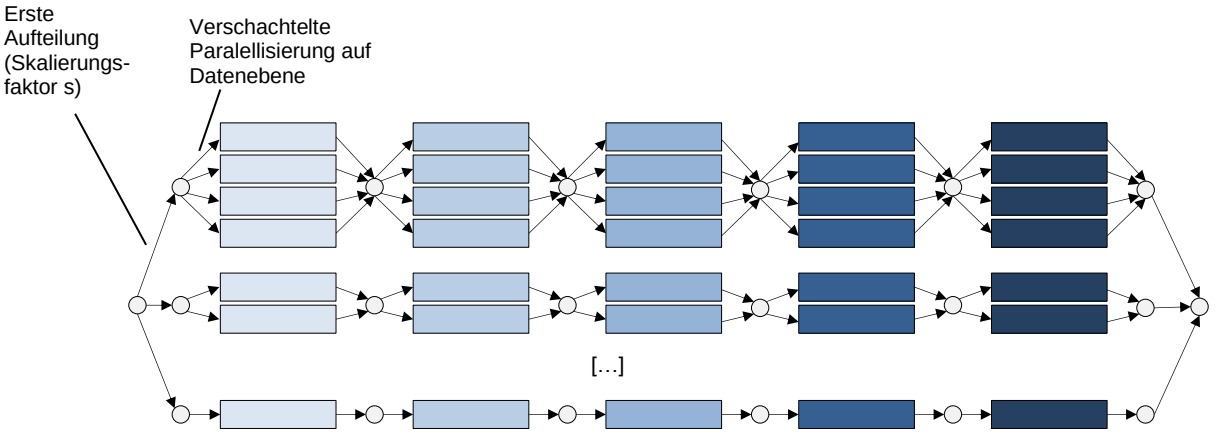


Abbildung 7.4:Verschachtelte Parallelisierung auf zwei Ebenen

Die Parallelisierung, d.h. die Erstellung von Sub-Tasks und die Verteilung auf die Prozessoren, wurde mit Hilfe der MPAL [117] Abstraktionsschicht durchgeführt. Diese ermöglicht die Abstraktion von der zugrundeliegenden Hardware und erhöht so die Portabilität der Anwendung auf andere HW-Architekturen. Neben der Parallelisierung kann auch ein umfangreiches Profiling bis auf Sub-Task-Ebene vorgenommen werden, sogenanntes Inline-Profiling. Diese Funktion wurde genutzt, um die Parallelität des HOG-Algorithmus zu evaluieren und das Kommunikationsverhalten zwischen den Tasks auf verschiedenen Prozessoren zu beschreiben.

7.2 Beschreibung Xeon Phi

Der hier verwendete Xeon Phi Coprozessor besteht aus 61 Prozessorkernen mit jeweils 4 In-Order Hardware Threads. Die Kerne haben eine Taktfrequenz von 1,2 GHz und jedem Kern sind 32kB L1 Cache und 512kB (private) L2 Cache zugeordnet.

Die Prozessorkerne sind mit einem Ring-Bus verbunden, an dem zusätzlich noch 8 GDDR-Controller sowie ein PCIe Interface angeschlossen sind (vgl. Abbildung 7.5). Dieser Ring-Bus besteht aus drei unabhängigen Bussen für jede Richtung: Der Datenbus ist 64 Byte breit und wird für die Übertragenen der Daten verwendet. Daneben existieren noch ein Steuerungs- und Adress-Bus sowie ein Bus für Bestätigungen und Cache-Kohärenzprotokolle. Alle Busse, mit Ausnahme des Daten-Busses, sind doppelt ausgelegt, so dass in Summe 10 Busse auf dem Chip realisiert sind [118].

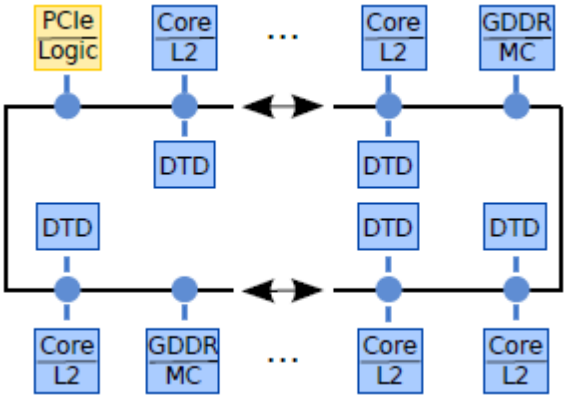


Abbildung 7.5:Schematische Darstellung des Daten-Ring Busses des Xeon Phi

Die Teilnehmer des Ring-Busses sind über sogenannte Ring-Stops an den Bus angebunden. Neben der Anbindung der Teilnehmer werden auch die einzelnen Bus-Segmente voneinander getrennt.

Die Cache-Kohärenz wird über verteilte Distributed Tag Directories (DTD) sichergestellt, die jedem L2-Cache eines Prozessorkerns zugewiesen werden. Dabei ist es nicht zwingend erforderlich, dass der lokale Inhalt eines L2-Chaches auch im zugehörigen DTD verwaltet wird. Damit nicht ein DTD zum Flaschenhals für die Speichernutzung wird, wird ein Hashing-Mechanismus verwendet, der die Inhalte des DTD gleichmäßig über alle DTDs verteilt.

Bei einem L2-Cach Miss wird zunächst in den DTD geprüft, ob die Daten in einem andern L2-Cach vorhanden sind und von dort gelesen werden können (remote cache). Andernfalls werden die Daten aus dem Arbeitsspeicher gelesen.

7.3 Hardware Modell

Um die Kommunikationsarchitektur des Xeon Phi Prozessors nachzubilden wurde ein NoC mit 70 virtuellen Kernen (VC) definiert. Dabei wurde zwischen Rechen-Kernen (compute cores, 61 VC), Speicher Schnittstellen (memory controller, 8 VC) und PCIe Schnittstelle (1 VC) unterschieden. Das NoC wurde als Ring mit Packet-Switching und XY-Routing modelliert.

Da die öffentlich zugängige Dokumentation über den Xeon Phi keine Aussagen über die genaue Anordnung der einzelnen Kerne macht, wurde ein Die-Shot des Prozessors analysiert, der in Abbildung 7.6 dargestellt ist. Dabei sind die Speicherkontroller gleichmäßig zwischen den Rechenkernen verteilt und ein Rechenkern ist durch den PCIe-Kontroller ersetzt. Dies wurde bei der Modellierung des NoC berücksichtigt.

Abbildung 7.6:Mögliche Strukturen auf dem Die-Shot des Xeon-Phi [119]

Der Fokus wurde auf die Modellierung der Datenkommunikation gelegt und die Kontroll- und Steuerinformationen zunächst vernachlässigt, da genaue Informationen über das Hashing der DTD des Prozessors nicht vorliegen. Die Ergebnisse der Untersuchung stellen die obere Grenze der zu erwartenden Kommunikationsleistung des Prozessors dar.

7.4 Profiling-Ergebnisse und Task-Graph

Der HOG-Algorithmus besteht aus mehreren Tasks. Für diese Untersuchung wurde sich auf den kommunikationsintensivsten Task „Scaling“ fokussiert. Die parallele Implementierung dieses Tasks auf dem Xeon Phi Prozessor mit 61 Kernen ergab jedoch lediglich eine maximale Beschleunigung der Ausführungszeit um den Faktor 17, wie in Abbildung 7.7 dargestellt. Es wird deutlich, dass bei einer Abbildung der Tasks auf bis zu vier Prozessorkerne eine nahezu ideale Skalierung vorliegt. Bei einer stärkeren Parallelisierung nimmt die Redundanz, die eine Maßzahl für den zusätzlichen administrativen Aufwand darstellt, kontinuierlich zu und die Abweichung der realen Ausführungszeit zu der Ausführungszeit bei einer idealen Skalierung wird immer größer. Ab dreißig Prozessorkernen lässt sich die Ausführungszeit durch weitere Parallelisierung nur noch marginal reduzieren.

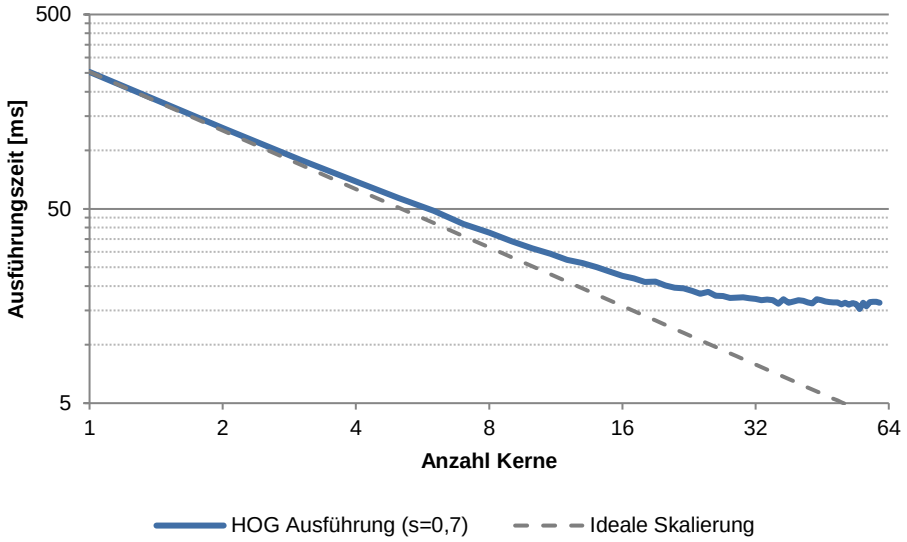


Abbildung 7.7: Paralleles Skalierungsverhalten des HOG-Algorithmus

Die Auswertung der von MPAL erzeugten Profiling Ergebnisse lassen auf einen erhöhten Aufwand für das Scheduling der Tasks auf Prozessorkerne sowie ungleich verteilte Rechenleistung schließen. Diese Effekte werden in dieser Arbeit nicht weiter betrachtet. Daneben lässt sich jedoch auch ein weiterer Anstieg der Ausführungszeiten der einzelnen Tasks um bis zu 20% oder 50 ms beobachten. Dieser Anstieg lässt sich nur durch Vorzögerungen durch die Kommunikationsarchitektur erklären (inter-core und inter-memory), da bei der Implementierung keine „mutex locks“ verwendet wurden.

In den von MPAL generierten Profiling Informationen werden nur die Ausführungszeiten der Tasks gemessen (siehe Abbildung 7.8). Diese beinhalten sowohl die Anteile für die eigentliche Ausführung des Tasks auf dem Prozessor wie auch die Verzögerungen durch die Kommunikation mit anderen Prozessoren und dem Speicher. Um diese Anteile zu separieren, wurden die Cache-Miss-Rates aus den Profiling Informationen extrahiert und mit den Memory-Delays für die einzelnen Cache Ebenen sowie die Verzögerungen durch die inter-core Kommunikation kombiniert. Diese wurde durch Micro-Benchmarks ermittelt (siehe dazu Kapitel 7.4.1 und 7.4.2).

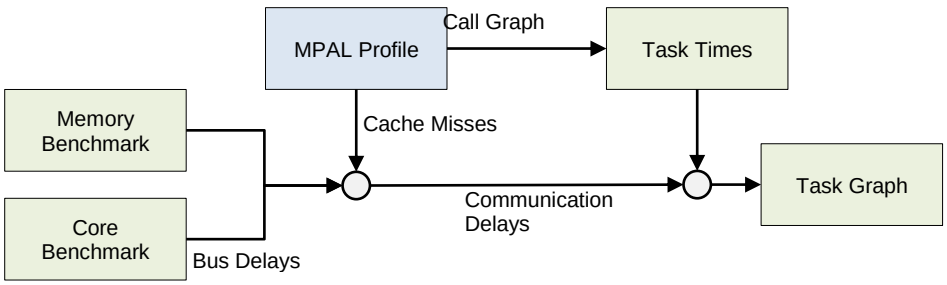


Abbildung 7.8: Vorgehen zur Erstellung von Task-Graphen

Das in dieser Arbeit modellierte Kommunikations-Pattern beruht auf einer Cache-Coherence Kommunikation. Daher wurden Tasks, die gemeinsame Cache-Lines nutzen, als diskrete Speicher-Ereignisse modelliert und aufgeteilt (Task-Splitting). L1 Cache Misses verursachen keine Verzögerung, solange die Cache Line im L2 Cache existiert. Fehlende L2 Cache Lines können sowohl aus einem Remote Cache (Remote Fill) oder dem Arbeitsspeicher (Mem Fill) bedient werden. Weitere Kommunikation wird durch im Voraus geladenen Daten und Daten, die geändert und wieder in den Speicher zurückgeschrieben werden müssen, verursacht. Aus den Profiling Daten kann nicht direkt abgelesen werden, ob ein L2-Cache Miss in einen Remote Fill oder einen Mem-Fill resultiert. Daher wurde in dieser Arbeit zusätzlich das Wissen über den Algorithmus und die Implementierung sowie der Call-Graph genutzt, diese Kommunikation zu beschreiben.

Die so ermittelten netto Ausführungszeiten werden für die Erstellung der Task-Graphen verwendet, die Kommunikations-Zeiten werden genutzt, um die Simulations-Ergebnisse zu verifizieren.

7.4.1 Memory Delay Benchmark

Um die Verzögerungen beim Zugriff auf die die unterschiedlichen Speicher zu modellieren wurde periodisch ein Daten-Array unterschiedlicher Größe gelesen und die Zugriffszeit gemessen. Um den Einfluss des Steuerungsaufwands zu minimieren, wurden jeweils mehrere Lese-Operationen in einer Messung durchgeführt.

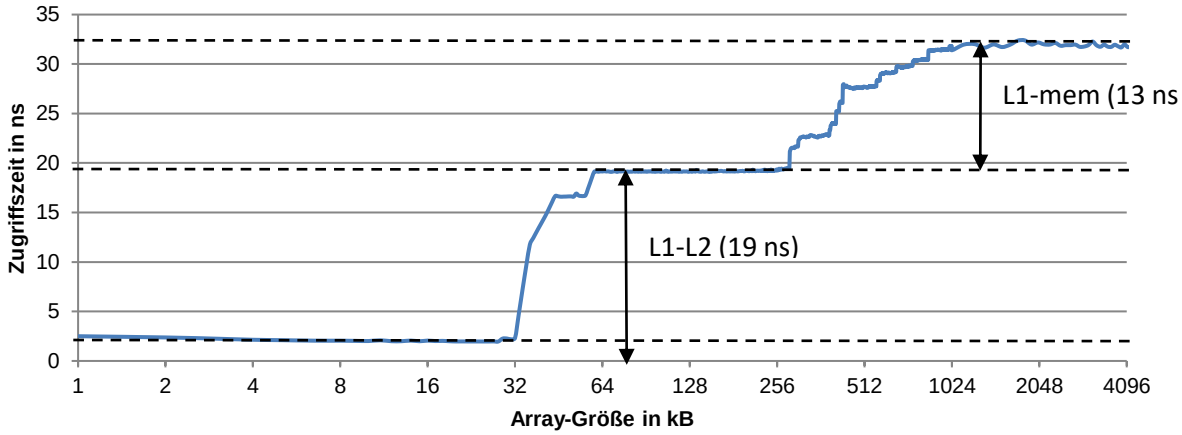


Abbildung 7.9: Verzögerungen für verschiedene Speicherebenen

In Abbildung 7.9 ist die durchschnittliche Zugriffszeit auf eine Cache-Line (64 Byte) für unterschiedliche Array-Größen aufgetragen. Bei Array-Größen bis zu 32kB werden diese komplett im L1-Cache gehalten. Sobald die Array-Größe diesen übersteigt, müssen die Daten aus dem L2-Cache nachgeladen werden. Diese zusätzliche Verzögerung (L1-L2) beträgt ca.

19 ns. Ab einer Array-Größe von 280 kB müssen Daten aus dem Arbeitsspeicher nachgeladen werden. Für größere Arrays geht die Verzögerung in eine Sättigung bei ca. 32 ns (L1-mem). Die Differenz aus der Verzögerung L1-L2 und L1-mem beschreibt den Anteil, der durch den Ring-Bus und den Speicherzugriff verursacht wird (L2-mem) und beträgt hier ca. 13 ns.

7.4.2 Core Communication Benchmark

Um die Verzögerung bei der Kommunikation zwischen einzelnen Prozessorkernen zu modellieren, wurde ein Daten-Array so dimensioniert, dass es in dem L2-Cache eines Prozessorkeren komplette gespeichert werden konnte. Im Verlauf des Benchmarks lesen nun alle Prozessorkerne nacheinander das Daten-Array aus dem L2-Cache des ersten Prozessorkerns. So wird sukzessive die Entfernung auf dem Ring-Bus erhöht bzw. nach dem dreißigsten Prozessorkern wieder reduziert. Dieses Experiment wurde mehrmals durchgeführt und die gemittelten Zugriffszeiten für eine Cache-Line ist in Abbildung 7.10 dargestellt.

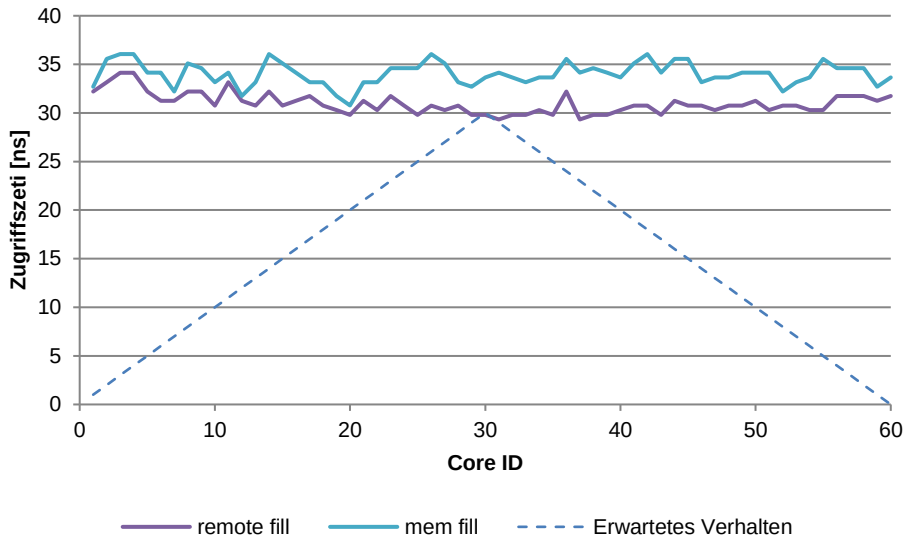


Abbildung 7.10: Verzögerungen Prozessorkern zu Prozessorkern-Kommunikation

Das erwartete Verhalten, dass die Zugriffszeit mit einer Steigerung der Distanz zwischen den Prozessorkernen ansteigt, würde in der in Abbildung 7.10 als gestrichelte Linie dargestellten Werten resultieren. Allerding zeigen die Messergebnisse ein nahezu konstantes Verhalten (remote fill). Dies Verhalten lässt sich durch den verwendeten Hashing-Mechanismus des Xeon Phi erklären. Dieser verbirgt die Distanz zwischen den einzelnen Prozessorkernen entlang des Rings.

Zusätzlich wurde ebenfalls für jeden Prozessorkern die Verzögerung für einen Zugriff auf den Speicher gemessen (mem fill). Auch hier ist die Verzögerung unabhängig von der Position im Ring Bus (und der Entfernung zum Speicher Controller). Die Verzögerung ist auch nur minimal (2%-12%) geringer, als bei der Kommunikation zwischen den Prozessorkernen

(remote fill). Beide Ergebnisse zeigen jedoch, dass die kommunikationsbedingte Verzögerung für eine L2-Cache Miss bei ca. 13 ns liegt.

7.5 NoC-Emulation

Um die auf dem realen Xeon Phi bestimmten kommunikationsbedingten Verzögerungen mit dem Modell zu vergleichen, wurden Benchmarks auf dem FPGA-basierten NoC-Emulator durchgeführt. Dazu wurde das oben beschriebene Kommunikations-Modell des Xeon Phi als Teil des Emulators synthetisiert und auf der, in Kapitel 3.6.1.2 beschriebenen, BEEcube Plattform betrieben.

Aufgrund von Speicherrestriktionen wurde das Kommunikationsverhalten für die einzelnen Teile des HOG-Algorithmus nacheinander emuliert. Dabei wurden die Daten- und Cache-Abhängigkeiten zwischen den einzelnen Stufen der Algorithmus berücksichtigt.

In Abbildung 7.11 sind die Ergebnisse dieser Emulation und die zuvor im Benchmark gemessenen kommunikationsbedingten Verzögerungen für die Stufen Scale (Sc) und Histogram Calculation (HC) dargestellt. Man sieht, dass die gemessenen Ergebnisse konstant höher sind, als die simulierten Verzögerungen. Wobei der Trend bei zunehmendem Parallelisierungsgrad (Anzahl an zur Verfügung stehenden Prozessorkerne) mit einer ansteigenden Verzögerung identisch ist.

Diese Diskrepanz hat mehrere Gründe: da nur die reine Datenkommunikation modelliert und simuliert wurde, wurden die durch das Cache-Kohärenzprotokoll verursachten Verzögerungen nicht berücksichtigt. Auch wird durch die Simulation lediglich eines Bildes der Ring-Bus nur zu Teilen ausgelastet und es entstehen keine Engpässe. Somit ist der Ring Bus für diese Applikation nicht das limitierende Element.

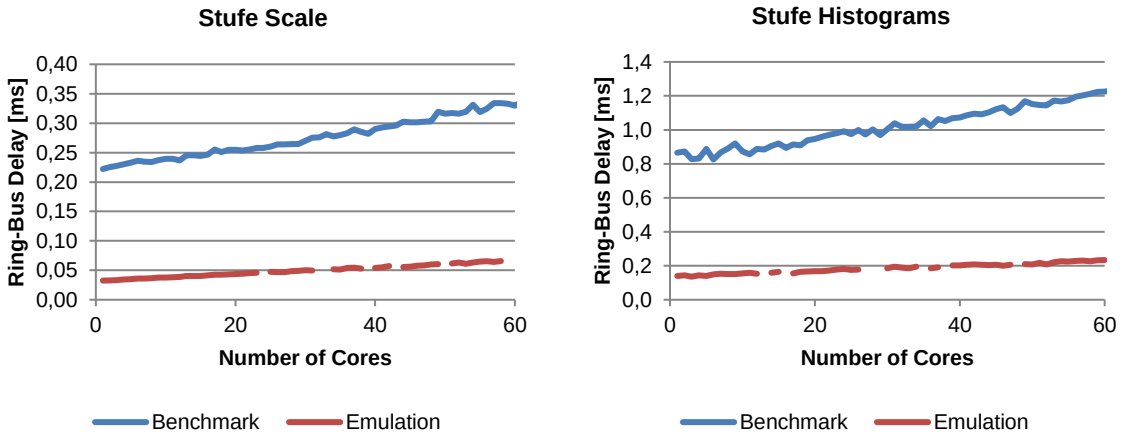


Abbildung 7.11: Aus MPAL extrahierte und simulierte Kommunikationsverzögerungen für ausgewählte Stufen des HOG Algorithmus

Aufbauend auf diesen Messungen wurde der Einfluss der NoC-Topologie auf die Verzögerung untersucht. Durch den Wechsel von einem Ring-Bus hin zu einem Mesh konnte eine Beschleunigung um bis zu 3% erreicht werden.

Bei der in diesem Kapitel vorgestellten Untersuchung konnte gezeigt werden, dass die Methodik grundsätzlich dazu geeignet ist, reale Applikationen auf einem Mehrkernprozessor zu analysieren und die Engpässe bei der Kommunikation zwischen Tasks- und Prozessoren zu identifizieren.

In dem konkreten Fall wurde festgestellt, dass der Großteil der Verzögerung jedoch nicht durch den eigentlichen Ring-Bus verursacht wird, sondern durch das verteilte Cache Kohärenzprotokoll. Mit Hilfe der Emulation auf einem FPGA konnte gezeigt werden, dass die Ausführung der Applikation dennoch um 3% beschleunigt werden könnte, in dem für das NoC eine andere Topologie (Mesh) verwendet wurde. Es muss allerdings hinterfragt werden, ob es sich für diesen Algorithmus auf dem verwendeten SoC der zusätzliche Aufwand, sowohl durch zusätzliche Entwurfskomplexität für das NoC wie auch durch auf dem Chip zusätzlich benötigte Fläche und Verlustleistung, für die Performancesteigerung rechnet. Wenn allerdings auf diesem SoC, neben dem HOG-Algorithmus, weiter Anwendungen ausgeführt werden, wird die Kommunikation der limitierende Faktor und die Performance Steigerung wird in diesem Fall entsprechend höher ausfallen.

8 Zusammenfassung

Der Trend zu Chips mit mehreren Prozessorkernen (Multi-Kern System on Chip Architekturen, SoC) wird sich auch in Zukunft fortsetzen, sodass mittelfristig SoC mit mehreren hundert Prozessorkernen erwartet werden. Bei dieser Vielzahl an funktionalen Einheiten können klassische Kommunikationsstrukturen, wie Busse oder Punkt-zu-Punkt-Verbindungen, die Anforderungen an Performance, Flexibilität und Implementierungskosten nicht mehr erfüllen. Eine häufig vorgeschlagene Lösung ist ein sogenanntes Network-on-Chip (NoC).

Im Rahmen dieser Arbeit wurde ein modulares NoC entwickelt, das eine Vielzahl an relevanten NoC-Parametern unterstützt. Die Implementierungen der einzelnen parametrisierbaren NoC-Komponenten sind in einer VHDL-Bibliothek zusammengefasst. Mit dieser Bibliothek werden NoC anhand spezifizierter NoC-Parameter automatisch generiert. NoC mit irregulären, hierarchischen oder mehrdimensionalen Topologien lassen sich nicht automatisch generieren. Für diese muss ein Entwickler NoC-Komponenten manuell verbinden und so das NoC aufbauen.

Mit Hilfe der VHDL Beschreibung wurden VLSI-Implementierungen repräsentativer NoC-Komponenten erstellt. Die Synthese (inklusive Layout) erfolgte mit Standardzellen. Eine Analyse dieser VLSI-Implementierungen zeigt, dass reguläre Teile des NoC (z.B. der Switch des Routing-Switches) einen signifikanten Anteil der Fläche und Verlustleistung ausmachen. Eine parametrisierbare Implementierung von NoC-Komponenten mit physikalisch optimierten Kernkomponenten senkt die Kosten der NoC-Komponenten deutlich.

Die Voraussetzung für eine systematische Untersuchung des Entwurfsraums für NoC, insbesondere wenn diese für definierte Anwendungen optimiert werden sollen, ist die Abschätzung der Kosten des NoC (z.B. Fläche und Verlustleistung) in einer frühen Entwurfsphase. Die in dieser Arbeit erstellten Modellfunktionen für diese Kosten basieren auf den zuvor genannten VLSI-Implementierungen von NoC-Komponenten. Die Grundlagen der Modellfunktionen liefern die Auswertung der Layouts, zur Bestimmung der Fläche, und die Simulationen der synthetisierten Netzliste, für die Verlustleistung in unterschiedlichen Arbeitspunkten.

Um sicher zu stellen, dass das NoC den Kommunikationsanforderungen genügt, muss bei dem Entwurf und der Dimensionierung eines NoC auch die Performance betrachtet werden. Deshalb wurden zunächst unterschiedliche Techniken zur Performance Analyse implementiert: statische Performance Analyse, Simulation von Colored Petri Nets, SystemC und VHDL-Beschreibungen, wie auch eine Emulation des NoC auf einem FPGA. Teil dieser Implementierungen sind NoC-Bibliotheken, die jeweils zu der VHDL NoC-Bibliothek äquivalent sind. Die Analyse und Bewertung der einzelnen Techniken zeigt, dass eine

statische Performance Analyse in frühen Entwurfsphasen Vorteile hat, die FPGA-basierte Emulation bietet die meisten Vorteile in späten Entwurfsphasen.

Basierend auf den NoC-Bibliotheken, Kostenmodellen und Performance Analyse Techniken wurde einer Entwurfsmethodik erarbeitet, die auch weniger erfahrenen Entwicklern den Entwurf von optimalen NoC für eine Applikation ermöglicht. Im Rahmen dieses automatisierten Spiral-Modells werden systematisch, in mehreren Iterationen NoC-Parameter variiert und Pareto-optimale NoC erzeugt. Die Bestimmung der Kosten und Performance in den ersten Iterationen erfolgt mit Modellfunktionen und statischer Performance Analyse, in den späteren Iterationen durch Auswertung und Simulation der VLSI-Implementierungen und FPGA-basierten Emulation.

Die Zuordnung, welcher NoC-Parameter in welcher Iteration variiert wird, erfolgt nicht zufällig. Es werden zuerst die Parameter mit einem großen Einfluss festgelegt und in den späteren Iterationen die „Feineinstellung“ vorgenommen. Einen sehr großen Einfluss hat auch das Mapping, d.h. die Abbildung der Funktionalen Einheiten des SoC auf die Network-Interfaces des NoC. Die Untersuchung und Optimierung verschiedener Mapping-Algorithmen lieferte ein Mapping Algorithmus, der eine möglichst gute Abbildungsqualität bei vertretbarem Rechenaufwand zu erzielt.

Verschiedenen NoC-Implementierungen, wie auch die Entwurfsmethodik, wurden anhand von generierten Datenverkehrsanforderungen verifiziert und Pareto-optimale NoC identifiziert. Die Anwendung der Entwurfsmethodik auf die Datenverkehrsanforderungen von realen, in der Literatur veröffentlichten, Anwendungen hat gezeigt, dass die erzeugten NoC bzw. NoC-Komponenten deutlich effizienter sind, als die in der Literatur gegebenen Implementierungen.

Eine Analyse des Kommunikationsverhaltens eines existierenden Multicore-Prozessors (Xeon Phi mit 61 Prozessorkernen und 70 NoC-Teilnehmern) für die parallele Implementierung eines Algorithmus (HOG Algorithmus) lieferte Datenverkehrsanforderungen für dieses System. Die Ergebnisse einer FPGA-basierten Emulation dieser Anforderungen auf einem NoC, dass der Kommunikationsarchitektur des SoC entspricht, wurden mit dem Kommunikationsverhalten des SoC verglichen. Grundsätzlich zeigt der Vergleich, dass die vorgestellte Methodik geeignet ist, die Kommunikation auf SoC zu analysieren und zu optimieren. Der Einfluss der Verzögerungen durch Kontrolldaten (Cache-Kohärenzprotokolle) hat in diesem Fall jedoch einen größeren Einfluss als zunächst vermutet.

Das Spiral-Modell zum NoC-Entwurf ermöglicht die effiziente, automatisierte Implementierung von NoC. Für zukünftige Arbeiten sollte der Einfluss der spezifischen Eigenschaften der funktionalen Einheiten eines NoC genauer untersucht und beim NoC

Entwurf berücksichtigt werden. Auch verspricht die Erweiterung der Methode auf dreidimensionale System-in-Package große Mehrwerte.

Wissenschaftlicher Werdegang

2008 – 2019	Promotionsstudium and der Gottfried Wilhelm Leibniz Universität Hannover
2002 – 2008	Promotionsstudium an der an der Rheinisch Westfälischen Technischen Hochschule Aachen
1997 – 2002	Diplomstudium der Informations- und Kommunikationstechnik an der Rheinisch Westfälischen Technischen Hochschule Aachen. Abschluss als Diplom-Ingenieur (Dipl.-Ing.), Diplomarbeit: „Entwurf und Implementierung einer heterogenen Hardware-Plattform zur Videosignalverarbeitung“ am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der Rheinisch Westfälischen Technischen Hochschule Aachen.
2004 – 2009	Diplomstudium der Betriebswirtschaftslehre an der FernUniversität Hagen. Abschluss als Diplom Kaufmann (Dipl.-Kfm.), Diplomarbeit: „Bewertung eines Verfahrens zur Kraftwerkseinsatzplanung anhand realer Daten des deutschen Strommarktes“ am Institut für elektrische Anlagen und Energiewirtschaft der Rheinisch Westfälischen Technischen Hochschule Aachen.
1998 – 2004	Diplomstudium der Betriebswirtschaftslehre an der Rheinisch Westfälischen Technischen Hochschule Aachen. Abschluss des Vordiploms.

Publikationsverzeichnis

2001

Neuenhahn, M., *Analyse hochqualitativer Algorithmen zur örtlichen Bildinterpolation*, Studienarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitung (EECS), RWTH Aachen

2003

Neuenhahn, M.: *Entwurf und Implementierung einer heterogenen, rekonfigurierbaren Hardware-Plattform zur Videosignalverarbeitung*, Diplomarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitung (EECS), RWTH Aachen

2004

Neuenhahn, M.; Blume, H.; Noll, T.G.: *Pareto Optimal Design of an FPGA-based Real-Time Watershed Image Segmentation*, Proceedings of the ProRISC2004, Veldhoven (Niederlande), 24. - 25. November 2004

2007

Botteck, M.; Blume, H. von Livonius, J.; Neuenhahn, M.; Noll, T. G.: *Programmable Architectures for Realtime Music Decompression*, Proceedings of the ParaFPGA 2007, Jülich, 4.-7.9.2007

Neuenhahn, M.; Blume, H.; Noll, T. G: *Quantitative Design Space Exploration of Routing-Switches for Network-on-Chip*, Proceedings of the URSI „Advances in Radio Science - Kleinheubacher Berichte“, Miltenberg, 24.-27. September 2007

Neuenhahn, M.C.; Lemmer, D.; Blume, H.; Noll, T. G.: *Quantitative Cost Modeling of Error Protection for Network-on-Chip*“, Proceedings of the ProRISC2007, Veldhoven (Niederlande), 29. - 30. November 2007

2009

Neuenhahn, M.: *Bewertung eines Verfahrens zur Kraftwerkseinsatzplanung anhand realer Daten des deutschen Strommarktes*, Diplomarbeit Betriebswirtschaftslehre am Institut und Lehrstuhl für Elektrische Anlagen und Energiewirtschaft (IAEW) der RWTH Aachen

2015

Pfefferkorn, D.; Schmider, A.; Paya-Vaya, G.; Neuenhahn, M.; Blume, H.:*FNOCEE: A framework for NoC evaluation by FPGA-based emulation*, 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2015, Samos, Griechenland, 19. – 23. Juli 2015.

2017

Arndt, J. O.; Spindeldreier, C.; Wohnrade, , K.; Pfefferkorn, D.; Neuenhahn, M.; Blume, H.: *FPGA Accelerated NoC-Simulation: A Case Study on the Intel Xeon Phi Ringbus Topology*, Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, HEART 2017, Bochum, Deutschland, 7. -9. Juni 2017.

Literaturverzeichnis

1. Moore, G.E., *Cramming More Components Onto Integrated Circuits.* Proceedings of the IEEE, 1998. **86**(1): p. 82-85.
2. Blume, H., H.T. Feldkaemper, and T.G. Noll, *Model-Based Exploration of the Design Space for Heterogeneous Systems on Chip*. The Journal of VLSI Signal Processing, 2005. 40(1): p. 19-34.
3. Jantsch, A. and H. Tenhunen, *WILL NETWORKS ON CHIP CLOSE THE PRODUCTIVITY GAP?*, in Networks on Chip, A. Jantsch and H. Tenhunen, Editors. 2003, Kluwer Academic Publishers: Dordrecht.
4. Zimmermann, H., *OSI reference model-The ISO model of architecture for open systems interconnection, in Innovations in Internetworking*. 1988, Artech House, Inc. p. 2-9.
5. Angiolini, F., et al. *Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness. in Design*, Automation and Test in Europe, DATE '06. 2006.
6. Wielage, P. and K. Goossens. *Networks on silicon: blessing or nightmare?* in Digital System Design. 2002.
7. Wentzlaff, D., et al., *On-Chip Interconnection Architecture of the Tile Processor.* Micro, IEEE, 2007. 27(5): p. 15-31.
8. Ho, R., K.W. Mai, and M.A. Horowitz, *The future of wires.* Proceedings of the IEEE, 2001. 89(4): p. 490-504.
9. Bainbridge, J. and S. Furber, *Chain: a delay-insensitive chip area interconnect.* IEEE Micro, 2002. 22(5): p. 16-23.
10. Kistler, M., M. Perrone, and F. Petrini, *Cell Multiprocessor Communication Network: Built for Speed.* Micro, IEEE, 2006. 26(3): p. 10-23.
11. Benini, L. and G. De Micheli, *Networks on chips: a new SoC paradigm.* Computer, 2002. 35(1): p. 70-78.
12. Dally, W.J. and B. Towles. *Route packets, not wires: on-chip interconnection networks.* in Design Automation Conference, 2001. Proceedings. 2001.
13. Rabaey, J.M., *System-on-Chip-Challenges in the Deep-Sub-Micron Era - A case for the network-on-a-Chip in Interconnect-Centric Design for Advanced SoC and NoC*, J. Nurmi, et al., Editors. 2005, Springer US. p. 3-24.
14. Benini, L. *Application Specific NoC Design. in Design*, Automation and Test in Europe, DATE '06. 2006.

15. ITRS, *International Technology Roadmap for Semiconductors*, 2005 Edition, Executive Summary. 2005.

16. Sylvester, D. and K. Keutzer, *A global wiring paradigm for deep submicron design.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2000. 19(2): p. 242-252.

17. Bjerregaard, T. and S. Mahadevan, *A survey of research and practices of Network-on-chip.* ACM Comput. Surv., 2006. 38(1): p. 1.

18. Veendrick, H., *Deep-Submicron CMOS ICs.* 2000, Dordrech, The Netherlands: Kluwer Academic Publishers.

19. Ho, R., *Dealing with issues in VLSI interconnect scaling*, in ISSCC. 2007.

20. Carloni, L.P., K.L. McMillan, and A.L. Sangiovanni-Vincentelli, *Theory of latency-insensitive design.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2001. 20(9): p. 1059-1076.

21. Zimmer, H. and A. Jantsch, *Error-Tolerant Interconnect Schemes, in Interconnect-Centric Design for Advanced SoC and NoC*. 2005. p. 155-176.

22. Lv, T., et al. *An adaptive dictionary encoding scheme for SOC data buses.* in Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings. 2002.

23. Chunjie, D., T. Anup, and S.P. Khatri. *Analysis and avoidance of cross-talk in on-chip buses.* in Hot Interconnects 9, 2001. 2001.

24. Bertozzi, D., L. Benini, and G. De Micheli, *Error control schemes for on-chip communication links: the energy-reliability tradeoff.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2005. 24(6): p. 818-831.

25. Neuenhahn, M.C., et al. *Quantitative Cost Modeling of Error Protection for Network-on-Chip.* in ProRISC Workshop. 2007. Veldhoven.

26. Duato, J., S. Yalamanchili, and L. Ni, *Interconnection Networks - An Engineering Approach.* 2003, San Francisco, USA: Morgan Kaufmann Publishers.

27. Kumar, S., et al. *A network on chip architecture and design methodology.* in VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on. 2002.

28. Al-Tawil, K.M., M. Abd-El-Barr, and F. Ashraf, *A survey and comparison of wormhole routing techniques in a mesh networks.* Network, IEEE, 1997. 11(2): p. 38-45.

29. Tamir, Y. and G.L. Frazier. *High-performance multiqueue buffers for VLSI communication switches*. in Computer Architecture, 1988. Conference Proceedings. 15th Annual International Symposium on. 1988.

30. Zimmer, H., et al. *Buffer-architecture exploration for routers in a hierarchical network-on-chip.* in Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. 2005.

31. Hu, J. and R. Marculescu. *Application-specific buffer space allocation for networks-on-chip router design. in Computer Aided Design*, 2004. ICCAD-2004. IEEE/ACM International Conference on. 2004.

32. Feliciian, F. and S.B. Furber. *An asynchronous on-chip network router with quality-of-service (QoS) support.* in SOC Conference, 2004. Proceedings. IEEE International. 2004.

33. Rostislav, D., et al. *An asynchronous router for multiple service levels networks on chip.* in Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on. 2005.

34. Coffman, E.G., M. Elphick, and A. Shoshani, *System Deadlocks*. ACM Comput. Surv., 1971. 3(2): p. 67-78.

35. Sfinghal, M., *Deadlock Detection in Distributed Systems*. Computer, 1989. 22(11): p. 37-48.

36. Chen, Y., *Advanced Connection Allocation Techniques in Circuit Switching Network on Chip*. Dissertation, Technische Universität Dresden, 2017.

37. Duato, J., A *New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks*. IEEE Trans. Parallel Distrib. Syst., 1993. 4(12): p. 1320-1331.

38. Dally, W.J. and H. Aoki, *Deadlock-free adaptive routing in multicomputer networks using virtual channels*. Transactions on Parallel and Distributed Systems, 1993. 4(4): p. 466-475.

39. Gaughan, P.T. and S. Yalamanchili, *Adaptive routing protocols for hypercube interconnection networks*. Computer, 1993. 26(5): p. 12-23.

40. Hangsheng, W., P. Li-Shiuan, and S. Malik. *A technology-aware and energy-oriented topology exploration for on-chip networks.* in Design, Automation and Test in Europe, 2005. Proceedings. 2005.

41. Hu, Y., et al. *Physical synthesis of energy-efficient networks-on-chip through topology exploration and wire style optimization.* in Computer Design: VLSI in Computers and

Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on. 2005.

42. Lee, K., S.-J. Lee, and H.-J. Yoo, *Low-power network-on-chip for high-performance SoC design.* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2006. 14(2): p. 148-160.

43. Neuenhahn, M.C., H. Blume, and T.G. Noll. *Quantitative analysis of network topologies for NoC-architectures on an FPGA-based emulator* in Kleinheubacher Tagung. 2006.

44. Srinivasan, K., K.S. Chatha, and G. Konjevod. *Linear programming based techniques for synthesis of network-on-chip architectures.* in Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on. 2004.

45. Goossens, K.G.W., et al., Gauranteeing *The Quality of Services in Networks on Chip*, in Networks on Chip, A. Jantsch and H. Tenhunen, Editors. 2003.

46. Rijpkema, E., et al., *Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip*, in Proceedings of the conference on Design, Automation and Test in Europe - Volume 1. 2003.

47. Goossens, K., et al. *Networks on silicon: combining best-effort and guaranteed services.* in Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings. 2002.

48. Rijpkema, E., K. Goossens, and P. Wielage. *A Router Architecture for Networks on Silicon.* in Progress 2001, 2nd Workshop on Embedded Systems. 2001.

49. Bolotin, E., et al., *QNoC: QoS architecture and design process for network on chip*. J. Syst. Archit., 2004. 50(2-3): p. 105-128.

50. Beigne, E., et al. *An asynchronous NOC architecture providing low latency service and its multi-level design framework.* in Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on. 2005.

51. Sathe, S., D. Wiklund, and D. Liu. *Design of a switching node (router) for on-chip networks.* in ASIC, 2003. Proceedings. 5th International Conference on. 2003.

52. Goossens, K., J. Dielissen, and A. Radulescu, *AEthereal network on chip: concepts, architectures, and implementations*. Design & Test of Computers, IEEE, 2005. 22(5): p. 414-421.

53. Millberg, M., et al. *Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip*. in Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings. 2004.

54. Bjerregaard, T. and J. Sparso. *A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip.* in Design, Automation and Test in Europe, 2005. Proceedings. 2005.

55. Weber, W.D., et al. *A quality-of-service mechanism for interconnection networks in system-on-chips.* in Design, Automation and Test in Europe, 2005. Proceedings. 2005.

56. Liang, J., et al., *An architecture and compiler for scalable on-chip communication.* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2004. 12(7): p. 711-726.

57. Liu, J., L.-R. Zheng, and H. Tenhunen, *Interconnect intellectual property for network-on-chip (NoC).* J. Syst. Archit., 2004. 50(2-3): p. 65-79.

58. Leroy, A., et al. *Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs.* in Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on. 2005.

59. Taylor, M.B., et al., *The Raw microprocessor: a computational fabric for software circuits and general-purpose programs*. Micro, IEEE, 2002. 22(2): p. 25-35.

60. Salminen, E., et al. *Requirements for network-on-chip benchmarking.* in NORCHIP Conference, 2005. 23rd. 2005.

61. Chang, C., J. Wawrzynek, and R.W. Brodersen, *BEE2: a high-end reconfigurable computing system*. Design & Test of Computers, IEEE, 2005. 22(2): p. 114-125.

62. Dick, R.P., D.L. Rhodes, and W. Wolf. *TGFF: task graphs for free.* in The Sixth International Workshop on Hardware/Software Codesign, 1998. (CODES/CASHE '98) 1998.

63. Weicker, R.P., *An overview of common benchmarks.* Computer, 1990. 23(12): p. 65-75.

64. Murali, S. and G. De Micheli. *Bandwidth-constrained mapping of cores onto NoC architectures.* in Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings. 2004.

65. Xu, J., et al. *A methodology for design, modeling, and analysis of networks-on-chip.* in Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on. 2005.

66. Kukkala, P., et al. *UML 2.0 implementation of an embedded WLAN protocol.* in Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on. 2004.

67. Neeb, C., M.J. Thul, and N. Wehn. *Network-on-chip-centric approach to interleaving in high throughput channel decoders.* in Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on. 2005.

68. Wiklund, D., *Development and Performance Evaluation of Networks on Chip*, in Department of Electrical Engineering. 2008, Linköping University: Linköping.

69. Lahiri, K., A. Raghunathan, and S. Dey. *Evaluation of the traffic-performance characteristics of system-on-chip communication architectures*. in VLSI Design, 2001. Fourteenth International Conference on. 2001.

70. Salminen, T. and J.P. Soininen. *Evaluating application mapping using network simulation.* in System-on-Chip, 2003. Proceedings. International Symposium on. 2003.

71. Pande, P.P., et al. *Effect of traffic localization on energy dissipation in NoC-based interconnect. in Circuits and Systems*, 2005. ISCAS 2005. IEEE International Symposium on. 2005.

72. Keutzer, K., et al., *System-level design: orthogonalization of concerns and platform-based design*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2000. 19(12): p. 1523-1543.

73. Hu, J. and R. Marculescu. *Energy-aware mapping for tile-based NoC architectures under performance constraints*. in Design Automation Conference,ASP-DAC 2003, Asia and South Pacific. 2003.

74. Rödder, W., F. Kulmann, and H.P. Reidmacher, *Operations Research-Optimieren mit intelligenten Strategien*. 2002, Hagen: FernUniversität in Hagen.

75. Blume, H. "Exploration des Entwurfsraumes für heterogene Architekturen zur digitalen Videosignalverarbeitung", Habilitation am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen, 2008.

76. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, *Optimization by Simulated Annealing*. Science, Number 4598, 13 May 1983, 1983. 220, 4598: p. 671-680.

77. Glover, F., *Future paths for integer programming and links to artificial intelligence*. Comput. Oper. Res., 1986. 13(5): p. 533-549.

78. Altera Corporation, *Nios II Processor Reference Handbook*. 2007, Altera Corporation.

79. Altera Corporation, *Software: Quartus II v7.1 SP1*. 2007, Altera Corporation.

80. Pfefferkorn, D., et al. *FNOCEE: A framework for NoC evaluation by FPGA-based emulation*. in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) 2015

81. Wieferink, A., et al., *System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms*. Computers and Digital Techniques, IEE Proceedings -, 2005. 152(1): p. 3-11.

82. Petri, C.A., *Kommunikation mit Automaten,* in Institut für instrumentelle Mathematik. 1962, Friedrich-Wilhelm-University Bonn: Bonn.

83. Girault, C. and R. Valk, *Petri Nets for Systems Engineering*. 2004, Berlin: Springer. 607.

84. Dotoli, M.F., M. P., *An Urban Traffic Network Model via Coloured Petri Nets*. Control Engineering Practice, 2006. 14: p. 17.

85. Shang, D.B., F.; Koelmans, A.; Yakovlev, A.; Xia, F., *Asynchronous system synthesis based on direct mapping using VHDL and Petri nets.* IEE Proceedings - Computers and Digital Techniques, 2004. 151(3): p. 12.

86. Vinter Ratzer, A., et al. *CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets*. in the 24th International Conference on Applications and Theory of Petri Nets (ICATPN) 2003. 2003.

87. Blume, H., et al., *Petri Net Based Modelling of Communication in Systems on Chip*, in Petri Net: Theory and Application. 2007, ARS-Publishing.

88. Mentor Graphics, Software: *ModelSim*. 2007, Mentor Graphics.

89. Schleifer, J., *Modellierung von Networks-on-Chip mit Deterministischen und Stochastischen Petri-Netzen*, Diplomarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme. 2006, RWTH Aachen: Aachen.

90. Fischer, E., *Analytische Modellierung zur Entwurfsraumexploration von Verbindungsnetzwerken in Vielkernprozessoren*. Mobile Nachrichtenübertragung. Vol. 74. 2015, Dresden.

91. Weiss, O., M. Gansen, and T.G. Noll. *A flexible datapath generator for physical oriented design.* in Solid-State Circuits Conference, 2001. ESSCIRC 2001. Proceedings of the 27th European. 2001.

92. Boehm, B.W., *A spiral model of software development and enhancement*. Computer, 1988. 21(5): p. 61-72.

93. Synopsys, Software: *Design Compiler*. 2007, Synopsys.

94. Cadence, Software: *Encounter*. 2006, Mentor Graphics.

95. Cadence, Software: *Design Framework*. 2006, Cadence.

96. Synopsys, Software: *NanoSim*. 2006, Synopsys.

97. Jian, L., S. Swaminathan, and R. Tessier. *ASOC: a scalable, single-chip communications architecture.* in Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on. 2000.

98. Glass, C.J. and L.M. Ni. *The Turn Model for Adaptive Routing*. in Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on. 1992.

99. Geisler, D., *Implementierung von physikalisch optimierten Komponenten eines Routing-Switche Network-on-Chip*, Studienarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme. 2008, RWTH Aachen: Aachen.

100. Stergiou, S., et al. *xpipes Lite: a synthesis oriented design library for networks on chips.* in Design, Automation and Test in Europe, 2005. Proceedings. 2005.

101. Vangal, S., et al. *An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS*. in Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International. 2007.

102. Lattard, D., et al. *A Telecom Baseband Circuit based on an Asynchronous Network-on-Chip*. in Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International. 2007.

103. Wolkotte, P.T., et al. *An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip.* in Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. 2005.

104. Ahonen, T., et al., *A Brunch from the Coffee Table-Case Study in NoC Platform Design*, in Interconnect-Centric Design for Advanced SoC and NoC. 2005. p. 425-453.

105. Tschauner, M., *Kostenmodellierung von NoC-Komponenten*, Studienarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme. 2008, RWTH Aachen: Aachen.

106. Jalabert, A., et al. *xpipesCompiler: a tool for instantiating application specific networks on chip.* in Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings. 2004.

107. Stensgaard, M.B. and J. Sparso. *ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology.* in Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on. 2008.

108. Jaspers, E.G.T. and P.H.N. de With. *Chip-set for video display of multimedia information.* in Consumer Electronics, 1999. ICCE. International Conference on. 1999.

109. Artisan Components Inc., *TSMC 90nm CLN90G Process SAGE-X v3.0 Standard Cell Library Databook.* 2005, Artisan Components, Inc.

110. Wang, H.-S., et al. *Orion: a power-performance simulator for interconnection networks.* in Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on. 2002.

111. Dall'Osso, M., et al. *xpipes: a Latency Insesitive Parameterized Network-on-Chip Architecture For Multi-Processor SoCs.* in 21st International Conference on Computer Design (ICCD'03). 2003.

112. Dalal, N. and B. Triggs, *Histograms of Oriented Gradients for Human Detection*, in International Conference Computer Vision and Pattern Recognition (CVPR), IEEE. 2005.

113. Arndt, O.J., et al., *Performance Evaluation of the Intel Xeon Phi Manycore Architecture Using Parallel Video-Based Driver Assistance Algorithms*, in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV). 2014. p. 125-132.

114. Arndt, O.J., et al., *FPGA Accelerated NoC-Simulation - A Case Study on the Intel Xeon Phi Ringbus Topology*, in International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017). 2017.

115. Burges, C.J.C., ed. *A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery*. 1998.

116. Kriesel, D., *Ein kleiner Überblick über Neuronale Netze*. 2007.

117. Arndt, O.J., T. Lefherz, and H. Blume. *Abstracting Parallel Programming and Its Analysis Towards Framework Independant Development*. in International Symposium Embedded Multicore/Many-core Systems-on-Chip (MCSoC). 2015.

118. Intel *Intel®Xeon Phi x100 Family Coprocessor - the Architecture* 2016, https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner Abrufdatum: 29.10.2016.

119. Intel *Intel Xeon Phi die-shot*. 2016, http://download.intel.com/newsroom/kits/xeon/phi/gallery/images/XeonPhiDie_02.jpg, Abgerufen am 31.10.2016.

Anhang A Unterstützte NoC-Parameter

Tabelle A.1: Unterstützte NoC-Parameter – Allgemein

Parameter	**Wertebereich**
Bruttodatenwortbreite	16 ... beliebig
Anzahl der Teilnehmer	beliebig
Technologie	TSMC90 (90 nm, 4 Metall-Layer)
Topologie (homogen)	Mesh
	Torus
	Ring
	Lineare Anordnung
	Hyper-Cube
	Star

Tabelle A.2: Unterstützte NoC-Parameter – Routing-Switch[1]

Parameter	**Parameter-Variante**
Buscodierung	keine
Implementierung des Routing-Switch	Zentralisierte Implementierung
	Verteilte Implementierung
Implementierung des Routing-Algorithmus	Zustandsmaschine
	Routing-Tabelle
Register an Eingangsports des Routing-Switches	keine
	an jedem Eingangsport
	an ausgewählten Eingangsports
Datenfehlererkennung- und –korrektur	keine
	Hamming-Code
	Enhanced Hamming Code
Vermittlungstechnik	Leitungsvermittlung
	Wormhole-Routing
Priorisierung von Daten und präemptiver Zugriff auf NoC-Ressourcen	ohne
	mit
Routing-Algorithmus	XY-Routing
	Adaptives XY-Routing
	Adaptives XY-Routing mit Backtracking
	Adaptives XY-Routing mit dynamischer Routing-Tabelle

[1] Die Fläche des Routing-Switches ist auch von der Länge der Links sowie der Lastkapazität der Eingangsports der angeschlossenen NoC-Komponente abhängig.

Tabelle A.3: Unterstützte NoC-Parameter – Network-Interface[1]

Parameter	**Parameter-Variante**
Buscodierung	keine
Datenfehlercodierung	keine
	Parity-Code
	Hamming-Code
	Enhanced Hamming Code
Datenfehlerbehandlung	Send and Forget
	Send and Wait
	Send and Wait mit Sliding Window Enhancement
Vermittlungstechnik	Leitungsvermittlung
	Wormhole-Routing
Priorisierung von Daten und präemptiver Zugriff auf NoC-Ressourcen	ohne
	mit
Routing-Algorithmus	XY-Routing
	Adaptives XY-Routing
	Adaptives XY-Routing mit Backtracking
	Adaptives XY-Routing mit dynamischer Routing-Tabelle
Behandlung von Verbindungsfehlern	Neuübertragung nach einer festen Anzahl von Wartezyklen
	Neuübertragung nach einer zufälligen Anzahl von Wartezyklen
	Neuübertragung mit erhöhter Priorität

Tabelle A.4: Unterstützte NoC-Parameter – Link

Parameter	**Parameter-Variante**
Synchronisation	Synchron
Konnektivität der Verbindungsleitungen	2 unidirektionale Verbindungen
	1 bidirektionale Verbindung
	1 unidirektionale Verbindung
Länge der Verbindungsleitung	beliebig
Abstand zwischen Verbindungsleitungen	
Breite der Verbindungsleitungen	

[1] Die Fläche des Network-Interfaces ist auch von der Lastkapazität an den Ausgangsports abhängig. Diese wird durch die angeschlossenen NoC-Komponenten bestimmt.

Anhang B Abkürzungen und Formelzeichen

B.1 Abkürzungen

A	Fläche
A*	A*-Algorithmus, ein Mapping-Algorithmus
ackn	Bestätigungssignal des korrekten Empfangs eines Datenwortes
A-d-RT	Routing-Algorithmus: Adaptives Routing mit dynamischer Routing-Tabelle
A-XY	Routing-Algorithms: Adaptives XY-Routing
A-XY-RT	Routing-Algorithmus: Adaptives XY-Routing, Realisierung mit Routing-Tabelle
B&B	Branch & Bound Algorithmus, ein Mapping-Algorithmus
BC	Buscodierung
BDC	Busdecodierung
BE	Best Effort Services
C++	Programmiersprache
CMOS	Complementary Metal Oxide Semiconductor
CPN	Colored Petri Net
DED	Double Error Detection. Z.B. Hamming Code
DPG	Datenpfad-Generator
DSP	Digitaler Signal Prozessor
E	Energie einer Datenübertragung
ECC/EDC	Error Correcting or Detecting Code. Block im Routing-Switch und Network-Interface
eFPGA	eingebetteter FPGA, Teil eines System-on-Chip
Fixed	Variante der Verbindungsfehlerbehandlung
FPGA	Field Programmable Gate Array
GS	guaranteed services
HOPS	Kostenmaß bei Mapping-Algorithmen (Anzahl der Hops)

int	Integer Datentyp
ISO	International Organization for Standardization
LV	Leitungsvermittlung
NI	Network-Interface
NI-Receive	Empfangendes Modul eines Network-Interfaces
NI-Receive,Control	Modul des empfangenden Teils eines Network-Interfaces das die Vermittlungstechnik realisiert
NI-Send	Sendendes Modul eines Network-Interfaces
NI-Send,CFH	Modul des sendenden Teils eines Network-Interfaces das die Verbindungsfehlerbehandlung realisiert
NI-Send,Control	Modul des sendenden Teils eines Network-Interfaces das die Vermittlungstechnik realisiert
NK	Nachfolgeknoten bei Suchbäumen
NoC	Network-on-Chip
OSI	Open System Interconnection Reference
PC	Personal Computer
Priority	Variante der Verbindungsfehlerbehandlung
P_V	Verlustleistung
Q	Datenquelle (Abbildung 3.5)
QoS	Quality of Service
Random	Variante der Verbindungsfehlerbehandlung
Reg	Register im Routing-Switch (Abbildung 4.34)

RS	Routing-Switch
S	Datensenke (Abbildung 3.5)
S&F	Send and Forget
S&W	Send and Wait
S&W SWE	Send and Wait mit Sliding Window Enhancement
SEC	Single Error Correction z.B.: Hamming Code
SEC/DED	Single Error Correction and Double Error Detection. z.B. Enhanced Hamming Code
SED	Single Error Detection. Fehlererkennungs-Algorithmus z.B. Paritäts-Code
SoC	System-on-Chip
T	Treiber (Abbildung 4.34)
TED	Triple Error Detection. z.B. Enhanced Hamming Code
VerilogHDL	Hardware-Beschreibungssprache
VHDL	Very High-Speed Integrated Circuit Hardware Description Language, Hardware-Beschreibungssprache
VLSI	Very Large-Scale Integration
WEIGHTED_HOPS	Kostenmaß bei Mapping-Algorithmen (Anzahl der Hops, gewichtet mit der übertragenen Datenmenge)
WH	Wormhole-Routing
XY	Routing-Algorithmus: XY-Routing
XY-BT	Routing- Algorithmus: Adaptives XY-Routing mit Backtracking
XY-RT	Routing- Algorithmus: XY-Routing, Realisierung mit Routing-Tabelle

B.2 Formelzeichen

A	Fläche oder Menge der Kanten eines Task-Graphen
$a_{i,j}$	Eine gerichtete Kante des Task-Graphen, die eine Datenübertragung vom Knoten vi zum Knoten vj repräsentiert.
A_L	Fläche der Links eines NoCs (lediglich 'Logik' z.B. Repeater, Register etc.)
A_L'	Fläche die für die Leitungen der Links benötigt wird
A_{NI}	Fläche der Network-Interfaces eines NoCs
A_{NoC}	Fläche des gesamten NoCs (ohne Fläche für Verbindungsleitungen)
A_{RS}	Fläche der Routing-Switches eines NoCs
C_{ground}	Kapazität einer Verbindungsleitung gegenüber dem Null-Potenzial
C_m	Kapazität einer Verbindungsleitung zu benachbarten Verbindungsleitungen.
C_{total}	Gesamte Kapazität einer Verbindungsleitung (vgl.2.2.2)
D	Datenmenge
D_b	Brutto Datenmenge einer Datenübertragung, inklusive Redundanz für Fehlerschutz und Steuerinformationen
D_n	Netto Datenmenge einer Datenübertragung n (2.3.2)
d_n	Datenblockgröße (2.3.2)
E	Energie (4.4)
E_C	Energie, die für eine Datenübertragung inklusive Verbindungsauf- und -abbau benötigt wird
E_{Link}	Energie, die bei einer Datenübertragung in einem Link verbraucht wird
$E_{NI\text{-}Receive}$	Energie, die bei einer Datenübertragung im empfangenden Network-Interface verbraucht wird
$E_{NI\text{-}Send}$	Energie, die bei einer Datenübertragung im sendenden Network-Interface verbraucht wird
E_{RS}	Energie, die bei einer Datenübertragung in einem Routing-Switch verbraucht wird

f_{max}	Maximal erreichbare Taktfrequenz, mit der das NoC betrieben werden kann
$G_{TG}(V,A)$	Task-Graph, bestehend aus den Knoten/Prozessen V, die durch die Menge der Kanten A verbunden sind
H	Anzahl der Hops einer Datenübertragung (Besuchte Routing-Switches)
L	Anzahl der besuchten Links bei einer Datenübertragung
l_{req}	Angeforderte Auslastung
l	Erreichte Auslastung
n	Anzahl funktionaler Einheiten eines NoC (3.4)
n_L	Anzahl der Links eines NoCs
n_{NI}	Anzahl der Network-Interfaces eines NoC
n_{RS}	Anzahl der Routing-Switches eines NoCs
$P_{NI\text{-}Receive}$	Verlustleistung, die bei einer Datenübertragung im empfangenden Network-Interface verbraucht wird
$P_{NI\text{-}Send}$	Verlustleistung, die bei einer Datenübertragung durchschnittlich im sendenden Network-Interface verbraucht wird.
P_{RS}	Verlustleistung, die bei einer Datenübertragung durchschnittlich in einem Routing-Switch verbraucht wird.
r	Datenrate
r_b	Brutto Datenrate einer Datenübertragung
$r_{brutto,max}$	Maximale Brutto-Datenrate des gesamten NoC
$r_{l,brutto,max}$	Maximale Brutto-Datenrate, die über einen Link übertragen werden kann
r_n	Netto Datenrate einer Datenübertragung
s	Skalierungsfaktor
T	Definierten Zeitraum, zum Beispiel für einen komplettes Experiment
t_{Ai}	Ausführungszeit, die ein Task auf einer funktionalen Einheit benötigt
t_C	Zeit für gesamte Verbindung (Auf- und Abbau + Datenübermittlung)

t_{opt}	Zeit, die ohne Verzögerung durch NoC für eine Datenübertragung benötigt wird
t_{real}	Zeit, die für eine Datenübertragung benötigt wird, inklusive der durch das NoC verursachten Verzögerung
t_T	Übertragungsdauer, Zeit für Aufbaue einer Verbindung und Datenübertragung
V	Menge der Knoten/Prozesse eines Task-Graphen
v_i	Ein Knoten des Task-Graphen, der einen Prozess symbolisiert
w	Nettodatenwortbreite
w'	Bruttodatenwortbreite
ΔP_x	Verlustleistung die bei einer Datenübertragung in einer NoC-Komponente x zusätzlich zur Verlustleistung dieser Komponente im Ruhezustand verbraucht wird
Δt	Latenz die durch ein NoC verursacht wird
Δt_{avg}	Durchschnittliche Latenz, die durch ein NoC verursacht wird
Δt_{max}	Maximale Latenz, die durch ein NoC verursacht wird
Δt_{min}	Minimale Latenz, die durch ein NoC verursacht wird
λ	Technologiefeinheit
σ	Schaltaktivität (4.3)

Anhang C Beschreibung des künstlichen Datenverkehrs

Tabelle C.5: TG_1 - Datenmenge, die pro Periode vom Start- zum Zielknoten übertragen werden soll

Startknoten	Zielknoten	Datenmenge pro Periode in Bit
0	1	1,25E+04
1	2	1,32E+04
1	3	3,04E+04
1	4	2,28E+04
1	5	1,25E+04
1	6	1,04E+04
6	7	1,47E+04
2	7	1,25E+04
6	8	2,51E+04
5	8	2,39E+04
2	8	8164
4	8	9496
3	8	2,39E+04
0	8	1,17E+04
8	9	9496
7	9	2,43E+04
0	9	2,53E+04
3	9	1,94E+04
5	9	1,32E+04
1	9	1,32E+04
2	9	8164
0	10	1,02E+04
3	10	1,94E+04
4	17	1,02E+04
8	10	3,04E+04
5	11	7266
9	11	9496
8	11	2,28E+04
3	11	7266
4	11	2,05E+04
8	12	9496
0	12	1,47E+04
11	12	1,04E+04
6	12	2,00E+04
10	12	1,66E+04
3	12	2,05E+04
9	12	2,51E+04
12	13	8091
12	14	3,03E+04
12	15	2,28E+04

Tabelle C.6: TG_2 - Datenmenge, die pro Periode vom Start- zum Zielknoten übertragen werden soll

Startknoten	Zielknoten	Datenmenge pro Periode in Bit
0	3	1,04E+06
3	4	2,75E+06
3	5	988511
2	6	2,43E+06
1	6	1,25E+06
6	7	1,32E+06
1	7	3,04E+06
3	7	2,28E+06
6	8	1,25E+06
5	8	1,04E+06
4	8	1,47E+06
3	8	1,25E+06
8	9	1,54E+06
0	10	2,39E+06
4	10	816439
5	11	949614
9	11	2,39E+06
8	12	1,17E+06
0	12	949614
11	12	2,43E+06
7	13	2,53E+06
7	14	1,94E+06

Anhang D Funktionen zur Modellierung der Fläche und Verlustleistung

Im folgenden Anhang sind Modellfunktionen für die Kosten eines NoC angegeben. Diese basieren auf den Ergebnissen, die in Kapitel 4 präsentiert wurden.

Tabelle D.7: Liste alle Parameter, die auf die Kosten der NoC-Komponenten einen Einfluss haben.

Parameter		**Optionen**	**Abkürzung**
Bruttodatenwortbreite[1]	w‘	>8	
Anzahl der Ports eines RS	p	>3	
Implementierung	I	Standard Zellen	Std
		Spezielle Register Zelle	Reg
		Standard Zellen mit physikalisch optimierte Kernkomponenten	Opt
Register an den Eigangsports	REG		
Vermittlungstechnik	V	Leitungsvermittlung	LV
		Wormhole-Routing	WH
Routing Algorithmus	RA	XY-Routing	XY
		Adaptives XY-Routing	A-XY
		Adaptives XY-Routing mit Backtracking	A-XY-BT
		Adaptives Routing mit dynamischer Routing-Tabelle	A-d-RT
Anzahl der Datenworte im Sendbuffer	b	> 0, die Größe des Buffers, der nur beim Wormhole-Routing verwendet wird, ist u.a. Abhängig von der Größe und Topologie des NoC	
Unterstützung von Priorisierten Verbindungen	PRI		
Verbindungsfehlerbehandlung	CVH	Erneuter Verbindungsaufbau nach fester Wartezeit	Fixed
		Erneuter Verbindungsaufbau nach zufälliger Wartezeit	Random
(Maximale) Anzahl von Wartezyklen beim CFH	w		

[1] Die Bruttodatenwortbreite wird neben der Nettodatenwortbreite durch Parameter wie Datenfehlerschutz und Buscodierung beeinflusst. Da diese Parameter nicht modelliert wurden ist $w = w‘$.

Tabelle D.8: Aufteilung der NoC-Komponenten in Teilkomponenten mit Zuordnung der Parameter, welche die Kosten der jeweiligen Komponente beeinflussen.

NoC-Komponenten			**Parameter**
Network-Interface			w, V, CFH, PRI, b
	Send		w, V, CFH, PRI
		Control	w, V
		CFH	CFH, PRI, w
		Buffer	w, b, (V)[1]
		Mux	w, (V)7
	Receive		w, V
		Control	w, V
Routing Switch			
	Router		RA
	Arbiter		p, RA, PRI
	Switch		p, w‘, I
	Register		V, REG, I, w‘

D.1 Routing-Switch

D.1.1 Parameter

D.1.2 Fläche

$$A_{RS} = A_{RS,\mathrm{Re}g} + A_{RS,Switch} + A_{RS,Router} + A_{RS,Arbiter} \tag{D.1}$$

$$A_{RS,\mathrm{Re}g} = \begin{cases} 0\mu m^2 & ,V = LV, I = * \\ \frac{1134,70 \cdot p + 201,19}{5874,69} \cdot (135,38 \cdot w' + 1622,00)\,\mu m^2 & ,V = LV, REG, I = Std \\ \frac{2735,40 \cdot p + 834,91}{14511,91} \cdot (406,14 \cdot w' + 1778,90)\,\mu m^2 & ,V = WH, I = Std \\ \frac{3870,10 \cdot p + 1036,10}{20386,60} \cdot (541,52 \cdot w' + 3400,90)\,\mu m^2 & ,V = WH, REG, I = Std \\ (77,56 \cdot w' \cdot p)\,\mu m^2 & V = LV, REG, I = Opt \\ (232,67 \cdot w' \cdot p)\,\mu m^2 & V = WH, I = Opt \\ (310,22 \cdot w' \cdot p)\,\mu m^2 & V = WH, REG, I = Opt \end{cases} \tag{D.2}$$

$$A_{RS,Switch} = \begin{cases} \frac{928,25 \cdot p^{1,52365}}{10227,66} \cdot (285,34 \cdot w' + 1096,78)\,\mu m^2 & I = Std \\ p \cdot (21,65 + 79,39 \cdot w')\,\mu m^2 & I = Opt,\ p <= 5 \end{cases} \tag{D.3}$$

[1] Komponente wird nur bei der Vermittlungstechnik Wormhole-Routing verwendet

$$A_{RS,Arbiter} = \begin{cases} 578,03 \cdot p^{1,6388}\mu m^2 & RA \neq XY-BT \\ 1023,11,03 \cdot p^{1,6388}\mu m^2 & RA \neq XY-BT, PRI \\ 1109,81 \cdot p^{1,6388}\mu m^2 & RA = XY-BT \\ 1959,52 \cdot p^{1,6388}\mu m^2 & RA = XY-BT, PRI \end{cases} \quad \text{(D.4)}$$

$$A_{RS,Router} = \begin{cases} 5053,57\mu m^2 & ,RA = A-d-RT \\ 426,17\mu m^2 & ,RA \neq A-d-RT \end{cases} \quad \text{(D.5)}$$

D.1.3 Verlustleistung

$$P_{RS,stat} = f(w', p, V, RA)$$

$$= \begin{cases} p^{1,1835} \cdot (4,4876 \cdot w' + 75,5163)\mu W & V = LV, RA \neq A-d-RT \\ p^{1,1005} \cdot (12,9928 \cdot w' + 102,2174)\mu W & V = WH, RA \neq A-d-RT \\ 1,0849 \cdot p^{1,1835} \cdot (4,4876 \cdot w' + 75,5163)\mu W & V = LV, RA = A-d-RT \\ 1,0849 \cdot p^{1,1005} \cdot (12,9928 \cdot w' + 102,2174)\mu W & V = WH, RA = A-d-RT \\ 1.0682 \cdot p^{1,1835} \cdot (4,4876 \cdot w' + 75,5163)\mu W & V = LV, RA \neq A-d-RT, PRI \\ 1.0682 \cdot p^{1,1005} \cdot (12,9928 \cdot w' + 102,2174)\mu W & V = WH, RA \neq A-d-RT, PRI \\ 1.1589 \cdot p^{1,1835} \cdot (4,4876 \cdot w' + 75,5163)\mu W & V = LV, RA = A-d-RT, PRI \\ 1.1589 \cdot p^{1,1005} \cdot (12,9928 \cdot w' + 102,2174)\mu W & V = WH, RA = A-d-RT, PRI \end{cases} \quad \text{(D.6)}$$

$$\Delta P_{RS}(w', p, V) = \begin{cases} (0,344 \cdot p + 0,828) \cdot (10,9 \cdot w' + 15,4)\mu W & ,V = LV \\ (0,296 \cdot p + 0,841) \cdot (11,9 \cdot w' + 21,2)\mu W & ,V = WH \end{cases} \quad \text{(D.7)}$$

D.2 Network-Interface

D.2.1 Fläche

$$A_{NI} = A_{NI-Send} + A_{NI-Receive} \quad \text{(D.8)}$$

$$A_{NI-Receive} = A_{NI-Receive,Control} \quad \text{(D.9)}$$

$$A_{NI-Send} = \begin{cases} A_{NI-Send,Control} + A_{NI-Send,CFH} & V = LV \\ A_{NI-Send,Control} + A_{NI-Send,CFH} + A_{NI-Send,Mux} + A_{NI-Send,Buffer} & V = WH \end{cases} \quad \text{(D.10)}$$

$$A_{NI-Send,Control} = \begin{cases} (7,796 \cdot w + 663,87)\,\mu m^2 & ,V = LV \\ (6,8635 \cdot w + 589,99)\,\mu m^2 & ,V = WH \end{cases} \quad \text{(D.11)}$$

$$A_{NI-Send,Mux} = (11,378 \cdot w + 94,147)\,\mu m^2 \quad \text{(D.12)}$$

$$A_{NI-Send,Buffer} = \begin{cases} 0\mu m^2 & ,b = 0 \\ (41,78 \cdot b + 109,13) \cdot (w+2)\,\mu m^2 & ,b > 0, I = Std \\ (11,28 \cdot b + 29,47) \cdot (w+2)\,\mu m^2 & b > 0, I = Reg \end{cases} \quad \text{(D.13)}$$

$$A_{NI-Send,CFH} = \begin{cases} (29,58+169,49\cdot\log_{10}(w))\,\mu m^2 & ,CFH = fixed \\ (203,58+169,49\cdot\log_{10}(w))\,\mu m^2 & ,CFH = fixed, PRI \\ (145,89\cdot w)\,\mu m^2 & ,CFH = random \\ (174+145,89\cdot w)\,\mu m^2 & ,CFH = random, PRI \end{cases} \quad \text{(D.14)}$$

$$A_{NI-Receive,Control} = \begin{cases} (7,8051\cdot w+490,69)\,\mu m^2 & ,V = LV \\ (6,3746\cdot w+292,72)\,\mu m^2 & ,V = WH \end{cases} \quad \text{(D.15)}$$

D.2.2 Verlustleistung

$$P_{NI\_Receive,Idle} = \begin{cases} 50,0\,\mu W & ,V = LV \\ (75,79+28,88\cdot w)\,\mu W & ,L = WH \end{cases} \quad \text{(D.16)}$$

$$\Delta P_{NI\_Receive} = \begin{cases} (0,7\cdot w-3,3)\,\mu W & ,V = LV \\ (11,6\cdot w+34,2)\,\mu W & ,L = WH \end{cases} \quad \text{(D.17)}$$

$$P_{NI\_Send,Idle} = \begin{cases} (3,42\cdot\ln(w)+36,46)\,\mu W & ,V = LV, CFH = fixed \\ (6,95\cdot\ln(w)+24,33)\,\mu W & ,V = LV, CFH = random \\ (19,23+7,37\cdot w)\,\mu W & ,L = WH, I = R \\ (274,7+105,3\cdot w)\cdot(0,068+0,029\cdot b)\,\mu W & ,L = WH, I = Std \end{cases} \quad \text{(D.18)}$$

$$\Delta P_{NI\_Send} = \begin{cases} \left((46,5+0,7\cdot w)\cdot(0,06\cdot\ln(w)+0,78)\right. \\ \left.-(3,42\cdot\ln(w)+36,46)\right)\mu W & ,V = LV, CFH = fixed \\ \left((46,5+0,7\cdot w)\cdot(0,13\cdot\ln(w)+0,54)-\right. \\ \left.(6,95\cdot\ln(w)+24,33)\right)\mu W & ,V = LV, CFH = random \\ \left((316,4+118,9\cdot w)\cdot\dfrac{276,0+0,20\cdot b\cdot w}{4160,6}-\right. \\ \left.(19,23+7,37\cdot w)\right)\mu W & ,L = WH, I = Reg \\ \left((316,4+118,9\cdot w)\cdot\dfrac{487,0+3,58\cdot b\cdot w}{4160,6}-\right. \\ \left.(274,7+105,3\cdot w)\cdot(0,068+0,029\cdot b)\right)\mu W & ,L = WH, I = Std \end{cases} \quad \text{(D.19)}$$